Otto-von-Guericke University Magdeburg

Software Engineering Department

Interdisciplinary Team Project

Automated spline trajectory planning for offline robot programming

Authors:
Dheeraj Tippani, Sravani Dhara, Kiran Babu

April 7, 2022

Advisers:

Supervisor
M.Sc.Nadia Schillreeff

Department of Computer Science
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Supervisor
Prof. Dr. rer. nat. Frank Ortmeier

Department of Computer Science
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Contents

Abstract

For a robotic arm motion planning, one of the most important step is trajectory planning. The end effector of the robotic arm will be made to follow the planned trajectory. To be able to instruct the robot to follow complex trajectories one needs to understand how the robot's internal path planning function works. Specifically, we are interested in the spline block function of the KUKA robot. The test subject in this study is the KUKA iiwa7 R800 robotic arm. Aim of the project is be able to model a spline function to generate a spline, that can match with the spline motion that the official KUKA robotic controller computes and commands the robot to execute. In an attempt to understand the nature of KUKA robot's spline function, several experiments were conducted. With the data collected in several experiments and attempting several approaches, we have concluded by modeling a spline/trajectory generator that is optimised to come as close as possible to the KUKA robot's spline block function.

Acknowledgements

Our team would like to express deepest gratitude to our supervisor Nadia Schillreff for her extensive support at every step of this project. She has put tremendous effort to guide us in the right direction and gave us insightful information. We thank her for providing us necessary facilities that were required for the completion of project.

1 Introduction and Motivation

1.1 Motivation

Robotic arms are used for numerous industrial applications and automation. A robot has to perform several operation during its operation, such as, picking up objects, collision detection, path planning, motion planning, safety monitoring, precision imperative jobs etc. Path planning is one of the most crucial tasks of a robotic arm. In industrial application programming a robot to follow a certain trajectory is traditionally done online programming fashion. Meaning, a robot should be taken off production for the engineer to teach the robot a new set of instructions or trajectories. This is a serious problem when the industrial robot whose down time can become very expensive. Relatively recent development of robot simulation softwares help in alleviating this problem by using Online-programming methods. In 'offline programming' the engineer does not have to take the robot off the production pipeline in order to feed new set of instructions, nor does one require a live robot to test a new set of commands. These steps can be performed in a virtual robot simulation environment and get tested several times before feeding the commands to the actual robot.

Once the engineer is sure of the commands to feed into the robot, the downtime is relatively negligible since the uploading of new commands does not have to be physical and manual, it can completely be remote. Such a convenience opens up new opportunities and new applications for the user to develop.

One such envisioned application is the development of a automated spline trajectory planning for offline programming based applications.

One can imagine a scenario where user wants a robot to follow a welding seam which is not exactly a straight line path in 3D space. A path that is not a straight line poses a lot of complex problems during robot programming, especially in offline programming. Our aim is to make the robot go through a complex curved path in 3D space.

To know how to instruct a robot to follow a curved line paths, we need to know how the robot computes a spline path given a list of way points. List of points that the robot has to go through by generating a smooth spline like continuous path in 3D space.

In a effort to automate the process of spline trajectory planning, we attempted to understand and model the hidden 'spline block' function of a famous lightweight industrial robot arm, KUKA iiwa 7 r800. In this project, we implemented an algorithm that can generate a spline trajectory that comes very close to the 'spline block' function of the KUKA robot.

1.2 Structure of this report

Chapter 2 will give an overview of Forward Kinematics, Inverse kinematics and their parameters to help plan feasible robotic arm motion. We also discuss how we have performed these operation and how is it useful in this project. We also included a brief description of the KUKA iiwa7 arm robot, its specifications and important components. It also covers an outline of splines, their characteristics and types.

In Chapter 3, A detailed explanation of the Experiment setup, parameters and configuration involved in the experiment for collecting the KUKA robot path information to understand its path planning better. We tried to give a clear reasoning for our choices and steps taken as detailed as possible for better understanding.

In Chapter 4 we discuss the initial analysis work done in the experiment data from the KUKA robot. We have tried several approaches to understand as much as possible about the KUKA robot's 'spline block' function. We discuss in detail our results from each of our analysis steps and how it led to our next step of analysis.

Chapter 5 is about generating the robotic arm path trajectory with just the input way points. We have conformed to Quintic B'ezier spline to replicate/model KUKA's 'spline block' function.

Chapter 6 Final chapter deals with the optimisation algorithm and respective optimised control variables. This is an extension of chapter 5 but also deserves an independent chapter to discuss the choice of optimisation variables, systematic scoring algorithm used, optimisation technique used to find the most effective unique solution.

2 Basic Introduction

This chapter is meant to give a brief introduction to all the prerequisites for the following chapters. We are aiming to cover as much information as needed to clearly understand this report and following chapters. In the following sections, a brief introduction to robot kinematics, and the methods that we have used in this project will be given. This includes the minimum information needed to perform Forward kinematics and Inverse kinematics, how it is useful in this project and how they are implemented. This chapter also includes information about KUKA iiwa7 industrial robotic arm, how we have used it in the project, necessary components that are involved in the scope of this project, limitations of the robot. ended the chapter with a description of 'Splines', what they are and how they are related to everything else, types of splines and what the boundary conditions that differentiate between types of splines.

2.1 Forward Kinematics

Any robotic arm is built of certain common basic components. These components are

- Base of the robot
  - The base is part of the robotic arm that is attached to a fixed ground. Every other component of the robot are attached to the base of the robot. The base also acts as the base co-ordinate frame of reference, with respect to which the position of every other component of the robot will be defined.
- Links

  - The links are bunch of rigid bodies that are connected to each other at joints. A chain of such kinematic links forms the body of the robotic arm. The very 1st link being connected to the base of the robot and the 2nd link, 2nd link being connected to the first link and the third link, and so on, until the end effector/TCP of the robot.

- Joints

  - The connection points between any two links (base and end effector are also considered as links) is called a joint. Links of the robot rotate with respect to each other to make the end effector reach any point in 3D space.

- End effector or TCP (ToolCenterPoint)

  - It is the furthest tip of the robotic arm, Which interacts with the outside world through a device, for example, a gripper or a welding torch.

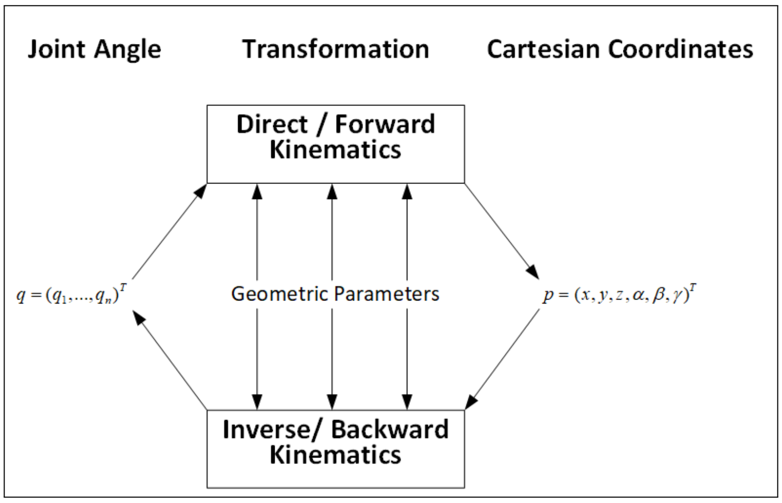

Joint Angle
Transformation
Cartesian Coordinates
Direct / Forward Kinematics
$q=(q_1,\ldots,q_n)^T$
Geometric Parameters
$p=(x,y,z,\alpha,\beta,\gamma)^T$
Inverse/ Backward Kinematics

Figure 2.1: Forward kinematics and Inverse kinematics relationship. Image courtesy:`https://en.wikipedia.org/wiki/Forward_kinematics`

Given we know the joint angles information, Forward kinematics is all about computing the position of the end effector in 3D space, refer to fig 2.1. This computation is represented by a transformation matrix, which contains all the information about the position and rotation of the end-effector with respect to the base co-ordinate frame of reference of the robot. A transformation matrix is built is by multiplying the transformation of (i+1)$^{\text{th}}$ joint's reference frame with respect to (i)$^{\text{th}}$ joint's reference frame, and doing this from the base joint to till the end effector, as depicted in fig 2.3.

$$ {}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_{i,i+1} & \sin\theta_i\sin\alpha_{i,i+1} & a_{i,i+1}\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_{i,i+1} & -\cos\theta_i\sin\alpha_{i,i+1} & a_{i,i+1}\sin\theta_i \\ 0 & \sin\alpha_{i,i+1} & \cos\alpha_{i,i+1} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Figure 2.2: DH transformation matrix. Image courtesy: `https://en.wikipedia.org/wiki/Forward_kinematics`

- 'd' represents the depth difference between (i+1)$^{\text{th}}$ joint reference frame and (i)$^{\text{th}}$ joint reference frame, along (i)$^{\text{th}}$ joint's z axis (The axis of rotation)
- 'theta' represents the angular difference between (i)$^{\text{th}}$ joint's x axis and (i+1)$^{\text{th}}$ joint's x axis, along $^{\text{th}}$th joint's z axis
- 'α' represents the angular difference between (i)$^{\text{th}}$ joint's z axis and (i+1)$^{\text{th}}$ joint's z axis, along (i+1)$^{\text{th}}$ joint's x axis
- 'a' represents perpendicular distance between (i)$^{\text{th}}$ joint and the (i+1)$^{\text{th}}$ joint along x axis of the (i)$^{\text{th}}$

In fig 2.2, DH Transformation matrix is a 4x4 matrix, of which, first 3x3 elements correspond to the rotation of (i+1)$^{\text{th}}$ joint with respect to i$^{\text{th}}$ joint, and, 4$^{\text{th}}$ column in the transformation matrix corresponds to 'translation' of (i+1)$^{\text{th}}$ joint with respect to i$^{\text{th}}$ joint.

The transformation matrix in the image above is applied to all the joint pairs starting from base to end effector, in the following fashion shown in fig 2.3

In this project, to understand how the robot creates a spline path from a list of way points, we performed certain set of 'experiments' where we made the robot go in a spline path to go through predefined way points.

$$\boxed{{}^{0}\mathbf{T}_{7} = {}^{0}\mathbf{T}_{1}{}^{1}\mathbf{T}_{2}{}^{2}\mathbf{T}_{3}{}^{3}\mathbf{T}_{4}{}^{4}\mathbf{T}_{5}{}^{5}\mathbf{T}_{6}{}^{6}\mathbf{T}_{7}}$$

Figure 2.3: Chain of transformation matrices from first joint pair until last joint pair. DENAVIT and HARTENBERG (1955)

The output from the experiment is in the form of joint angles recorded at around 100 data points per second. This output from the robot has to be converted into 3D Cartesian coordinate system, so that we can analyse the trajectories. This is where we had to use 'Forward kinematics'.

The task of performing forward kinematics is standardized and simplified by the usage of Denavit Hartenberg convention of assigning co-ordinate axes at each of the joints of the robot. image shown above depicts a Transformation matrix generated using DH parameters.

2.2 Inverse Kinematics

As shown in fig1, the task of figuring out joint configurations given the position of the end effector is called Inverse kinematics. In this project, in order to instruct the KUKA robot to go to Cartesian points that we had to feed the robot the associated joint angles. There are other ways to instruct the robot to go through our predefined list of way points, but we chose to do it by the way of converting Cartesian points into joint angle configurations. The reasoning for this is discussed further detail in Chapter 3.

For a robot with 7 degrees of freedom(dof), the number of possible solutions are infinite. To address this problem we have replicated the implementation in FARIA et al. (2018). When the number of known values are 6, 'x, y, z, α,β, γ', and the number of unknown values are same, i.e, 6 joint angles of a 6-dof robot, then there exists a unique solution for the robot's joint configurations. But, when the number of unknown values are more than 6, i.e, the 7 joint angles of a 7-dof robot, then there can be multiple solution/multiple ways to make the end effector reach the required "x, y, z, α,β, γ" configuration.

In this way, a 7-dof robot is more advantageous when compared to a 6-dof robot, because, this additional set of possible solutions to achieve same goals can be useful in collision avoidance in tight space maneuvers, also in singularity avoidance.

2.3 KUKA iiwa7

KUKA iiwa 7 is a 7 degrees of freedom robot. It is a lightweight robotic arm of around 24 kg with a payload capacity of up to 7 kg, it is designed for applications in industrial automation. The robot has a maximum reach of 0.8 meter in all directions.KUKA

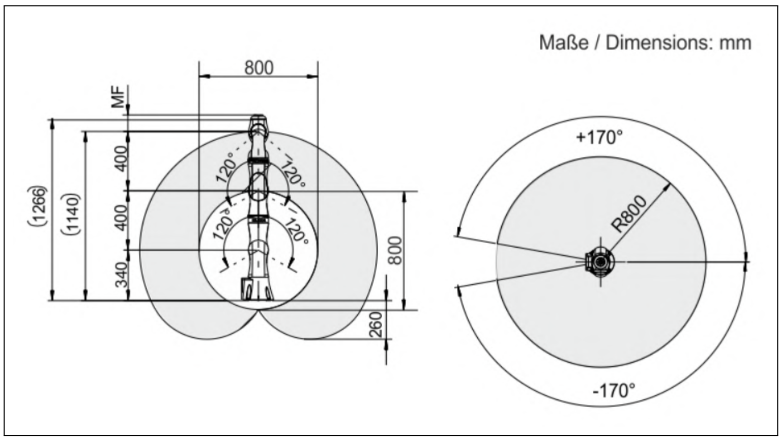


Figure 2.4: KUKA iiwa 7 r800 robot's work envelope

KUKA robotic systems consists of primarily three components,

- The robotic arm/ Manipulator
    - Here we are referring to both the 'end effector' and 'robot' itself.
- KUKA SUNRISE ROBOT CONTROLLER (KRC)
    - The hardware necessary for communicating with the KUKA robot. This is run by 'KUKA Sunrise OS'. It is also home for 'KUKA Sunrise workbench software' which is responsible for most important operations like safety, path planning, motion planning etc.
- KUKA smartPAD control panel.

  - The hardware necessary to give commands to the robot during production. User can control the robot's movement, execute computed trajectories etc.,

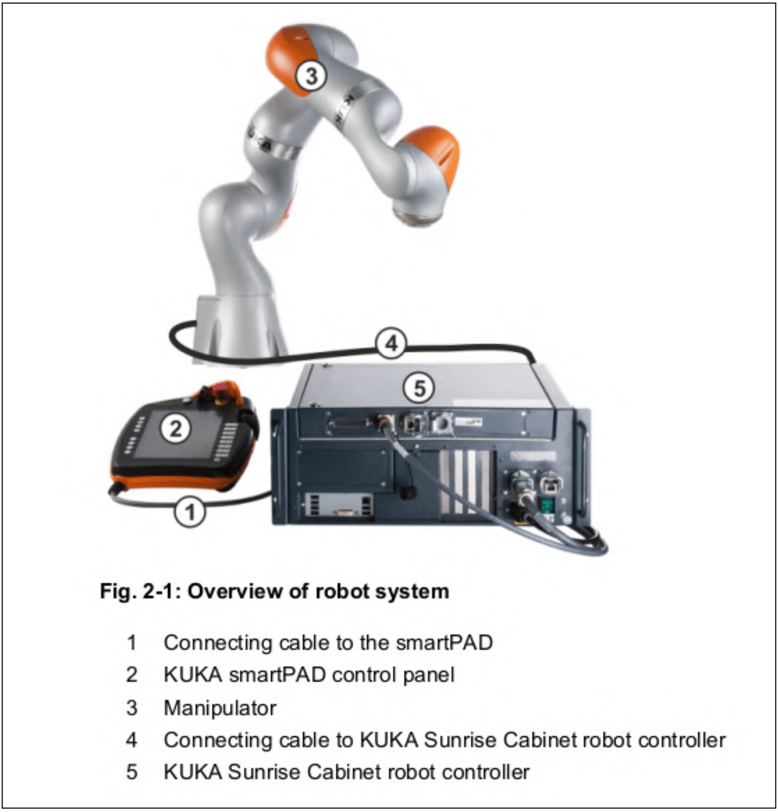


Fig. 2-1: Overview of robot system

1	Connecting cable to the smartPAD
2	KUKA smartPAD control panel
3	Manipulator
4	Connecting cable to KUKA Sunrise Cabinet robot controller
5	KUKA Sunrise Cabinet robot controller

Figure 2.5: KUKA robot essential components

What do we know about KUKA?:

The robot can take a series of way points as input to create a spline path.

What do we want to know about KUKA?:

We want to know how KUKA generates a spline path using the input way points.

Why do we want to know?:

So that, given we have some curved line first, we can find a combination of way points which can make KUKA follow our curved line.

we can find an inverse function of KUKA's spline function, that can generate way points from the input spline that, which can be fed into real world robot to make it pass through the way points, consequently making it follow the curve we are interested in.

2.4 Splines

Splines are piece-wise polynomial parametric curves i.e, splines are combination of multiple small pieces formed by the input points. A pair of points successively make a segment which is a piece of spline. Spline piece possess an internal rule guided polynomial function. Various degrees of polynomial form types of splines which are discussed in detail in the section 2.3.2. Spline has parameters using which the final curve can be manipulated.

2.4.1 Characteristics of Splines

Continuity

In terms of curvature matching splines can be defined with respect to their continuity levels. C_0, C_1 and C_2 are possible levels. A spline curve is said to be C_0 Continuity when the two segments at point of join have same values. When the two segments exhibit same slope at point of joining they would be C_1 Continuity. When they possess same curvature at the joining point they are said to be in Continuity C_2. C_2 ensures smooth curve. We need C2 to reduce wear and tear of robot parts because c2 will have minimal jerk.

Interpolation and Approximation

Splines polynomial can go through all the input points given or just reach the neighborhood of the input points. In interpolation the spline visits each and every point specified as input. However in approximation, spline need not mandatorily visit each point. Interpolation is better for robot scenario because it visits the target locations.

Localism and Globalism

When there is a change in the spline segment if it affects just the neighboring segments then the spline is said to have localism. While globalism being the property of a spline where the entire spline will be effected with a change in at least one point in the way points.

2.4.2 Types of Splines

B-spline

B-spline can be defined by the control points P_0 to P_m and each segment of a B-spline is a cubic polynomial. The i^{Th} segment consists of P_i to P_{i+3} as it's control points by using below spline equation

$$\begin{aligned} s_i(t) = \frac{1}{6}(-P_i + 3P_{i+1} - 3P_{i+2} + P_{i+3})\,t^3 + \frac{1}{6}(3P_i - 6P_{i+1} + 3P_{i+2})\,t^2 \\ + \frac{1}{6}(-3P_i + 3P_{i+2})\,t + \frac{1}{6}(P_i + 4P_{i+1} + P_{i+2}) \end{aligned}$$

Cubic B´ezier

As the name suggests it is a 3 degree polynomial B´ezier curve. It is represented by the equation 2.1

$$B(t) = (1-t)^3 P_0 + (3-t)^2 P_1 + (3-t)^2 P_2 + t^3 P_3 \tag{2.1}$$

Variable t represents the time step and varies between 0 and 1. Equation 2.1 is used to calculate each segment of spline trajectory. P_0, P_1,P_2 and P_3 are the control points of each spline segment. P_1 and P_3 are the input start and end way points of the current segment i.e, while the input spline points are P_A, P_B, P_C ,P_D, P_E then while calculating the spline segment between P_A and P_B the values of P_1 and P_3 would be P_A and P_B respectively. And when calculating spline segment between P_D, P_E the values of P_1 and P_3 would be P_D, P_E respectively. While calculating the spline segment between i and i+1 point the co-efficient of Cubic B´ezier are:

$$P_0 = P_i \tag{2.2}$$

$$P_1 = P_0 + \frac{1}{3} * Tangent_i \tag{2.3}$$

$$P_2 = P_3 - \frac{1}{3} * Tangent_{i+1} \tag{2.4}$$

$$P_3 = P_{i+1} \tag{2.5}$$

Since there is only first derivative(tangent) involved in cubic B´ezier curve it exhibits C_1 continuity. P_1 and P_2 determine the direction of spline segment. Tangent of cubic spline can be calculated by differentiating equation 2.1

Quintic B´ezier

It is a 5 degree polynomial B´ezier curve which is similar to cubic B´ezier curve but has second derivative involved in computation. Thus ensuring C2 continuity. It has 6 control points in total. This spline type is clearly explained in Section 5.2.

3 Experiment setup and Data Acquisition

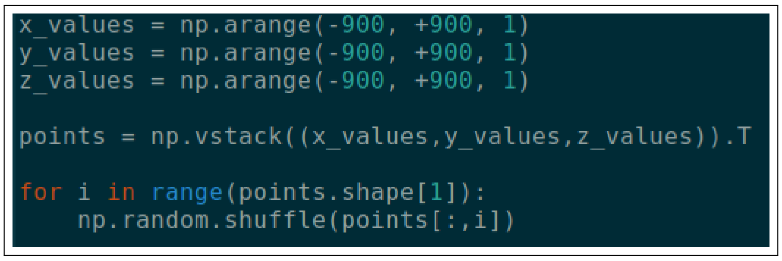
The goal of the project is to understand and replicate KUKA's path planning algorithm behind the spline function. To understand this behaviour, some preliminary set of experiments had to be conducted to extract path information from the robotic arm, so that KUKA's path planning can be understood better. The process of start of building an experiment setup and data acquisition happens in the following chronology. each step will be discussed briefly in the following sections.
Generate points -> Convert points to joint configurations (IK) -> Filter out unreachable points -> Formulate a set of spline blocks for robot programming -> Collect joint angles during robot's motion through the points we chose.

3.1 Point Generation

As established, First step of building an experiment setup is to have a set of points through which we want KUKA to go through. we refer to such points as way points. Way points will be used in designing the experiments in the later stages which will be discussed in detail in Chapter 4. way points can be either randomly generated or chosen using some known function.

3.1.1 Random points

With respect to the robot's base frame of reference, we just have to generate some Cartesian 3d points with respective (x, y, z) values. Robotic arm's maximum length is taken as a limit. Each point is a combination of three coordinates, x, y and z. Each coordinate is an array with 1800 sampled values, ranging from -900 to +900. Each coordinate array is shuffled independent of each other. The arrays are then combined together to form 1800 random points inside a cube with diagonal extremes being (-900,-900,-900) and (+900, +900,+900)

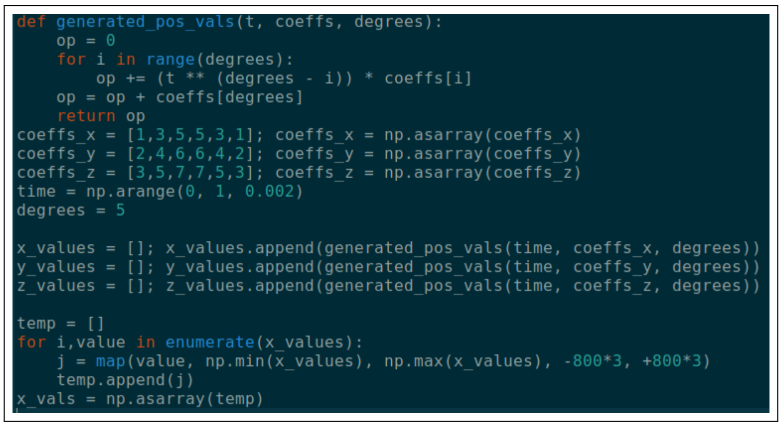


```python
x_values = np.arange(-900, +900, 1)
y_values = np.arange(-900, +900, 1)
z_values = np.arange(-900, +900, 1)

points = np.vstack((x_values,y_values,z_values)).T

for i in range(points.shape[1]):
    np.random.shuffle(points[:,i])
```

Figure 3.1: Random way-points generation

3.1.2 Sampling from arbitrary 5-degree polynomial

We wanted to observe if we generate way points from a Quintic polynomial with simple set of coefficients, in hope of observing similar coefficients when we interpolate the trajectory that robot has executed. We have formulated three 5-degree polynomials, each with 6 arbitrarily chosen coefficients. The 5-d polynomials correspond to behaviour of the axes x, y and z. The aim is to generate a list of at least 500 points, where each coordinate value is a sample from the respective 5-d polynomial, i.e, one polynomial per co-ordinate axis.



```python
def generated_pos_vals(t, coeffs, degrees):
    op = 0
    for i in range(degrees):
        op += (t ** (degrees - i)) * coeffs[i]
    op = op + coeffs[degrees]
    return op
coeffs_x = [1,3,5,5,3,1]; coeffs_x = np.asarray(coeffs_x)
coeffs_y = [2,4,6,6,4,2]; coeffs_y = np.asarray(coeffs_y)
coeffs_z = [3,5,7,7,5,3]; coeffs_z = np.asarray(coeffs_z)
time = np.arange(0, 1, 0.002)
degrees = 5

x_values = []; x_values.append(generated_pos_vals(time, coeffs_x, degrees))
y_values = []; y_values.append(generated_pos_vals(time, coeffs_y, degrees))
z_values = []; z_values.append(generated_pos_vals(time, coeffs_z, degrees))

temp = []
for i,value in enumerate(x_values):
    j = map(value, np.min(x_values), np.max(x_values), -800*3, +800*3)
    temp.append(j)
x_vals = np.asarray(temp)
```

Figure 3.2: Point generation from arbitrary 5-d Polynomial

5 degree - robot forum

The reason we have started our analysis of the KUKA robot's trajectory with a 5-degree polynomial is because during the literature survey, in a German forum called Robot-forum we have learnt that the KUKA robots spline block function implements some 5 degree polynomial for path planning.The respective statement is made by official developer of the KUKA robot controller(KRC) system. Although this is unofficial in nature, we believed that this could be a reasonable start point to our analysis.

3.2 Cartesian to Configuration

We can program the KUKA robot to go through our defined points in two ways.

1. input: position(x, y, z), angles about Z, Y and X respectively(A,B,C), Status and Turn.
2. input: Joint configurations at each points.

To obtain status, turn and Euler angles, from the robot, we need to feed in the joint configurations first. If we have the joint configurations, we will not be needing to compute status, turn and Euler angles anymore. So, we chose to input KUKA robot about the way points using the 2$^{\text{nd}}$ option.

We have to use Inverse kinematics techniques that we've discussed in the 2nd Chapter, to find all the possible joint configurations that would guarantee that the robot's end effector will point at the respective way point. We have implemented inverse kinematics(IK) method using method developed in SPRUNK (2008), for a 7 dof robotic arm. As a result, for each point we obtain a set of possible configurations of joint angles.

3.3 Configuration check - robot specific

In order to reduce the number of points and number of solutions per each point we developed two filtering steps to remove redundant information.

3.3.1 Reachability

There is no limit to how many points we can sample from a given volume in 3d space. In order to limit the number of samples, we have not considered any points with decimals in the x,y,z, coordinate values. Another important criteria is to check if the robot is capable of reaching the selected point. In the section 3.1.1 we have discussed that the volume from which points are sampled is a cube of 1800 (centimeters) in side length. But the KUKA robot cannot reach most of the points inside this cube volume. The work envelope volume of KUKA around it's base is well visualised in the image shown here.

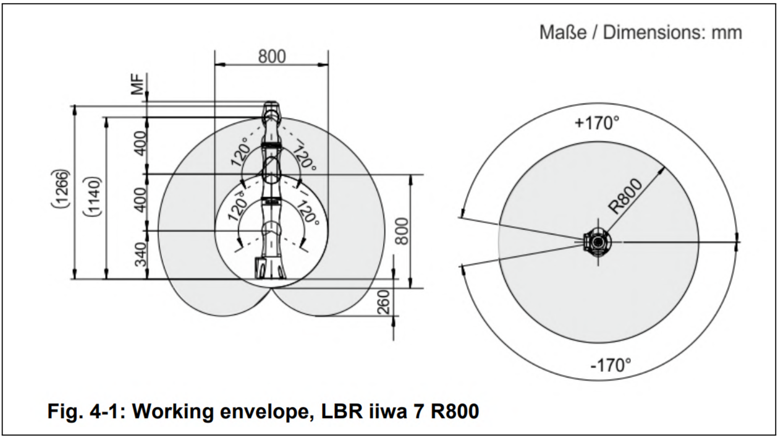
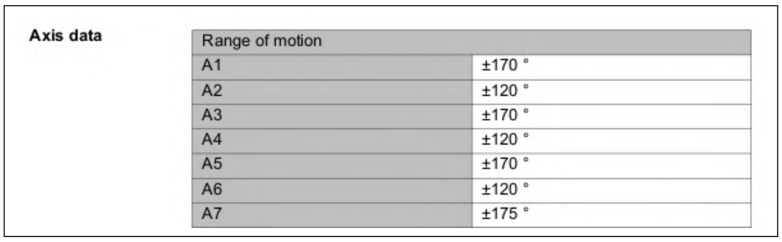


Figure 3.3: Work envelope of KUKA iiwa7 R800 robot.

3.3.2 Limit avoidance

After filtering out the points that are not in the working envelope of the robot, using inverse kinematics technique as presented by Faria et al. (2018), we have obtained set of joint configurations for each of the sampled points. But we have not yet taken the limitations of joint angles into consideration yet. In this step, we use the joint angle limitations of KUKA iiwa7 R800 robot **?** to further filter out the redundant configurations.


Axis data

Range of motion	
A1	±170 °
A2	±120 °
A3	±170 °
A4	±120 °
A5	±170 °
A6	±120 °
A7	±175 °

Figure 3.4: Limitations of joint angles of the Robot

3.4 Robot Program

After the previous step we have all the necessary information to encode the experiment in a language that robot can understand, which is KUKA Robot Language(KRL). Here we declare the way points in the form of joint configurations. In the following sections we discuss about the parameters needed to run the experiment and a closer look at a snippet of robot program.
The following are the input we have to provide the robot in advance in the programs.

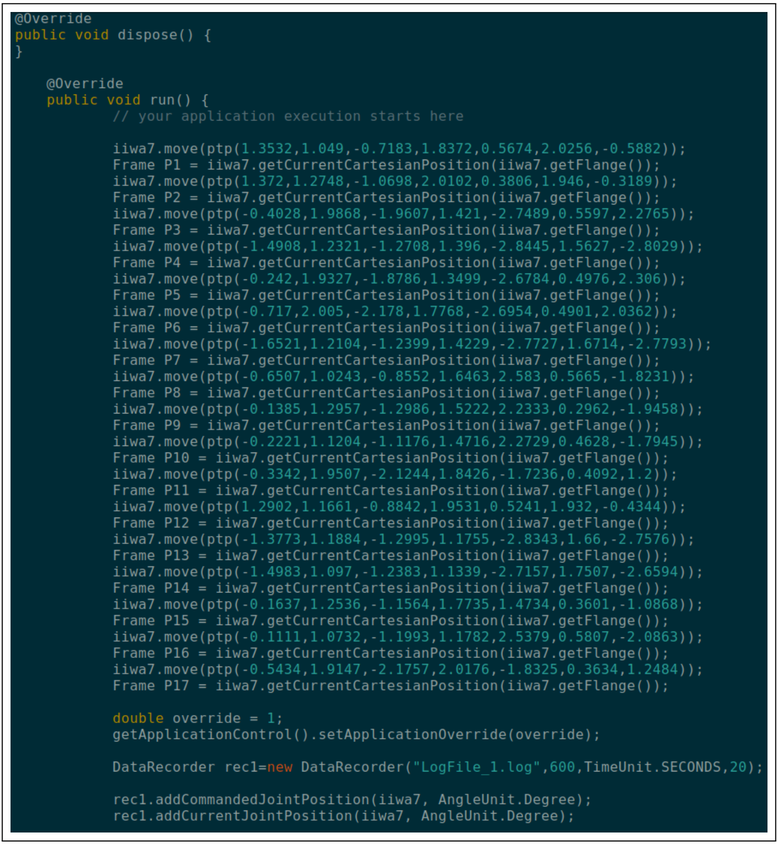
- Joint configurations of the way points
- Rate of data-logging and Units of joint angles

3.4.1 Program snippet

The following is a snippet of the robot program. The following is the snippet of how the experiments are formulated in the robot program.

3.5 Data Recording

After the previous step, the robot will execute all the trajectories that are designed in the experiment, design of the experiments will be discussed in detail in Chapter 4. Once the robot starts executing the code, end effector start from home position and starts to pass through the way points in the order of their occurrence in the robot program.



```
@Override
public void dispose() {
}

    @Override
    public void run() {
            // your application execution starts here

            iiwa7.move(ptp(1.3532,1.049,-0.7183,1.8372,0.5674,2.0256,-0.5882));
            Frame P1 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(1.372,1.2748,-1.0698,2.0102,0.3806,1.946,-0.3189));
            Frame P2 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.4028,1.9868,-1.9607,1.421,-2.7489,0.5597,2.2765));
            Frame P3 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-1.4908,1.2321,-1.2708,1.396,-2.8445,1.5627,-2.8029));
            Frame P4 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.242,1.9327,-1.8786,1.3499,-2.6784,0.4976,2.306));
            Frame P5 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.717,2.005,-2.178,1.7768,-2.6954,0.4901,2.0362));
            Frame P6 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-1.6521,1.2104,-1.2399,1.4229,-2.7727,1.6714,-2.7793));
            Frame P7 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.6507,1.0243,-0.8552,1.6463,2.583,0.5665,-1.8231));
            Frame P8 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.1385,1.2957,-1.2986,1.5222,2.2333,0.2962,-1.9458));
            Frame P9 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.2221,1.1204,-1.1176,1.4716,2.2729,0.4628,-1.7945));
            Frame P10 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.3342,1.9507,-2.1244,1.8426,-1.7236,0.4092,1.2));
            Frame P11 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(1.2902,1.1661,-0.8842,1.9531,0.5241,1.932,-0.4344));
            Frame P12 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-1.3773,1.1884,-1.2995,1.1755,-2.8343,1.66,-2.7576));
            Frame P13 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-1.4983,1.097,-1.2383,1.1339,-2.7157,1.7507,-2.6594));
            Frame P14 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.1637,1.2536,-1.1564,1.7735,1.4734,0.3601,-1.0868));
            Frame P15 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.1111,1.0732,-1.1993,1.1782,2.5379,0.5807,-2.0863));
            Frame P16 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());
            iiwa7.move(ptp(-0.5434,1.9147,-2.1757,2.0176,-1.8325,0.3634,1.2484));
            Frame P17 = iiwa7.getCurrentCartesianPosition(iiwa7.getFlange());

            double override = 1;
            getApplicationControl().setApplicationOverride(override);

            DataRecorder rec1=new DataRecorder("LogFile_1.log",600,TimeUnit.SECONDS,20);

            rec1.addCommandedJointPosition(iiwa7, AngleUnit.Degree);
            rec1.addCurrentJointPosition(iiwa7, AngleUnit.Degree);
```

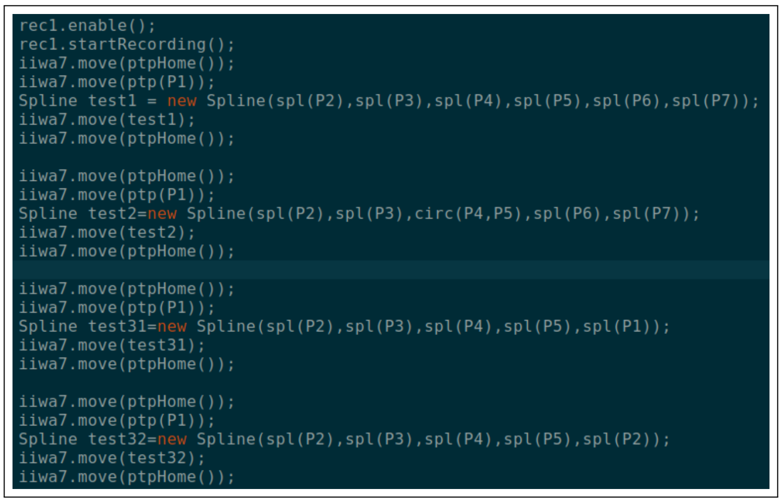
Figure 3.5: Robot program for point declaration

3.5.1 Recorder rate

The robot will start recording the moment robot begins to execute the code. In the robot program, rate of logging the joint angle information is declared. We declare the number the log points should be recorder in a second. We have logged at the rate of 20 to 50 data points per second, i.e, once every 50 - 20 milliseconds.

3.5.2 Format - joint angles

Joint angles information at regular time intervals is recorded during the motion of the robot. The raw output datalog looks as shown in the image below. The columns correspond to following information:



```
rec1.enable();
rec1.startRecording();
iiwa7.move(ptpHome());
iiwa7.move(ptp(P1));
Spline test1 = new Spline(spl(P2),spl(P3),spl(P4),spl(P5),spl(P6),spl(P7));
iiwa7.move(test1);
iiwa7.move(ptpHome());

iiwa7.move(ptpHome());
iiwa7.move(ptp(P1));
Spline test2=new Spline(spl(P2),spl(P3),circ(P4,P5),spl(P6),spl(P7));
iiwa7.move(test2);
iiwa7.move(ptpHome());

iiwa7.move(ptpHome());
iiwa7.move(ptp(P1));
Spline test31=new Spline(spl(P2),spl(P3),spl(P4),spl(P5),spl(P1));
iiwa7.move(test31);
iiwa7.move(ptpHome());

iiwa7.move(ptpHome());
iiwa7.move(ptp(P1));
Spline test32=new Spline(spl(P2),spl(P3),spl(P4),spl(P5),spl(P2));
iiwa7.move(test32);
iiwa7.move(ptpHome());
```

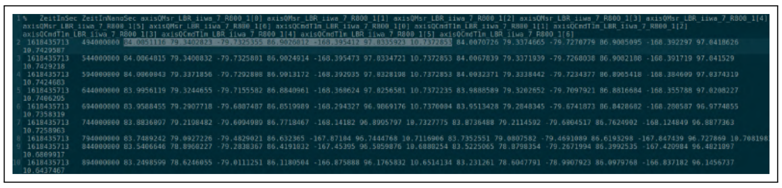
Figure 3.6: Robot program for experiments

- 1st and 2nd column: Time in seconds and nanoseconds from the start of the execution of program.

- 3rd to 9th columns:Joint angles 'Measured' by internal precision sensors during motion, with respect to robot base frame of reference, from the 1st-Joint to 7th-joint at every time step.

- 10th to 16th columns:Joint angles 'Commanded' by the robot controller prior to motion, with respect to robot base frame of reference, from the 1st-Joint to 7th-joint at every time step.

Note that there is slight difference between the "intended joint angle(Commanded)" and "achieved joint angle(Measured)".



```
%   ZeitInSec ZeitInNanoSec axisQMsr_LBR_iiwa_7_R800_1[0] axisQMsr_LBR_iiwa_7_R800_1[1] axisQMsr_LBR_iiwa_7_R800_1[2] axisQMsr_LBR_iiwa_7_R800_1[3] axisQMsr_LBR_iiwa_7_R800_1[4]
axisQMsr_LBR_iiwa_7_R800_1[5] axisQMsr_LBR_iiwa_7_R800_1[6] axisQCmdT1m_LBR_iiwa_7_R800_1[0] axisQCmdT1m_LBR_iiwa_7_R800_1[1] axisQCmdT1m_LBR_iiwa_7_R800_1[2]
axisQCmdT1m_LBR_iiwa_7_R800_1[3] axisQCmdT1m_LBR_iiwa_7_R800_1[4] axisQCmdT1m_LBR_iiwa_7_R800_1[5] axisQCmdT1m_LBR_iiwa_7_R800_1[6]
 1618435713   494000000 84.0051116 79.3402823 -79.7325355 86.9026812 -168.395412 97.0335923 10.7372853 84.0070726 79.3374665 -79.7270779 86.9085095 -168.392297 97.0410626
10.7429587
 1618435713   544000000 84.0064815 79.3400832 -79.7325881 86.9024914 -168.395473 97.0334721 10.7372853 84.0067839 79.3371939 -79.7268038 86.9082188 -168.391719 97.041529
10.7429218
 1618435713   594000000 84.0060843 79.3371856 -79.7292888 86.9013172 -168.392935 97.0328198 10.7372853 84.0032371 79.3338442 -79.7234377 86.8965418 -168.384609 97.0374319
10.7424683
 1618435713   644000000 83.9956119 79.3244655 -79.7155582 86.8840961 -168.368624 97.0256581 10.7372235 83.9888589 79.3202652 -79.7097921 86.8816684 -168.355788 97.0208227
10.7406295
 1618435713   694000000 83.9588455 79.2907718 -79.6887487 86.8519989 -168.294327 96.9869176 10.7370084 83.9513428 79.2848345 -79.6741873 86.8428682 -168.208587 96.9774855
10.7358319
 1618435713   744000000 83.8836897 79.2198482 -79.6094989 86.7718467 -168.14182 96.8995797 10.7327775 83.8736488 79.2114592 -79.6004517 86.7624902 -168.124849 96.8877363
10.7258963
 1618435713   794000000 83.7489242 79.0927226 -79.4829021 86.632365 -167.87184 96.7444768 10.7116906 83.7352551 79.0807582 -79.4691089 86.6193298 -167.847439 96.727869 10.7081987
 1618435713   844000000 83.5406646 78.8968227 -79.2838367 86.4191832 -167.45395 96.5859876 10.6888254 83.5225065 78.8798354 -79.2671994 86.3992535 -167.420984 96.4821097
10.6809917
 1618435713   894000000 83.2498599 78.6246055 -79.0111251 86.1180504 -166.875888 96.1765832 10.6514134 83.231261 78.6047791 -78.9907923 86.0979768 -166.837182 96.1456737
10.6437467
```

Figure 3.7: Raw data log file looks as depicted here

3.5.3 Post Processing

This is the final and easy part of the data acquisition. The raw data log file contains information about the joint angles at a very high frequency, from start of the execution to the end of the execution. Data acquisition is complete when we get position of the end effector in Cartesian co-ordinate system, at every time step of the experiments. So, here we perform forward kinematics to obtain the position of the robot's end effector.

After the conversion of joint angles into end-effector position, only remaining step is to label the way points and home position in the output data log points. We have done this manually, but this can also be done automatically with a few simple lines of code.

4 Experiment

The primary goal of the project is to understand how the KUKA robotic controller generates a spline path given a certain number of way points in the spline block. To know how KUKA plans its movement through the way-points, we have designed set of experiments to extract the trajectory information from the robot under varying combinations of way-points in the spline block. In the following sections we discuss about the experiments, results and conclusions.

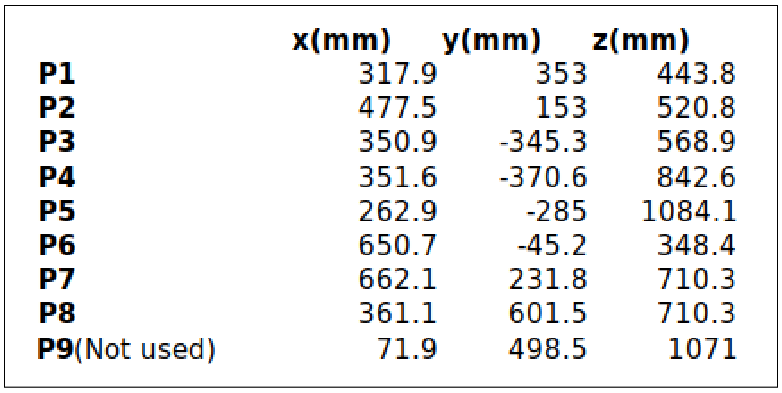
4.1 Understanding robot path - using Polynomial interpolation

In order to obtain trajectory information we conducted experiments with some random points as way points. Now we have a list of 3-d points that the robot's end effector passed through, with certain predefined time difference between each data point.

4.1.1 Assumptions

- Our first assumption was that the robot's spline function is a quintic polynomial. We have previously discussed the reasoning for this assumption in the Chapter 3.1.
- Second assumption is that the robot controller is computing a quintic polynomial independently for different axes, i.e, each spline is defined by three quintic polynomials.

The following fig 4.1 is the list of way points around the robot.



	x(mm)	y(mm)	z(mm)
P1	317.9	353	443.8
P2	477.5	153	520.8
P3	350.9	-345.3	568.9
P4	351.6	-370.6	842.6
P5	262.9	-285	1084.1
P6	650.7	-45.2	348.4
P7	662.1	231.8	710.3
P8	361.1	601.5	710.3
P9(Not used)	71.9	498.5	1071

Figure 4.1: List of Way-points

The following is the list of experiments, each with varying configurations of way points.

- h - p1 - s(p2 p3 p4 p5 p6 p7) - h
- h - p1 - s(p2 p3 p4 p5) l(p6) s(p7) - h
- h - p1 - s(p2 p3 p4 p5) c(p6 p7) - h
- h - p1 - s(p2 p3 p8 p5 p6 p7) - h
- h - p1 - s(p2 p3 p8 p8) c(p6 p7) - h
- h - p1 - s(p2) - h
- h - p1 - s(p2 p3) - h
- h - p1 - s(p2 p3 p4) - h
- h - p1 - s(p2 p3 p4 p5) - h

Where 'h' represents Home position of the KUKA robot. 'Home' position of robot end effector is (0,0,1266) in 3D Cartesian space, units being milli meters.

A spline block and corresponding way points inside the spline block are written as "S(list of way points)" in KUKA robot language(KRL). When the robot executes the code, it starts the spline block right from p1, i.e, one point before the first point inside of the spline block. Hence, when we consider performing an polynomial interpolation, we better take the point prior to start of spline block also into consideration.

Visualisation of all the experiment in 3D space is as in fig 4.2. Given our

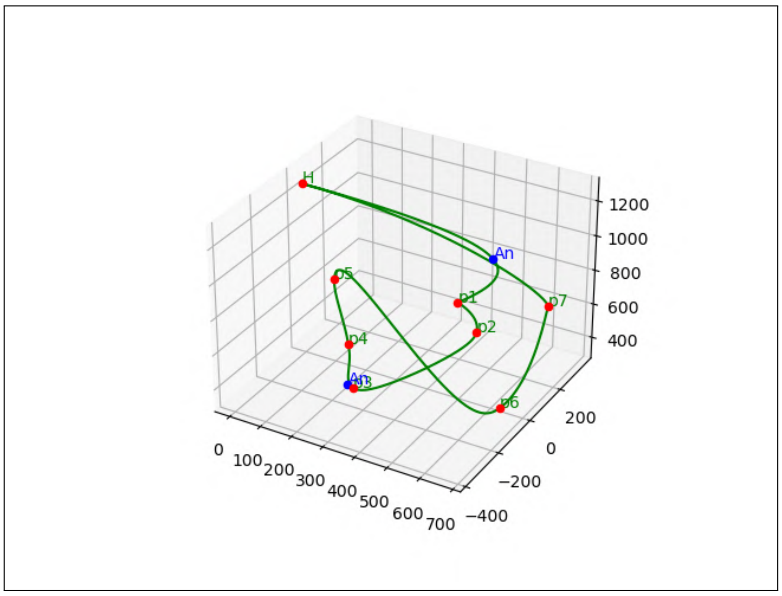


Figure 4.2: Visualization of one of the trajectories (H_s1234567_H) from the set of robot experiments

assumption that robot's spline function is a quintic polynomial, we used least squares polynomial interpolation method to define the robot's trajectory into three quintic polynomials, i.e, for each trajectory, the list of points are a list of x, y and z coordinates. We have computed a 5th degree polynomial that fits all the variations of the all the x coordinates from the start of the spline trajectory to the end of sppline trajectory. Same is repeated for the y coordinate values and the z. Resulting in three quintic polynomials per trajectory.

4.1.2 Function and input

Using the below polyfit function of the NumPy, a python library, we have tried to generate the 5th-degree polynomial equation from log points of the KUKA robot of each trajectory.

numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)

This equation gives the best fitting curve to a given set of points**(x,y)** of degree **deg** by minimizing the sum of squares.

This function takes seven inputs with three compulsory parameters and four optional parameters and returns a vector of coefficients that minimises the square error in the order deg,deg-1,...0. The three compulsory parameters are x y, representing the values we want to fit on two axes and polynomial degree(deg).
For our experiment, we wanted to generate the Polynomial equation of degree five for 3D points (x,y,z) changing in time. To fit our requirement, we have generated the curve for each axes varying in time, that is (x,t), which represents x-axes points changing in time and similarly for y and z axes points (y,t) and (z,t).

4.1.3 Quintic polynomial - whole trajectory

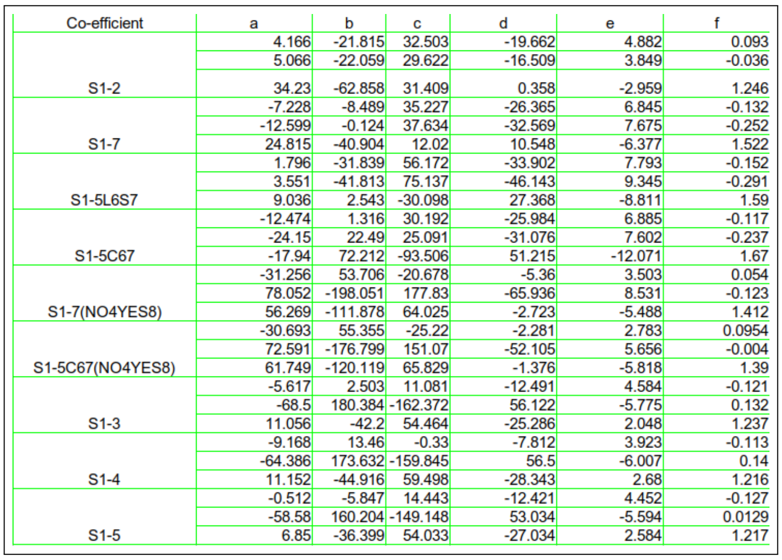
Polynomial interpolation for the whole of spline blocks of all experiments is done together. Hoping to find any noticeable pattern in the coefficients of the quintic polynomials of trajectories, we have obtained three polynomials per each experiment trajectory. The resultant polynomials did not show any kind of obvious pattern or similarity. We were expecting to find at least some kind of similarity between the experiments with very small variation. Observe fig 4.3 for reference.

With the obtained quintic polynomials, we have generated 3-d points so we can compare with the actual robot's trajectory with the trajectory that our interpolated polynomial would produce. The following images visualize the difference.

4.1.4 Quintic polynomial - individual segments

It is evident from the plots that the polynomial that is interpolated for the whole trajectory is not doing enough justice to the actual trajectory of the robot. We had to either increase the complexity of the polynomial by increasing the number of degrees of the polynomial or we can also try to find the polynomial for any specific segment in all the experiments.

For example, we can choose the segment between P2 - P3 because this is a common segment amongst all of the experiments. By doing this we



Co-efficient	a	b	c	d	e	f
	4.166	-21.815	32.503	-19.662	4.882	0.093
	5.066	-22.059	29.622	-16.509	3.849	-0.036
S1-2	34.23	-62.858	31.409	0.358	-2.959	1.246
	-7.228	-8.489	35.227	-26.365	6.845	-0.132
	-12.599	-0.124	37.634	-32.569	7.675	-0.252
S1-7	24.815	-40.904	12.02	10.548	-6.377	1.522
	1.796	-31.839	56.172	-33.902	7.793	-0.152
	3.551	-41.813	75.137	-46.143	9.345	-0.291
S1-5L6S7	9.036	2.543	-30.098	27.368	-8.811	1.59
	-12.474	1.316	30.192	-25.984	6.885	-0.117
	-24.15	22.49	25.091	-31.076	7.602	-0.237
S1-5C67	-17.94	72.212	-93.506	51.215	-12.071	1.67
	-31.256	53.706	-20.678	-5.36	3.503	0.054
	78.052	-198.051	177.83	-65.936	8.531	-0.123
S1-7(NO4YES8)	56.269	-111.878	64.025	-2.723	-5.488	1.412
	-30.693	55.355	-25.22	-2.281	2.783	0.0954
	72.591	-176.799	151.07	-52.105	5.656	-0.004
S1-5C67(NO4YES8)	61.749	-120.119	65.829	-1.376	-5.818	1.39
	-5.617	2.503	11.081	-12.491	4.584	-0.121
	-68.5	180.384	-162.372	56.122	-5.775	0.132
S1-3	11.056	-42.2	54.464	-25.286	2.048	1.237
	-9.168	13.46	-0.33	-7.812	3.923	-0.113
	-64.386	173.632	-159.845	56.5	-6.007	0.14
S1-4	11.152	-44.916	59.498	-28.343	2.68	1.216
	-0.512	-5.847	14.443	-12.421	4.452	-0.127
	-58.58	160.204	-149.148	53.034	-5.594	0.0129
S1-5	6.85	-36.399	54.033	-27.034	2.584	1.217

Figure 4.3: Co-efficient of a Quintic polynomials of all the experiments

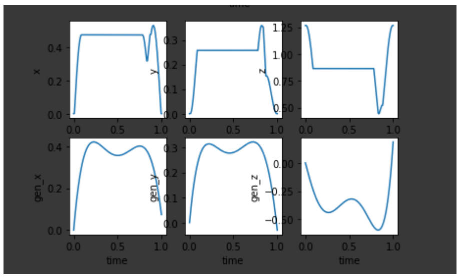


Figure 4.4: Experiment:S1-7

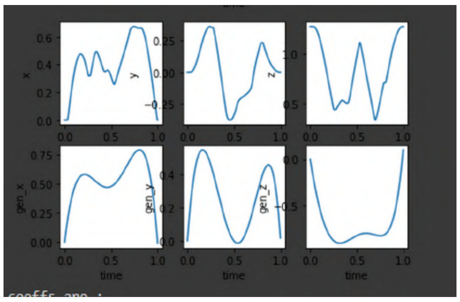


Figure 4.5: Experiment: S1-5L6S7

were hoping to capture the variation in the polynomials we obtains that are caused by the difference in the neighbouring waypoints in the experiment. We have also performed similar study for the segment P1-P2. The following images visualize the variation in polynomials with respect to X, Y and Z axes for all the experiments.

Where the top row of plots in fig 4.6, 4.7 and 4.8 are from robot's original trajectory. Second row of plots are of Interpolated quintic polynomial generated trajectory.

We have noticed that the polynomial coefficients of the segment P2-P3 are very much similar for the experiments 2,3,4 and 9 because the immediate neighbours of the segment P2-P3 in these experiments are common. "P1

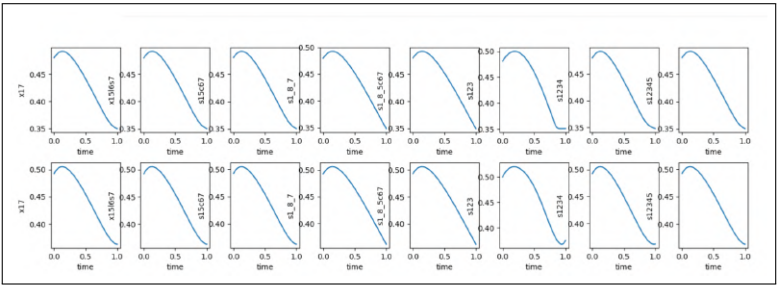


Figure 4.6: X in P2-3 in all experiments

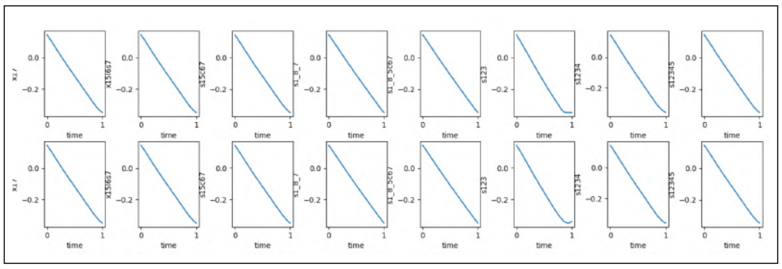


Figure 4.7: Y in P2-3 in all experiments

comes before segment P2-P3 , and P4 comes after segment P2-P3 ", this is the Common pattern in the experiments 2,3,4 and 9. we can refer to experiment list from section 4.1.1, for better understanding.

4.1.5 4 degree and 3 degree

We have also performed 4th degree and cubic polynomial interpolation to see if trajectory generated from such smaller polynomials will match to the robots original trajectory. We noticed that the polynomials smaller than 5th degree is capturing less detail in robot's trajectory than trajectories with 5th degree polynomials.

4.1.6 5 degree with generated points

In the aforementioned version of experiments list in section 4.1.1, the waypoints that were used are randomly sampled from a uniform distribution. A detailed description of how the way points are generated is given in Chapter 3.

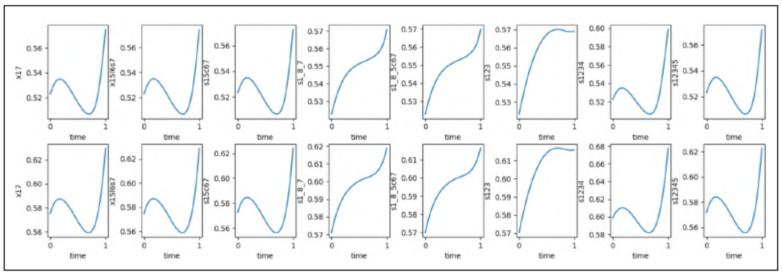


Figure 4.8: Z in P2-3 in all experiments

In the following approach, we have generated more sets of waypoints by sampling each coordinate value from an arbitrarily chosen quintic polynomial. A detailed description of this is given in Chapter 3.

We have updated the experiment list for the new set of waypoints. Following is the list of new set of experiments that were programmed to kuka robots.

- h - p1 - s(p2 p3 p4 p5 p6 p7) - h
- h - p1 - s(p2 p3 p4 p5 p1) - h
- h - p1 - s(p2 p3 p4 p5 p2) - h
- h - p1 - s(p2 p3 p4 p5 p3) - h
- h - p1 - s(p2 p3 p4 p5 p4) - h
- h - p1 - s(p2 p3 p4 p5 p6) - h
- h - p1 - s(p2 p3 p4 p5 p7) - h
- h - p1 - s(p2 p3 p4 p5 p8) - h
- h - p1 - s(p2 p3 p4 p5 p9) - h
- h - p1 - s(1 2 3 1 5 6 7) - h
- h - p1 - s(1 2 3 2 5 6 7) - h
- h - p1 - s(1 2 3 4 5 6 7) - h
- h - p1 - s(1 2 3 5 5 6 7) - h
- h - p1 - s(1 2 3 6 5 6 7) - h

- h - p1 - s(1 2 3 7 5 6 7) - h
- h - p1 - s(1 2 3 8 5 6 7) - h
- h - p1 - s(1 2 3 8 5 6 7) - h
- h - p1 - s(1 3 4) - h
- h - p1 - s(1 2 3) - h
- h - p1 - s(1 2) - h
- h - p1 - s(2 3) - h
- h - p1 - s(3 4) - h

Using the points that were sampled from predefined quintic polynomials, we repeated the pipeline of analysis as done for the waypoints that were randomly generated. We were hoping to obtain quintic polynomial through interpolation of the robot's data, that is similar to the much familiar quintic polynomial that we have arbitrarily chosen to generate waypoints.

Result

We could not establish any concrete correlation between polynomials used to generate the way points and the interpolated polynomial of robot's trajectory. - We have also noticed that the points sparcely sampled from predefined polynomial and shuffling them to randomize the distribution would make our sampling a random sampling. Meaning we are back to previous state.

4.2 Quintic polynomial - path planning

In our next approach we have implemented a method that is designed for path planning of an autonomous guided vehicle TAKAHASHI et al. (1989). It was meant for creating a smooth path for a mobile robot to transition from one point to another point in 2-d space. The function that is developed by implementing this technique for generating a quintic spline that can go through points will need the following input information between every two points.

- start point in 3d space
- end point in 3d space
- x,y,z components of velocity and acceleration at the start
- x,y,z components of velocity and acceleration at the end
- Maximum acceleration and Maximum jerk
- time/iteration step size.

The idea behind this implementation is that, there exists a quintic polynomial whose coefficients are found with the available information such as start position, velocity, acceleration and end position, velocity, acceleration. Said information is used as boundary conditions to find the coefficients of a quintic polynomial. A trajectory is plotted with such a polynomial through small increments of time steps. By doing this, the resultant trajectory is guaranteed to be a quintic polynomial and satisfies the boundary conditions.

4.2.1 2D space to 3D space

We have converted the code that is implemented for trajectory planning of autonomous vehicles in 2D space into a trajectory planning algorithm for 3D space. In order to minimise the total set of input parameters needed we have modified the previous implementation by deducing start angle and end angle from the individual components of the velocity and acceleration itself. Maximum velocity and acceleration are constants that can be obtained from the robot program that we have previously implemented to obtain experimental data.

4.2.2 Start angle and end angle dependencies

In the original implementation of the 2D path planner, angle of starting motion from the start point and angle of arriving motion to the end point were considered. Their purpose was for deducing x and y components of velocity and acceleration. This information is prerequisite to obtain a quintic polynomial trajectory. Which makes us invariably dependent on the data log obtained from the KUKA robot. To avoid such dependency, the values of start and end angle an arbitrary values are set initially and later varied for the best possible values. To be independent of this method instead of angles the code is modified to input velocity and acceleration. But the velocity and acceleration are calculated using data log. The neighboring log points for each segment are extracted from data log and using these points initial velocity,final velocity, initial acceleration and final acceleration are calculated. This algorithm still depends on the datalog and cannot calculate the total spline motion using just input points. So a new approach was developed that is discussed in the upcoming chapter to calculate spline motion with just input points.

4.2.3 Proof that kuka follows 5 degree

Though above method cannot generate a independent trajectory, Since the 5 degree polynomial used has mimicked the datalog behaviour it can be confirmed that KUKA uses 5 degree polynomial to generate it's spline motion.

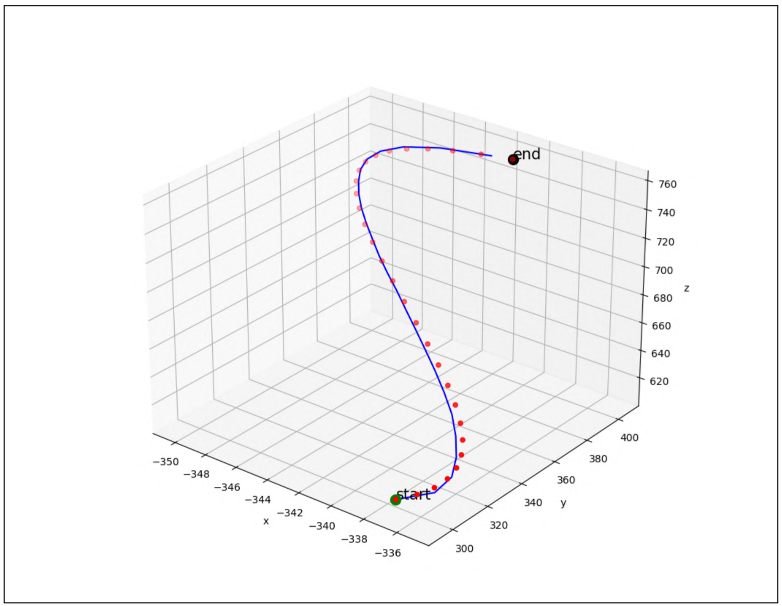


Figure 4.9: The trajectory generated using the quintic path planning algorithm (red) vs the trajectory of the robot(blue).

The above mentioned algorithm can easily be extended to the whole spline block with multiple way points, by using the principles of piece wise spline generation. In this scenario, one has to compute the individual components of velocity and acceleration once per each way point, since the end velocity and acceleration of one point in a segment is the start velocity and acceleration of the same point in the next segment, all while maintaining c2 continuity.

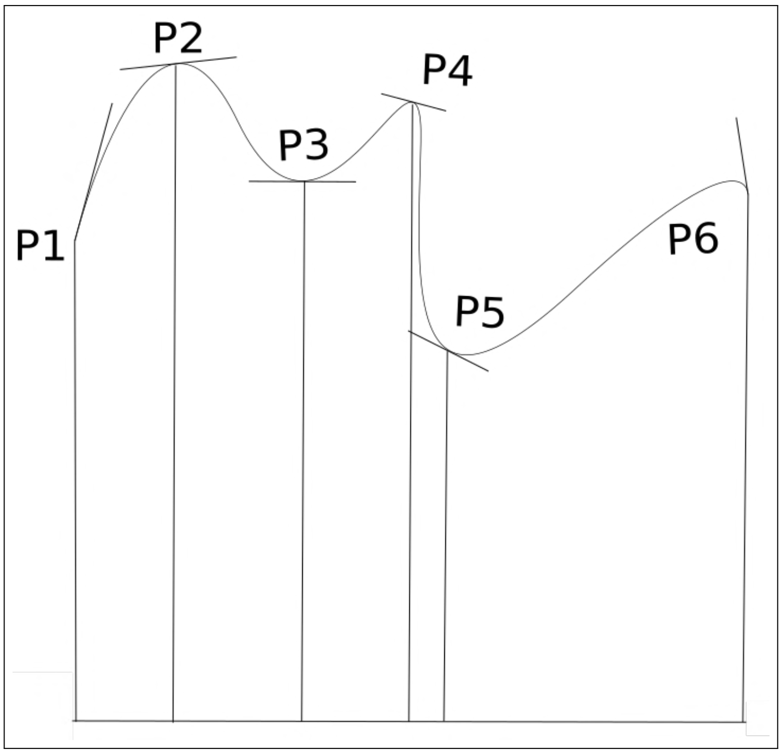


Figure 4.10: Depiction of c2 continuity guaranteed, given the information of first and second derivative at each point are known apriori.

5 Quintic B´ezier

5.1 Algorithm development

This chapter is about the generation and optimization of trajectory from just the input way points. In the previous chapter we dealt with generation of trajectory that has dependency on either the start and stop angles. This was changed in the later part of the previous chapter but the algorithm however expects the velocity and acceleration at each way point. So a new pipeline was developed and adapted based on the SPRUNK (2008).

The whole trajectory is generated only based on the input way points provided.First half of the chapter is about initial trajectory generation using heuristics and the latter talks about the optimization of it.

5.2 Trajectory Generation

B´ezier curve is defined by it's control points. The number of control points define the order of curve. For example when there are three control points, it is cubic B´ezier curve. From the previous chapters it is evident that KUKA iiwa 7 R800 follows 5 degree polynomial for its spline generation. A B´ezier curve with 5 control points is called **Quintic B´ezier** and it follows the following equation.

$$S(t) = (1-t)^5 P_0 + 5(1-t)^4 tP_1 + 10(1-t)^2 t^3 P_2 + 10(1-t)^3 t^2 P_3 + 5(1-t) t^4 P_4 + t^5 P_5 \tag{5.1}$$

The element t refers to the value of time and varies between 0 and 1 from the start and end of the spline segment respectively. The step size at which the t changes would determine the granularity of generated curve. For example when t step size is 10 the t values vary from [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]. at each t value the position value is computed using the equation 5.13, the smaller the step size it becomes granular. Coefficients P_0 - P_5 define the control points of spline segment.

A Quintic B´ezier spline segment S_a between the way points W_1 and W_2 would consists of P0-P5 control points. The neighboring segment S_b between the way points W2 and W3 would have 6 control points P0-P5 at different location. Control Points P_0-P_5 or a spline segment S_i between the way points W_i and W_{i+1} can be computed using Equations

$$P_0 = W_i \tag{5.2}$$

$$P_1 = (\frac{1}{5} * T_i) + P_0 \tag{5.3}$$

$$P_2 = (\frac{1}{20} * A_i) + 2 * P_1 - P_0 \tag{5.4}$$

$$P_3 = (\frac{1}{20} * A_{i+1}) + 2 * P_4 - P_5 \tag{5.5}$$

$$P_4 = P_5 - (\frac{1}{5} * T_{i+1}) \tag{5.6}$$

$$P_5 = W_{i+1} \tag{5.7}$$

The A_i and T_i represent the acceleration and tangent at position i. The are calculated using the heuristics discussed in the 5.3 section.

5.3 Initial Trajectory

5.3.1 First Derivative Heuristic

Input way points are used as the start and end points of each segment in a Quintic B´ezier Spline. Splines can be continuous at various levels. To achieve the C2 level continuity we need to define the curvature i.e, the second derivative. Before that first derivative is defined. Finding first derivative also referred as tangent from now on can be broken down into two

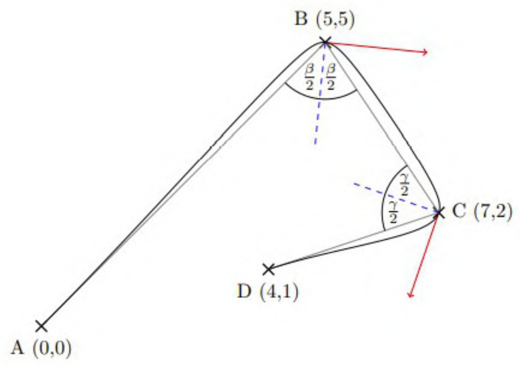


Figure 5.1: The picture depicts tangent magnitude is proportional to its neighbor segments and the direction is perpendicular to the angular bisector [SPRUNK (2008)]

parts. First part referring to the tangent for the way points that are in between the first and last way points i.e, the mid points. The second part refers to the tangent of the first and last way point.

5.3.2 Tangent of mid points

The tangent magnitude of a way point P_i is half to the minimum between the length of segments made by P_i with its neighbouring points i.e, P_{i-1} and P_{i+1}. The Control Variable1 is initialised to 0.5 as mentioned in SPRUNK (2008) and will be further optimised using algorithm mentioned in chapter 6.

$$Tangent_{magnitude} = ControlVariable1 * min(|p_{i-1}, p_i|, |p_i, p_{i+1}|) \tag{5.8}$$

If α is the angle made by the points P_{i-1},P_i,P_{i+1} then tangent is in the perpendicular direction to the angular bisector of the angle α, it is towards the direction of the trajectory. Now that the tangent magnitude and direction are defined tangent can be constructed for the mid points.

5.3.3 Tangent of end points

Since for the start or end points there is a single neighbor way point, tangent magnitude is half of the length of neighbor segment to the point P_i.For the start point tangent direction is towards the robot heading. For the last point tangent direction is matched with the straight line connection to the previous way pointSPRUNK (2008).

$$Tangent_{magendpoint} = ControlVariable1 * |p_{i-1}, p_i| \tag{5.9}$$

$$Tangent_{magstartpoint} = ControlVariable1 * |p_i, p_{i+1}| \tag{5.10}$$

5.3.4 Second Derivative Heuristic

The whole point behind doing second derivative is to ensure that the curvature of the robot is smooth. Following the c2 continuity would reduce the wear out of robot parts. Cubic spline would have the minimization of its second derivative corresponding to small changes in curvature SPRUNK (2008). But it would still have non-zero second derivative (from now on referred as acceleration) at the first and last way points. There would be curvature discontinuity from cubic B´ezier spline segments formed using the points P_{i-1},P_i,P_{i+1} but it also has minimal curvature.

So the second derivative of quintic B´ezier at joint point P_i would be calculated as the weighted average values of two cubic splines (one cubic spline formed by [P_{i-1},P_i] and the second formed by [P_i,P_{i+1}]).

5.3.5 Acceleration of mid points

For the sake of explanation the notion that is explained in Fig 6.1 is used. So to calculate the acceleration at point B the neighbor segments are S_{AB} and S_{BC}. Each segment is considered as cubic spline. By using the tangent values calculated previously, following the notation t_i (Tangent at point i) and points i(Point at i i.e, P_i) the segment AB has points A, B and tangents t_A, t_B. Similarly the segment BC has Tangents t_B, t_C and poinst B, C.

The control points of segment S_{AB} considering it as cubic spline are :

$$S_{\mathrm{AB}} : A, A + \frac{1}{3} t_{\mathrm{A}}, B - \frac{1}{3} t_{\mathrm{B}}, B \tag{5.11}$$

$$S_{\mathrm{BC}} : B, B + \frac{1}{3} t_{\mathrm{B}}, C - \frac{1}{3} t_{\mathrm{C}}, C$$

(5.12)

The second derivative of the spline S$_{\mathrm{AB}}$ and S$_{\mathrm{BC}}$ at point B can be calculated as:

$$S''_{\mathrm{AB}}(1) : 6\left(\left(A + \frac{1}{3} t_A\right) - 2\left(B - \frac{1}{3} t_B\right) + B\right) = 6A + 2t_A + 4t_B - 6B \tag{5.13}$$

$$S''_{\mathrm{BC}}(0) : 6\left(B - 2\left(B + \frac{1}{3} t_B\right) + \left(C - \frac{1}{3} t_C\right)\right) = -6B - 4t_B - 2t_C + 6C \tag{5.14}$$

Since there is would be slight effect based on length of these segments the second derivative would be sum of their weighted average based on inverse of their length. Let's say the accelertion value a then :

$$a = ControlVariable2 * \alpha * S''_{\mathrm{AB}}(1) + ControlVariable3 * \beta * S''_{\mathrm{BC}}(0) \tag{5.15}$$

The Control Variable2 and Control Variable3 are initialised to 1 and will be further optimised in the following chapter. α is inversely proportional to its length and Beta is inversely proportional to its length. Considering the notion d$_{\mathrm{AB}}$ as the magnitude of segment AB and d$_{\mathrm{BC}}$ as magnitude of segment BC, $\alpha$and β are :

$$\alpha = \frac{d_{\mathrm{BC}}}{d_{\mathrm{AB}} + d_{\mathrm{BC}}} \tag{5.16}$$

$$\beta = \frac{d_{\mathrm{AB}}}{d_{\mathrm{AB}} + d_{\mathrm{BC}}} \tag{5.17}$$

5.3.6 Acceleration of End points

Since there is only one segment available the same procedure as above is followed by excluding the weighted average part.

$$a_{start} = ControlVariable4 * S^{''}{}_{\mathrm{AB}}(1) \tag{5.18}$$

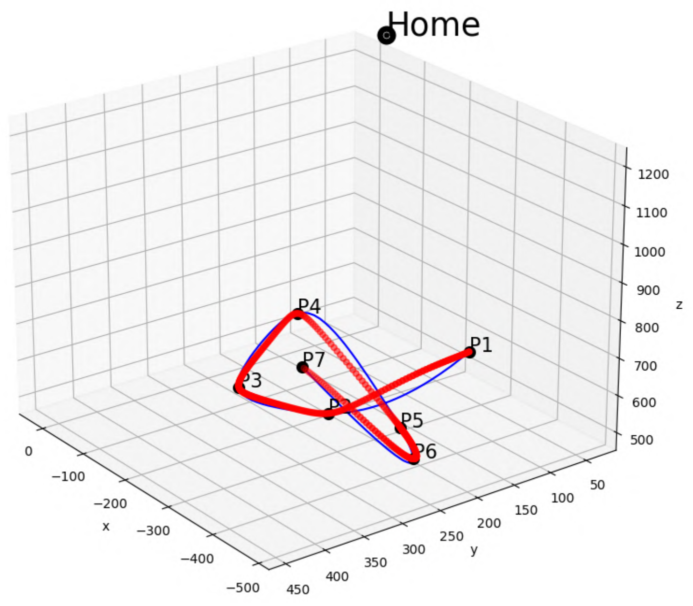
$$a_{end} = ControlVariable4 * S^{''}{}_{\mathrm{BC}}(0) \tag{5.19}$$

The control Variable4 is initialised to 1 and will be further optimised in the next chapter. The next chapter deals with how the optimization is carried out for the control variables mentioned in the equations 5.8, 5.9, 5.10, 5.15, 5.18, 5.19 so that the generated algorithm can further converge to the spline behaviour of KUKA.

6 Optimization

6.1 Optimization Variables

Whole trajectory can be influenced by certain variables. These variables are called optimization variables. The right combination of optimization variables can smooth the trajectory and reduce sharp curves SPRUNK (2008). The values used for the control variables(also referred as optimization variables) are initialised to 0.5,1,1,1 based on the values mentioned in SPRUNK (2008). However these result in trajectory that is different from the curve produced by KUKA. Hence these values can be further optimised to match the KUKA generated trajectory. The following figure with red points as algorithm generated curve and blue as the KUKA generated curve represents optimised curves.



6.1.1 Tangent Magnitude

While calculating the tangent magnitude of a way point using the equation 5.8, the length of tangent decides how far the robot has to travel in the direction given by tangent direction. This parameter decides multiplication factor to get desired tangent from initial calculated tangent magnitude. Tangent magnitude influences the curve sharpness i.e, the elongation of tangent would create less sharp curve for the way point.

Hence this parameter is referred as elongation factorSPRUNK (2008). There are 2 possible elongation factors, one for the tangent mid points from equation 5.8 and other from equation 5.9 and 5.10 corresponding to elongation factor for tangents of end points.

6.1.2 Acceleration Magnitude

Acceleration magnitude is the combination of $S''_{AB}(1)$ and $S''_{BC}(0)$ by a factor of α and β. These two parameters can also be optimised. The influence of alpha and beta would be to deform the adjacent segment. The optimization of above mentioned four variables are done using Differential Evolution algorithm explained in the section 6.2

6.2 Differential Evolution and similarity measure

With various optimization algorithms readily available, differential evolution could optimize multiple variables and has large search space with no prior assumptions.

6.2.1 Differential Evolution

The differential Evolution function is available in the scipy VIRTANEN et al. (2020) library developed based on the STORN and PRICE (1997) algorithm. It is a stochastic optimization algorithm which performs minimization of function called from it. This algorithm is used to find the global minimum of a multivariate function. Since there are four variables to be optimised this algorithm serves the purpose very well.

$$scipy.optimize.differential\_evolution \tag{6.1}$$

Differential evolution has many input parameters, some of the very important parameters among them are **func,bounds,strategy,maxiter,popsize** and output is the optimize result i.e, a boolean array indicating success of optimization and also the result.fun returns the optimized value. Input parameters are further explained in below section(6.2.1.1)

Func

The objective function that has to be minimised using the differential algorithm is called in Func. It takes the format of f(x, *args) where x represents 1-D array argument. Additional parameters can be mentioned using *args

Bounds

Bounds are the boundaries for each variable that has to be optimized using differential evolution algorithm. It can be specified using either a set of bound for whole class or for each instance of x element in X-array.

Strategy

This algorithm supports many strategy to create trial candidates at each pass,they are **best1bin, best1exp, rand1exp, randtobest1exp, currenttobest1exp, best2exp, rand2exp, randtobest1bin, currenttobest1bin, best2bin, rand2bin, rand1bin** since it is out of scope to explain each here they are mentioned and if interested further reading is recommended by the reader. Currently the algorithm uses best1bin which chooses two members from the population as initial candidates and their difference is used for further mutation.

maxiter

This parameter describes number of generations the whole population is evolved. The larger the value , it takes more time for computation.

Popsize

It represents the population size multiplier and when input parameter is seit to 'sobol' then population is calculated with power of 2

$$popsize * len(x) \tag{6.2}$$

6.2.2 Similarity measure

Differential Evolution algorithm expects a callable function to optimize. Since there is existing experimental data of Kuka robot and also the trajectory generated initially using the heuristics mentioned in chapter 5, a function that compares both of them and gives a score to understand how well the generated trajectory matches the experimental data is required. Similarity measure libraryJEKEL et al. (2019) has been developed to quantify the difference between two curves. It has goo =d number of functions to calculate the distance between them or area covered between them and many more. The dynamic time wrapping function(dtw) would compute the non metric distance between two curves that are time-series generated. In the current scenario it takes experimental data and numerical data representing the data obtained using experiment and trajectory generated respectively.

$$r, d = dtw(expdata, numdata, metric =' euclidean') \tag{6.3}$$

It measures the euclidean distance between them and outputs a distance value in float. The smaller output value corresponds to similar curves and larger output value represents the curves are dissimilar and have a large distance.

6.3 Final Optimised values

Out of twenty two experiments performed some of them have as little as 2 way points and the major experiments have 5 or 6 way points. Each experiment produced 4 optimal variable values that are suitable for it. However a global optimal values for these four variables had to be determined.

There are various methods to perform this. One of them is just picking the 4 variable values of an experiment that had least dtw(dynamic time wrapping distance). The other method is to feed all the experiments to the differential evolution algorithm to produce single set of optimised values.

The second method takes too much time to compute, however the first method gives optimal values specific to the experiment. Then their weighted dtw distance is calculated and the final optimal values are taken for the experiment that has lowest weighted dtw distance. The experiments with less number of segments exhibit less dtw value due to their nature of exactly fitting 5 degree polynomial, for example for a spline of two points it is a straight line equation fit into 5 degree polynomial so it clearly fits and the dtw distance is low. To avoid such bias the spline experiments with atleast 6 spline points are considered for final optimization.

A
Detailed Result Analysis

The two images shown here are obtained by performing differential evolution optimization to find the best set of control variables that can guarantee maximum match between the KUKA robot's spline trajectory and the trajectory generated by our Quintic B´ezier spline algorithm. In the first image , the trajectory that we have generated (shown in red) is generated from the control variable that are optimised to minimise average distance metric across all the experiments/spline blocks.

In the second image we see the trajectory that is generated by control variables that are meant to minimize the distance metric specifically for this particular set of way points configuration, in other words, this specific spline.

From the aforementioned analysis, we have observed that, there is not much of a significant difference between the trajectory that is generated by control variables that are optimised to minimise overall average score and the trajectory that is generated by control variables that are optimised to minimise the distance for each specific experiment. With the Optimized DTW(Dynamic time warping) distance Score:1846.840 and 1830.287 respectively. We are limited by the amount of data we have collected and types of spline configurations we have explored.

In the Table A.1, from the 3rd column labeled "Control Variable 1" to the 6th column "Control Variable 4" are the variables that have been obtained from the optimization step. The optimisation algorithm we have used requires three pieces of information as an input, one is the list of variables, their respective bounds and a metric that is minimised.

In the Table A.1, 7th column, 8th column and 9th column represent DTW distance metrics for three different types of trajectories we have generated

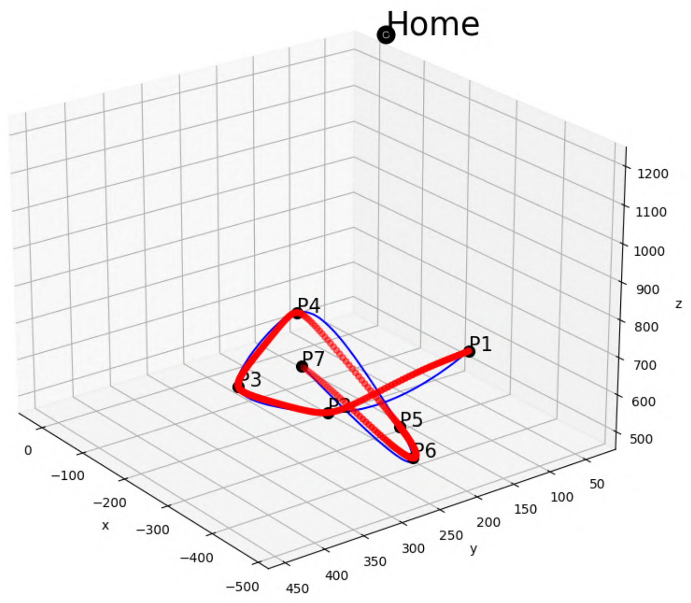


Figure A.1: Comparison between Robot's trajectory (Blue) and the Trajectory generated unoptimised control variables (Red)

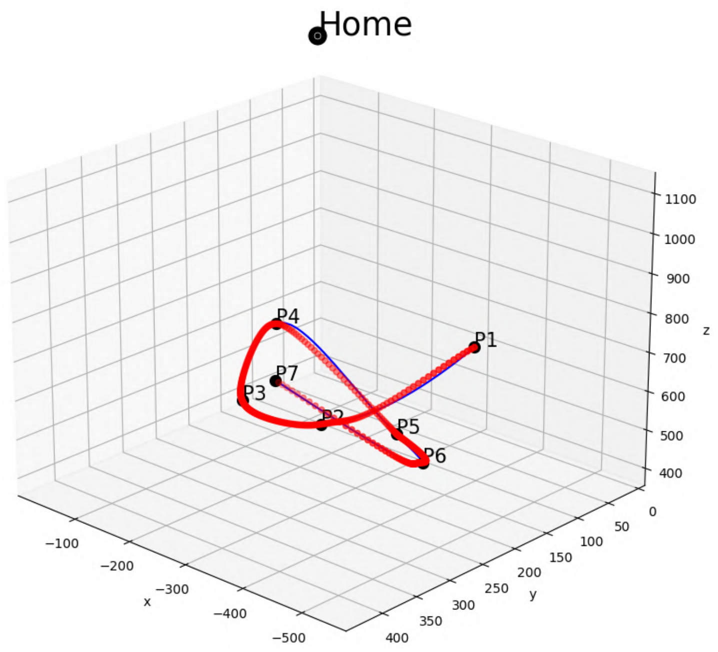


Figure A.2: Comparison between Robot's trajectory (Blue) and the Trajectory generated from overall experiments based optimised control variables (Red)

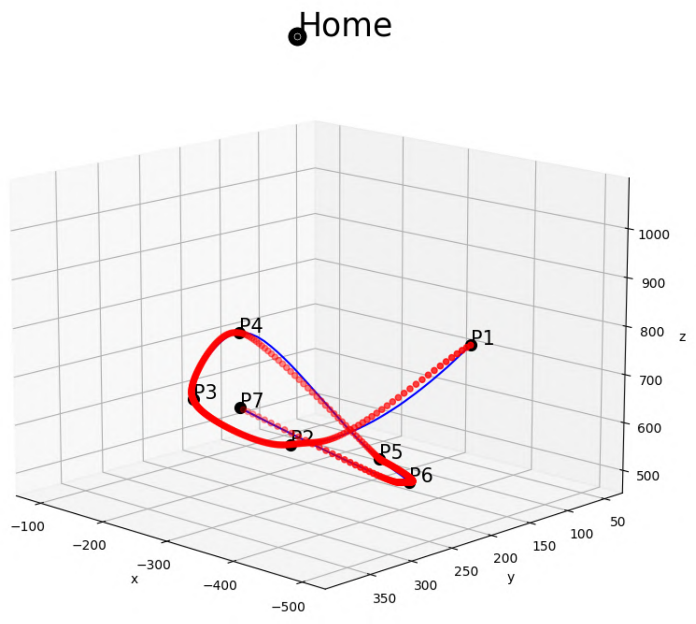


Figure A.3: Comparison between Robot's trajectory (Blue) and the Trajectory generated from individual experiment based optimised control variables (Red)

S.no	Experiment	Control Variable 1	Control Variable 2	Control Variable 3	Control Variable 4	Individually Optimised Score	Overall optimised score	Unoptimised
1	home(H)1234567H	0.51625016	0.21824477	0.09832954	-4.70750856	1822.457666	1836.716063	3359.556999
2	H123451H	0.49150283	0.29449145	0.0030767	-3.01724061	1438.147992	1528.982396	3000.818554
3	H123452H	0.93627083	0.84603186	0.97434401	-2.76241202	1475.736079	1649.883389	2605.36705
4	H123453H	0.43832923	0.06750762	0.0241364	-0.99963715	1532.597786	1781.190768	2502.233389
6	H123456H	0.65109629	0.38521984	0.3804515	-4.61705169	1481.260395	1558.262149	2901.605697
7	H123457H	0.40909474	0.01349362	0	-1.3055848	1585.943594	1716.378627	2466.975886
8	H123458H	0.69497262	0.4831442	0.40740118	-4.15549924	1554.536256	1630.656229	3048.87614
9	H123459H	0.6555418	0.5402687	0.28669777	-4.48028927	1509.667314	1605.065772	3032.369057
10	H1231567H	0.51458768	0.14121976	0.16821533	-4.74459839	1919.000404	1943.838491	3513.337202
12	H1234567H	0.51486911	0.19911942	0.11412941	-4.71831165	1830.287485	1846.840547	3365.414375
15	H1237567H	0.51137583	0.12729807	0.04581003	-4.33867638	1450.368702	1463.700662	2737.791898
16	H1238567H	0.42161375	0	0.17204282	-5.1465091	1755.028781	1817.795799	2994.561541
17	H1238567H	0.4292206	0	0.11627169	-4.94535119	1764.556978	1826.240003	2994.917367
18	H134H	1	0	1	-1.31121945	587.9776739	661.6721785	1538.02076
19	H123H	1	0	1	-0.9397716	643.4726597	853.5466808	1519.321947
20	H12H	0.31531583	0.82849286	0.4007474	1.40770912	116.827006	116.827006	116.827006
21	H23H	0.13927194	0.19206982	0.25749711	-2.44304374	124.8729228	124.8729228	124.8729228
22	H34H	0.56966978	0.74041517	0.72877876	8.43227581	109.3931614	109.3931614	109.3931614
	Globally optimised variable set	**0.4898907210**	**0.2540456360**	**0.0038133780**	**-4.5758553200**			

Table A.1: This table consists of the four control variables obtained from optimization step and respective optimal DTW distances.

Individually optimised Trajectory scores	**Overall experiments optimized Trajectory scores**	Difference	**% difference**
1822.457666	1836.716063	14.25839742	0.782371941
1438.147992	1528.982396	90.8344037	6.316067901
1475.736079	1649.883389	174.1473102	11.80070832
1532.597786	1781.190768	248.5929817	16.22036675
1481.260395	1558.262149	77.00175353	5.198394136
1585.943594	1716.378627	130.4350328	8.22444337
1554.536256	1630.656229	76.11997283	4.896635413
1509.667314	1605.065772	95.39845792	6.319170922
1919.000404	1943.838491	24.83808684	1.294324211
1830.287485	1846.840547	16.55306286	0.904396878
1450.368702	1463.700662	13.33196025	0.9192118
1755.028781	1817.795799	62.76701786	3.576409602
1764.556978	1826.240003	61.68302573	3.49566642
587.9776739	661.6721785	73.69450467	12.53355492
643.4726597	853.5466808	210.074021	32.64692258
116.827006	116.827006	0	0
124.8729228	124.8729228	0	0
109.3931614	109.3931614	0	0

Table A.2: This table consists of the DTW distance metrics for two types of trajectories we have generated with respect to the actual KUKA robot's spline trajectory. First type: Individual experiment wise optimised trajectory. Second type: Overall experiments based optimised trajectory

unoptimized trajectory scores	**overall experiments** optimized trajectory scores	Difference	% difference
3359.556999	1836.716063	1522.840936	82.91106973
3000.818554	1528.982396	1471.836158	96.26246592
2605.36705	1649.883389	955.4836609	57.9121935
2502.233389	1781.190768	721.0426213	40.48093188
2901.605697	1558.262149	1343.343548	86.207802
2466.975886	1716.378627	750.5972585	43.73144985
3048.87614	1630.656229	1418.219911	86.97234195
3032.369057	1605.065772	1427.303285	88.92490953
3513.337202	1943.838491	1569.498711	80.74223851
3365.414375	1846.840547	1518.573828	82.2254975
2737.791898	1463.700662	1274.091236	87.04588777
2994.561541	1817.795799	1176.765742	64.73585989
2994.917367	1826.240003	1168.677364	63.99363511
1538.02076	661.6721785	876.3485819	132.4445262
1519.321947	853.5466808	665.7752666	78.0010375
116.827006	116.827006	0	0
124.8729228	124.8729228	0	0
109.3931614	109.3931614	0	0

Table A.3: This table consists of the DTW distance metrics for two types of trajectories we have generated with respect to the actual KUKA robot's spline trajectory.First type: Individual experiment wise optimised trajectory. Second type: Un-optimized trajectory

In the Table A.1, 7th column represents the minimised scores for each experiment. The score represent the distance between trajectory that we have generated using our approach and the the trajectory that the KUKA robot has computed. The DTW distance is unit-less, they are meant for relative quantitative comparison of two trajectories only.

In the Table A.1, 8th column represents the minimised scores for the trajectory that is generated when we tried to optimise the average of all DTW distances across all experiments. The corresponding control variables for such an optimisation is given in the row 23 of Table A.1.

In the Table A.1, 9th column represents the minimised scores, for the first type is the trajectories that we have generated without involving any kind of optimisation, i.e, with the control variable values being heuristically computed based on the input way points alone. The score for the unoptimized of trajectory is seen in the 9th column.

In the Table A.2 and A.3, We have showcased the DTW distances between the the trajectories we have generated and the actual KUKA robot's trajectories. Since we have generated trajectories with three different strategies or types, we are comparing between the three types of trajectories by comparing their respective DTW scores in each experiment.

In the Table A.2, The first column represents the trajectory that is generated by optimising the control variables for each experiments independently. Each experiment has its own optimised set of variables, as showcased in Table A.1. The second column represents trajectory that is generated by only one set of 4 control variables, row 23 in Table A.1. This unique set of control variables are a obtained by averaging the DTW distances of all experiments and considering that as a minimisation metric for the optimiser. This resulted in a Unique set of 4 control variables that can generate trajectories that are almost as good as the independently optimised control variables. 4th column in Table A.2 shows the percentage difference between the two types of trajectories generated.

Finally, In the Table A.3, The first column represents trajectories that are built without involving any kind of optimisation. Before the optimization step, during the implementation of our final approach of trajectory generation, some of the variables that control the spline path are heuristically computed based on the input way point information alone. We have used these heuristics to generate the trajectories. We have compared it with the KUKA robot's actual trajectory, using a quantitative DTW distance as metric. The first column represents these metrics. The second column represents the distance metrics of trajectories generated by Unique set of control variables applied to all the experiments. 4th column represents the percentage difference between the two distances. It is clear that the difference between the un-optimised version of trajectory and the optimised version of the trajectory is very apparent. Refer to figures A.1, A.2, A.3 for comparison between the types of trajectories vs the Robot's actual trajectory.

A.1 Conclusion

KUKA robot's spline function can be replicated with certain degree of error margin. We still cannot model the exact function that the KUKA robotic system uses to compute a spline. But, we can generate a spline function that runs very close to KUKA robot's spline function, given the same set of inputs. A greater accomplishment of the project is the reduction in the amount of search space for the 'real' KUKA spline function.

A.2 Future scope

Using the KUKA spline trajectory generation function described in the chapter 6, a new function that can generate way points for desired CAD spline/curve input, could be developed. Methods of trajectory generation described in chapter 5 could be further modified by adding more advanced optimization variables to detect the spline function of KUKA accurately. The quintic path generation method described in chapter 4 could be modified to remove the dependency of In the data log. 2, The first column represents the trajectory that is generated by optimising the control variables for each experiments independently. Each experiment has its own optimised set of variables, as showcased in Table A.1. The second column represents trajectory that is generated by only one set of 4 control variables, row 23 in Table A.1. This unique set of control variables are a obtained by averaging the DTW distances of all experiments and considering that as a minimisation metric for the optimiser. This resulted in a Unique set of 4 control variables that can generate trajectories that are almost as good as the independently optimised control variables.

B
Abbreviations and Notations

Acronyms

Acronym	Meaning
DTW	Dynamic time warping
dtw	dynamic time wrapping distance
popsize	population size
maxiter	maximum iterations
$\mathbf{S}''_{AB}(\mathbf{1})$	End of Spline segment between A and B points
$\mathbf{S}''_{AB}(\mathbf{0})$	Start of Spline segment between A and B points
$\mathbf{P}_i$	Point i

C
List of Figures

D
Bibliography

[DENAVIT and HARTENBERG 1955] J. Denavit and R. S. Hartenberg. **A kinematic notation for lower-pair mechanisms based on matrices**. 1955.

[FARIA et al. 2018] C. Faria, F. Ferreira, W. Erlhagen, S. Monteiro and E. Bicho. **Position-based kinematics for 7-DoF serial manipulators with global configuration control, joint limit and singularity avoidance**. Mechanism and Machine Theory, Vol. 121:317–334, 2018.

[KUKA] **LBR iiwa 7 R800, LBR iiwa 14 R820**.

[SPRUNK 2008] C. Sprunk. **Planning motion trajectories for mobile robots using splines**. 2008.

[TAKAHASHI et al. 1989] A. Takahashi, T. Hongo, Y. Ninomiya and G. Sugimoto. **Local path planning and motion control for agv in positioning**. In: Proceedings. IEEE/RSJ International Workshop on Intelligent Robots and Systems'.(IROS'89)'The Autonomous Mobile Robots and Its Applications, pp. 392–397. 1989, IEEE.

[VIRTANEN et al. 2020] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors. **SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python**. Nature Methods, Vol. 17:261–272, 2020.

[Storn and Price 1997] R. Storn and K. Price. **Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces**. Journal of global optimization, Vol. 11(4):341–359, 1997.

[Jekel et al. 2019] C. F. Jekel, G. Venter, M. P. Venter, N. Stander and R. T. Haftka. **Similarity measures for identifying material parameters from hysteresis loops using inverse analysis**. International Journal of Material Forming, 2019.

Declaration of Academic Integrity

We hereby declare that I have written the present work ourself and did not use any sources or tools other than the ones indicated.

Datum:01.04.2022 ....

(Signature)