Robotic Arm Geometry and Simple Trajectory Planning

Farros Alferro [1] **Jose Victorio Salazar** [2]

1. Introduction

One of the most important fields of robotics is robotic arms and manipulation. Many experts suggest that robotics began with the development of the first industrial manipulator, Unimate, in the early 1950s. Throughout history, several robotic arms have been developed and employed for a variety of jobs ranging from welding and vehicle assembly to food handling.

In this experiment, we will learn about the link between the angles of a robot joint and the position of its endpoint (kinematics) as well as simple path planning for making a smooth motion using a robot manipulator. In addition, we will use the knowledge gained to generate trajectories for a robotic arm and duplicate them on a robotic simulator using Processing [1].

2. Principles

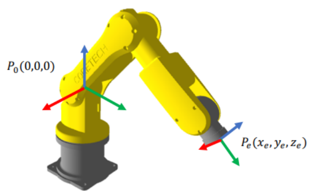
2.1. Robotic Arms

Robotic manipulators, also known as robotic arms, are robots that are comparable to human limbs in that they are made up of a series of connected segments and joints. They may be designed to execute a variety of tasks swiftly, efficiently, and precisely. When regulating the motion of a robotic arm, we must consider numerous factors, including kinematics, dynamics, motion planning, and motion control.

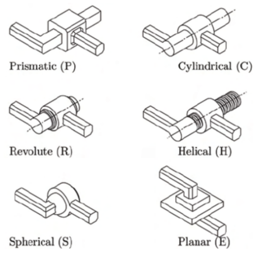
Robotic arms, unlike human arms, come in a variety of configurations based on the type of joints utilized and how they are attached. The segments are the bodies that connect two joints and are sometimes referred to as **links**. The **joints**, which are the parts responsible for movement, can be of several types (revolute, prismatic, helical, spherical, cylindrical and planar, as can be seen in Fig. 1(b)). The number of different directions in which movement can occur (called the Degrees of Freedom) vary depending on the joint. For instance, a revolute joint can move in one DoF while cylidrical joint can move in two DoF.

[1]Computer Vision Lab., Tohoku University, Sendai, Japan [2] Smart Robots Design Lab., Tohoku University, Sendai, Japan. Correspondence to: Farros Alferro <farros.alferro.t3@dc.tohoku.ac.jp>, Jose Victorio Salazar <j.salazar@srd.mech.tohoku.ac.jp>.

Theme 5 Report on *Laboratory Experiment II* which was done on December 2nd, 2022. Copyright 2022 by the author(s).



(a) 3 DoF Arm



(b) Classifications of Joints [2]

Figure 1. Overview of robotic arms [3].

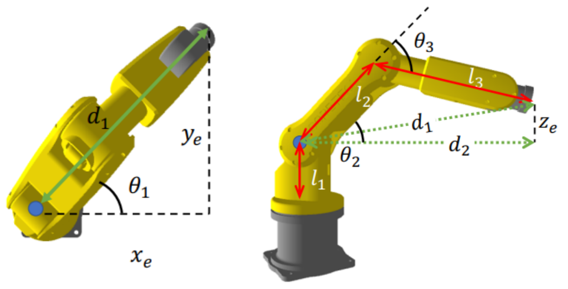
For the purpose of simplicity, we calculate the kinematics of the 3 DoF robot illustrated in Fig. 1(a) using trigonometrical relationships, focusing just on the position P_e rather than the entire position and orientation of the end-effector.

2.2. Kinematics and Arm Geometry

Kinematics is the study of the motion of the robotic arm independent of the force/torque applied on it. The position and orientation of the manipulator can be seen from two perspectives: the position and orientation (the **pose** in cartesian space) of the tip of the arm, known as the end-effector, or the rotation **angle** of each joint (joint space). The link between them is specified by robot arm kinematics: **forward kinematics** allows us to compute the end-effector posture from the angles of each joint, while **inverse kinematics** allows us to determine the angles of each joint given the location of the end-effector.

To clearly identify the link between the angles of the joints and the position of the end-effector, we first specify the coordinate frame of the robot, as shown in Figure 2. In this diagram, $P_e(x, y, z)$ denotes the position of the endpoint while θ_1, θ_2, and θ_3 are the angles between each joints. In addition, l_1, l_2, l_3 define the length of each link. Finally, d_1 and d_2 denote the distance from the origin $P_0(0, 0, 0)$ to the end-effector and the distance projected in the $x - y$ plane, respectively.

From the below figure, the angle θ_1 and θ_3 can be derived as follows:



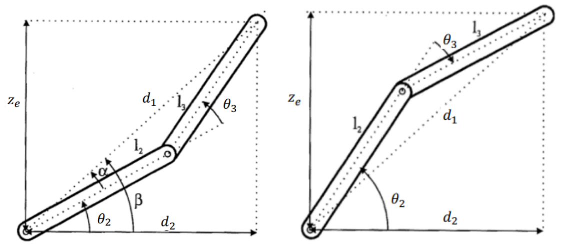
(a) Top view (b) Side view

Figure 2. Robot coordinate frame [3].

$$\theta_1 = atan\left(\frac{y_e}{x_e}\right) \tag{1}$$

$$\theta_3 = acos\left(\frac{d_1^2 - l_2^2 - l_3^2}{2 l_2 l_3}\right) \tag{2}$$

where $d_1 = p_x^2 + p_y^2 + p_z^2$. As shown in Fig. 3(a) and Fig. 3(b), there are two alternative solutions for this geometrical configuration for a given point P_e. These two options are commonly referred to as elbow down and elbow up. The angle is the same, but the sign is different, so changing the sign of θ_3 provides both alternatives.



(a) Elbow down (b) Elbow up

Figure 3. Links 2 and 3 geometrical relation [4].

Next, calculating θ_2 based on Figure 3(a):

$$\beta = atan\left(\frac{z_e}{d_2}\right) \tag{3}$$

$$\alpha = atan\left(\frac{l_3 \sin\theta_3}{l_2 + l_3 \cos\theta_3}\right) \tag{4}$$

$$\theta_2 = \beta - \alpha \tag{5}$$

where $d_2^2 = x_e^2 + y_e^2$. By substituting equations 3 and 4 to equation 5, the value of θ_2 is obtained as follows:

$$\theta_2 = atan\left(\frac{z_e}{\sqrt{x_e^2 + y_e^2}}\right) - atan\left(\frac{l_3 \sin\theta_3}{l_2 + l_3 \cos\theta_3}\right) \tag{6}$$

We can derive the set of joint angles that will allow the manipulator to reach arbitrary positions $P_e(x, y, z)$ in space using these three expressions of θ_1, θ_2, and θ_3. However, it is critical to consider what occurs when we attempt to calculate the joint angles for a location that is outside of the manipulator's reachable space. In equation 2, for example, if the numerator is greater than the denominator, the argument of the acos function will be more than 1 (or less than -1), which is beyond the domain of the acos function and will result in an error.

2.3. Kinematic Control and Trajectory Planning

Kinematic control includes subjects such as trajectory planning, interpolation, Jacobian matrices, and others. The process of creating the geometric path that the endpoint will take is known as trajectory planning. On the other hand, trajectory interpolation is the process of generating the motion law as a function of time that follows the planned geometric path while adhering to specific constraints on the trajectory's continuity (smoothness) and its time derivatives up to a given degree. We can effectively construct the arm motion using these two approaches. Figure 4 summarizes the method of controlling the position of the arm in general.

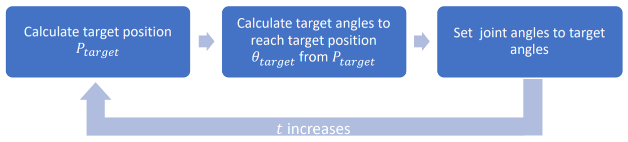

Calculate target position P_{target}
Calculate target angles to reach target position θ_{target} from P_{target}
Set joint angles to target angles
t increases

Figure 4. Kinematic control loop [3].

For a simple trajectory, we employ a linear motion from point $P_a(x_a, y_a, z_a)$ to point $P_b(x_b, y_b, z_b)$. The end-effector position could thus be defined parametrically:

$$P_{target}(s) = \begin{cases} P_a & \text{if } t < t_0 \\ P_a + (P_b - P_a)\left(\frac{s(t)}{s_{max}}\right) & \text{if } t_0 \le t \le t_f \\ P_b & \text{if } t > t_f \end{cases}$$

where the boundary conditions are $s(t_0) = 0$ and $s(t_f) = s_{max}$. Moreover, t_0 and t_f are the times at which the motion begins and terminates, respectively.

Considering the nature of the motion in time, we employ the motion law $s(t)$ rather than directly calculating P_{target} as a function of time. If P_{target} is directly proportional to time, then its second derivative (acceleration) is not a continuous function. To get the system to move/stop instantly at a constant pace at the endpoints, infinite acceleration/deceleration would be required, which is physically impossible. The system's response in the best-case situation would be a sudden acceleration, which is risky and might cause malfunctions or even accidents. Instead, we can define a motion law that is a continuous function up to a particular degree of derivation, ensuring the continuity and smoothness of changes in position, velocity, and acceleration.

In this experiment, we utilize two categories of motion law $s(t)$: polynomial (1st, 3rd, 5th order) and trapezoidal interpolation.

2.3.1. 1ST-ORDER POLYNOMIAL (CONSTANT VELOCITY)

This is the case for linear motion, where the motion law $s(t)$ is defined as:

$$s(t) = a_0 + a_1 t \tag{7}$$

by inputting the boundary conditions $s(t_0) = s_0$ and $s(t_f) = s_f$, the motion law is obtained:

$$\begin{aligned} s(t) &= s_0 + \left(\frac{\Delta t}{T}\right)(s_f - s_0) \\ \dot{s}(t) &= \frac{1}{T}(s_f - s_0) \\ \ddot{s}(t) &= 0 \end{aligned} \tag{8}$$

where $\Delta t = t - t_0$ and $T = t_f - t_0$. 1st-order polynomial law should not be used as its velocity and acceleration profiles are not continuous. The system requires infinite acceleration at the end and beginning of its movement (zero velocity to constant velocity and vice versa in a very short time $\Delta t \to 0$). To resolve this, a higher-order polynomial motion law must be employed.

2.3.2. 3RD-ORDER POLYNOMIAL

The motion law for this case is defined as:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{9}$$

where the boundary conditions are the same as in 1st-order case, but with additional velocity profiles: $\dot{s}(t_0) = \dot{s}(t_f) = 0$. Substituting the boundary conditions, the motion law equation is obtained:

$$\begin{aligned} s(t) &= s_0 + \left[3\left(\frac{\Delta t}{T}\right)^2 - 2\left(\frac{\Delta t}{T}\right)^3\right](s_f - s_0) \\ \dot{s}(t) &= \frac{6}{T}\left[\left(\frac{\Delta t}{T}\right) - \left(\frac{\Delta t}{T}\right)^2\right](s_f - s_0) \\ \ddot{s}(t) &= \frac{6}{T^2}\left[1 - 2\left(\frac{\Delta t}{T}\right)\right](s_f - s_0) \end{aligned} \tag{10}$$

2.3.3. 5TH-ORDER POLYNOMIAL

The motion law for this case is defined as:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{11}$$

where the boundary conditions are the same as in 3rd-order case, but with additional acceleration profiles: $\ddot{s}(t_0) = \ddot{s}(t_f) = 0$. Substituting the boundary conditions, the motion law equation is obtained:

$$\begin{aligned} s(t) &= s_0 + \left[10\left(\frac{\Delta t}{T}\right)^3 - 15\left(\frac{\Delta t}{T}\right)^4 \right. \\ &\quad \left. + 6\left(\frac{\Delta t}{T}\right)^5\right](s_f - s_0) \\ \dot{s}(t) &= \frac{30}{T}\left[\left(\frac{\Delta t}{T}\right)^2 - 2\left(\frac{\Delta t}{T}\right)^3 + \left(\frac{\Delta t}{T}\right)^4\right](s_f - s_0) \\ \ddot{s}(t) &= \frac{60}{T^2}\left[\left(\frac{\Delta t}{T}\right) - 3\left(\frac{\Delta t}{T}\right)^2 + \left(\frac{\Delta t}{T}\right)^3\right](s_f - s_0) \end{aligned} \tag{12}$$

2.3.4. TRAPEZOIDAL INTERPOLATION

This interpolation is separated into three sections. Assuming a positive displacement, i.e. $s_f > s_0$, the acceleration is positive and constant in the first section, and so the velocity is a linear function of time and the position is a parabolic curve. The acceleration is zero in the second section, the velocity is constant, and the location is a linear function of time. In the final section, there is a continuous negative acceleration, the velocity drops linearly, and the position is again a polynomial function of degree two. The duration T_a of the acceleration phase is commonly employed to be equal to the duration T_d of the deceleration phase for certain trajectories [5].

1. **Acceleration phase**, $t \in [0, T_{acc}]$. The motion law is defined as:

$$s(t) = a_0 + a_1 t + a_2 t^2 \tag{13}$$

by inputting initial conditions: $s(0) = s_0$, $\dot{s}(0) = 0$, and $\dot{s}(T_{acc}) = v_c$, the motion law and its derivatives are obtained:

$$\begin{aligned} s(t) &= s_0 + \frac{v_c}{2T_{acc}}t^2 \\ \dot{s}(t) &= \frac{v_c}{T_{acc}}t \\ \ddot{s}(t) &= \frac{v_c}{T_{acc}} \end{aligned} \tag{14}$$

2. **Constant velocity phase**, $t \in [T_{acc}, t_f - T_{acc}]$. The motion law is defined as:

$$s(t) = b_0 + b_1 t \tag{15}$$

where for continuity reasons:

$$b_1 = v_c \text{ and } s(T_{acc}) = s_0 + \frac{v_c T_{acc}}{2} = b_0 + v_c T_{acc}$$

the motion law and its derivatives are obtained:

$$\begin{aligned} s(t) &= s_0 - \frac{v_c T_{acc}}{2} + v_c t \\ \dot{s}(t) &= v_c \\ \ddot{s}(t) &= 0 \end{aligned} \tag{16}$$

3. **Deceleration phase**, $t \in [t_f - T_{acc}, t_f]$. The motion law is defined as:

$$s(t) = c_0 + c_1 t + c_2 t^2 \tag{17}$$

by inputting continuity and final conditions: $s(t_f) = s_f$, $\dot{s}(t_f) = 0$, and $\dot{s}(T_{acc}) = v_c$, the motion law and its derivatives are obtained:

$$\begin{aligned} s(t) &= s_f - \frac{v_c}{2T_{acc}}(t_f - t)^2 \\ \dot{s}(t) &= \frac{v_c}{T_{acc}}(t_f - t) \\ \ddot{s}(t) &= -\frac{v_c}{T_{acc}} \end{aligned} \tag{18}$$

It should be noted that other conditions must be met in order to identify the trapezoidal trajectory unequivocally. A common condition is that the time length of the acceleration and deceleration phases Ta satisfies the evident constraint $T_{acc} \leq T/2 = (t_f - t_0)/2$. Furthermore, there may be some restrictions on the actuation system's maximum velocity and acceleration. Obviously, these factors have an impact on the trajectory's viability.

Finally, considering the general case $t_0 \neq 0$ with value of $T_{acc} = (t_f - t_0)/3 = T/3$ is chosen, the motion law (in position) is defined as follows:

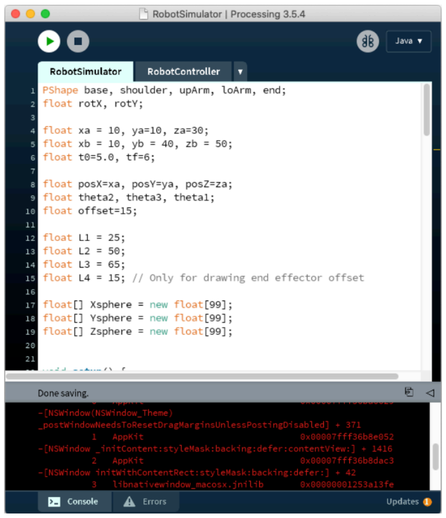
$$s(t) = \begin{cases} s_0 + \frac{9}{4}\left(\frac{\Delta t}{T}\right)^2 (s_f - s_0) & \text{for } t_0 \leq t < t_0 + \frac{T}{3} \\ s_0 + \frac{1}{4}\left(6\frac{\Delta t}{T} - 1\right)(s_f - s_0) & \text{for } t_0 + \frac{T}{3} \leq t < t_f - \frac{T}{3} \\ s_f - \frac{9}{4}\left(1 - \frac{\Delta t}{T}\right)^2 (s_f - s_0) & \text{for } t_f - \frac{T}{3} \leq t < t_f \end{cases} \tag{19}$$

where $\Delta t = t - t_0$ and $T = t_f - t_0$.

3. Experiment

3.1. Setup

In this experiment, we simulate kinematic control on a three-degree-of-freedom robotic manipulator using Processing [1], a Java-based Integrated Development Environment with a rich graphical library that is primarily used for visual design and programming in a visual environment (Figure 5). Processing sketches can be run directly from the IDE, avoiding the need to build and run them from the Terminal. To run the application, click the Play button on the toolbar, which will open the execution window. To stop it, simply use the *Esc* key or click the Stop button on the toolbar.



```
PShape base, shoulder, upArm, loArm, end;
float rotX, rotY;

float xa = 10, ya=10, za=30;
float xb = 10, yb = 40, zb = 50;
float t0=5.0, tf=6;

float posX=xa, posY=ya, posZ=za;
float theta2, theta3, theta1;
float offset=15;

float L1 = 25;
float L2 = 50;
float L3 = 65;
float L4 = 15; // Only for drawing end effector offset

float[] Xsphere = new float[99];
float[] Ysphere = new float[99];
float[] Zsphere = new float[99];
```

```
Done saving.
-[NSWindow(NSWindow_Theme)
_postWindowNeedsToResetDragMarginsUnlessPostingDisabled] + 371
        1   AppKit                          0x00007fff36b8e052
-[NSWindow _initContent:styleMask:backing:defer:contentView:] + 1416
        2   AppKit                          0x00007fff36b8dac3
-[NSWindow initWithContentRect:styleMask:backing:defer:] + 42
        3   libnativewindow_macosx.jnilib   0x00000001253a13fe
```

Figure 5. Processing sketch [3].

The code scripts for this experiment are provided by the instructors and can be downloaded from *google drive* [6]. There are two main scripts given:

- `RobotSimulator.pde`, which consists of the main functions: `setup()` for initializing the environment and loading the robot designs, `draw()` for displaying the environment and the robot arm.

- `RobotController.pde`, which consists of functions that control the robots and are going to be edited to solve the given assignments.

3.2. Robot Arm Visualization and Joint Angles

In this experiment, the manipulator's joint angles, which have been defined as global variables (θ_1, θ_2, θ_3), are adjusted within the `updateParameters()` function. When the sketch is run, if the joint angle variables (in radians) are properly specified, each joint will be at the desired angle. Figure 6 depicts a few examples.

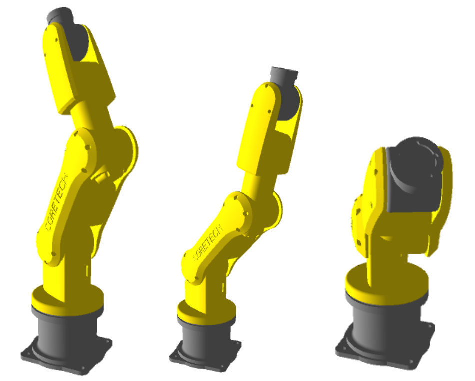

(a) $(\frac{\pi}{3}, \frac{\pi}{3}, \frac{\pi}{3})$ (b) $(\frac{\pi}{3}, \frac{\pi}{4}, \frac{\pi}{6})$ (c) $(\frac{\pi}{8}, \frac{\pi}{10}, \frac{\pi}{12})$

Figure 6. Examples of manipulator's adjusted joint angles (θ_1, θ_2, θ_3).

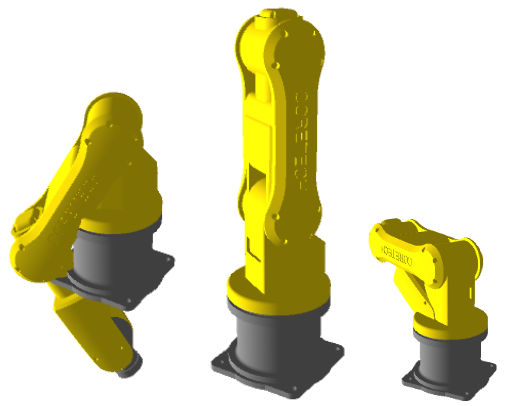

(a) $(-\frac{\pi}{3}, -\frac{\pi}{3}, -\frac{\pi}{3})$ (b) $(\frac{\pi}{3}, \frac{\pi}{2}, \pi)$ (c) $(\frac{\pi}{3}, \pi, \frac{5\pi}{6})$

Figure 7. Examples of error occurred on the manipulator's joint angles (θ_1, θ_2, θ_3).

However, it is possible for the links to overlap each other as the script does not consider the constrain for the joint angles. Several examples of the error are illustrated in Figure 7.

3.3. Robot Arm Inverse Kinematics

In this task, we employ a function named `IK()` that transforms the endpoint position $P_e(x_e, y_e, z_e)$ to manipulator's joint angles (θ_1, θ_2, θ_3). Then, within `updateParameters()` function, we set the values of P_e before invoking the implemented function. By manually changing the sign of θ_3, the two alternative configurations (elbow up and elbow down) stated in Section 2.2 can be produced. Figure 8 depicts some examples of this experiment.

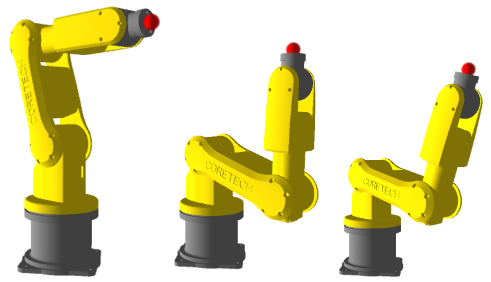

(a) (30, 40, 50) (b) (30, 40, 50) (c) (50, 50, 50)

Figure 8. Examples of inverse kinematics of the manipulator by changing the endpoint position (x_e, y_e, z_e). Note that Figures 8(a) and 8(b) show elbow up and elbow down configurations, respectively.

However, unlike the last experiment, the script disregards the constraint of joint angles, enabling the linkages to overlap. It is also possible to set the endpoint position outside of the attainable range. The simulator will not display the manipulator in this situation. Figure 9 depicts the aforementioned errors.

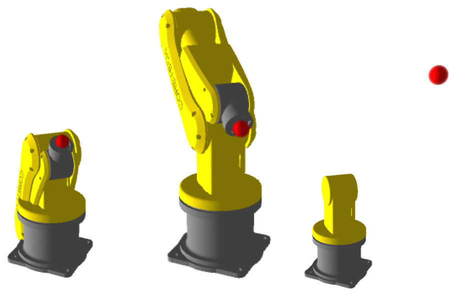

(a) (10, 10, 10) (b) (70, 80, 90) (c) (15, 15, 5)

Figure 9. Examples of error of the manipulator's inverse kinematics.

3.4. Robot Arm Trajectory Planning

In this task, we adjust the endpoint values as a function of time. On the sketch, the time is stored in seconds in the global variable `gTime`. To begin, we'll create a linear trajectory between two places, with the position changing proportionally to time (e.g., no motion law). Then, using simple boundary conditions, we will create a function to determine the motion law $s(t)$ and use it to adjust the position. Section 4.1 will go through the distinctions between these two motion laws. In this experiment, we use the 5th-order motion law (Section 2.3.3). The manipulator's motion is depicted in Figure 10.

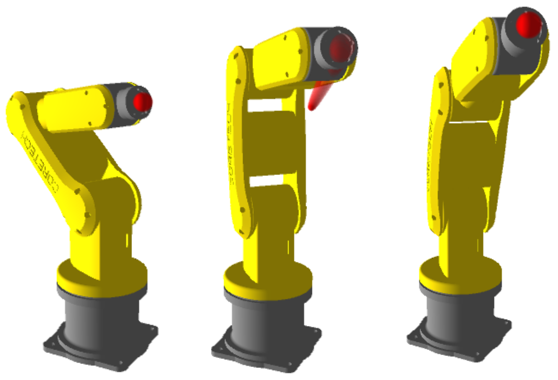

(a) $t = t_0$ (b) $t_0 < t < t_f$ (c) $t = t_f$

Figure 10. Manipulator's endpoint motion from $P_a(10, 20, 40)$ to $P_b(60, 50, 70)$.

4. Results and Discussions

4.1. Task 1

Discussion: *Plot the position, velocity and acceleration of the endpoint in time (could be a single axis) both with and without using the 5th-order polynomial motion law. Discuss about the difference between both. You can computationally calculate the velocity as the difference between the current position and the previous position, divided by the elapsed time between cycles, and the acceleration as the difference between current and previous velocity divided by elapsed time. To keep track of previous velocity and time, you can use global variables defined at the top of the file* `RobotController`*. To print values in the Processing terminal, you can use the function* `println()`.

The concept of linear and 5th-order interpolation has been discussed in Section 2.3.1 and 2.3.3. These methods were implemented to the manipulator's motion and then the endpoint's positions for each method were recorded under the same running time. Then velocity and acceleration were calculated using formula:

$$(t'_i,\ v_i) = \left(\frac{(t_{i+1} + t_i)}{2},\ \frac{(x_{i+1} - x_i)}{(t_{i+1} - t_i)} \right) \tag{20}$$

$$(t''_i,\ a_i) = \left(\frac{(t'_{i+1} + t'_i)}{2},\ \frac{(v_{i+1} - v_i)}{(t'_{i+1} - t'_i)} \right) \tag{21}$$

where v_i and a_i denote the ith velocity and acceleration of the endpoint, respectively. Moreover, t'_i and t''_i denote the time data used to plot the velocity and acceleration graph respectively. Finally, the plot for position, velocity, and acceleration was plotted on the same figure to ease the comparison as shown in Figure 11 (x-axis).

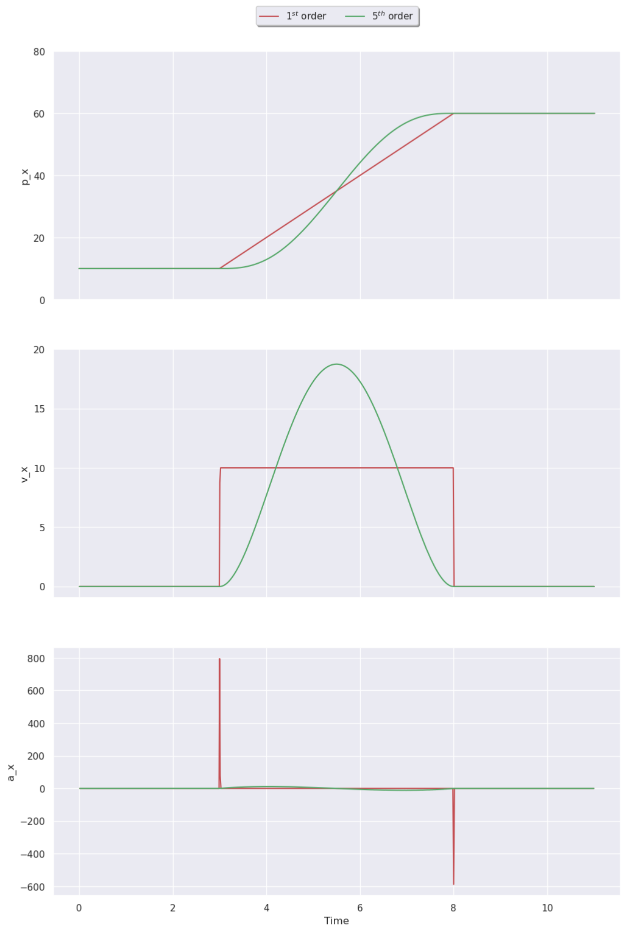


Figure 11. Position, velocity, and acceleration of the manipulator's endpoint on x-axis from $P_a(10, 20, 40)$, $t_0 = 3$ to $P_b(60, 50, 70)$, $t_f = 8$ for both linear and 5th-order interpolation.

As shown in the above figure, there are significant difference between interpolations for all categories (position, velocity, and acceleration). For position P_x, the method's name represents the shape of the plot: in the linear motion, the position value is constantly changing, while in the 5th-order motion, the shape is warped until it reaches the final position. Then we take the position's derivative, resulting in a 4th-order (parabolic) and a constant velocity value for the 5th-order interpolation and the linear interpolation, respectively. Lastly, an acceleration curve is formed for the polynomial motion (Figure 12) while the acceleration value for linear motion remains zero.

In addition, the 5th-order interpolation guarantees the po-

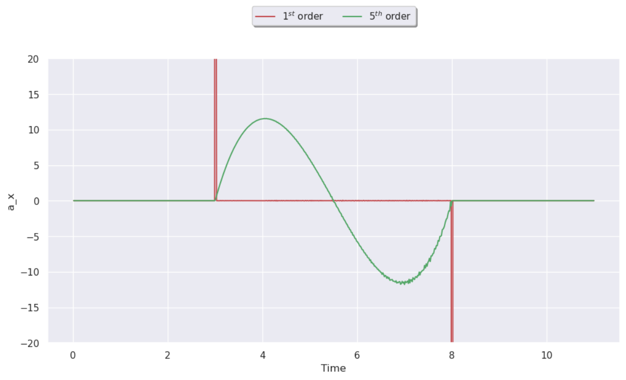


Figure 12. Detailed look of the 5th-order interpolation acceleration.

sition, velocity, and acceleration continuity as it does not show any sign immediate value change. However, some small spikes occur at the end of the movement, which might result from the discrete-time data.

On the other hand, smoothness (continuity) does not occur in linear interpolation. In order to achieve a certain velocity value from zero initial state, the manipulator has to accelerate swiftly (indicated by the spike in the acceleration plot), which also results in an immediate velocity change. Moreover, the manipulator also has to decelerate in the same manner as the acceleration at the end of the movement.

We want the position and orientation of the manipulator to vary smoothly with time, as smoothness means reducing vibration in the structure as well as peak acceleration of the robot. As already discussed in Section 2.3.1, we avoid discontinuity by utilizing a higher degree of interpolation, such as 5th-order polynomial [7].

4.2. Task 2

Discussion: *Change the geometric path such that instead of a line, it follows a circle. Implement it in your* `updateParameters()` *function*

In this task, we ignore one of three axes (y-axis) to simplify the calculations. Using the inverse kinematics method, the endpoint position is changed over time by defining its coordinate as parametric equations in polar coordinate (r, ψ), where r is the radius and ψ is the angle formed in the $x - z$ plane ranging from 0 to 2π. The following is the defined Cartesian coordinate:

$$(x_e, y_e, z_e) = \begin{cases} (x_0 + r, y_0, z_0) & \text{for } t \leq t_0 \\ (x_0 + r\cos(2\pi s(t)), & \text{for } t_0 < t \leq t_f \\ y_0, z_0 + r\sin(2\pi s(t))) & \\ (x_0 + r, y_0, z_0) & \text{for } t > t_f \end{cases}$$

where $(x_0 + r, y_0, z_0)$ and $s(t)$ are the endpoint's initial position and the interpolation function respectively. Note that in a circular trajectory, the initial and final positions are similar. Moreover, here we add r to the initial x-axis position so that the endpoint will move in a circular path with its center located at (x_0, z_0) (seen from above $x - z$ plane). Figure 13 depicts the motion of the endpoint, whereas Figure 14 illustrates the path. In this case, the 5th-order polynomial is employed to create a smooth movement.

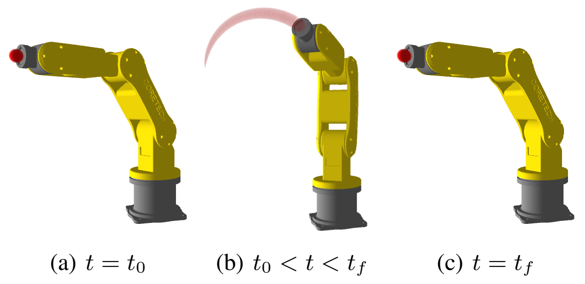


Figure 13. Manipulator's endpoint circular motion with initial position $(50, 50, 40)$ and radius $r = 30$.

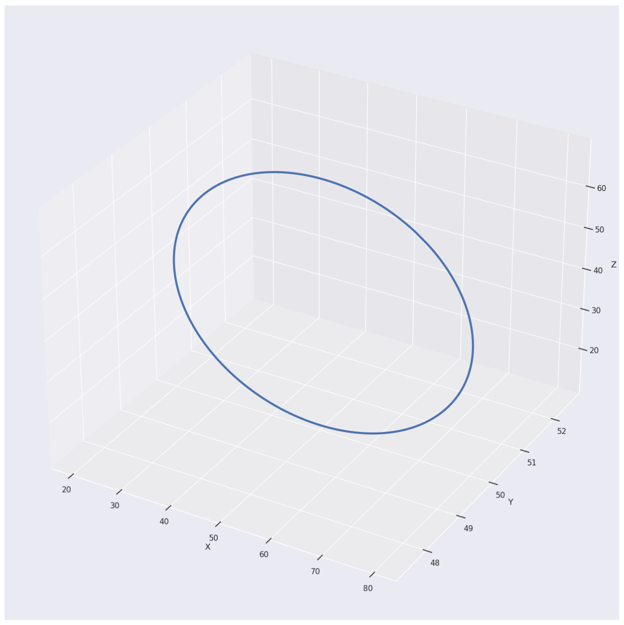


Figure 14. The circular path created by the endpoint movement.

4.3. Task 3

Discussion: *Search for a different motion law (trapezoidal interpolation, 3rd-order polynomial interpolation) and implement it for your motion. Discuss about the differences with the used 5th-order polynomial interpolation.*

In addition to the interpolation function discussed in Section 4.1, we did additional experimentation on trapezoidal and

3rd-order polynomial interpolation (discussed in Sections 2.3.2 and 2.3.4). The velocity and acceleration profiles for both functions were also calculated using Equations 20 and 21. The position, velocity, and acceleration plot of the endpoint for each interpolation function are illustrated in Figure 15.

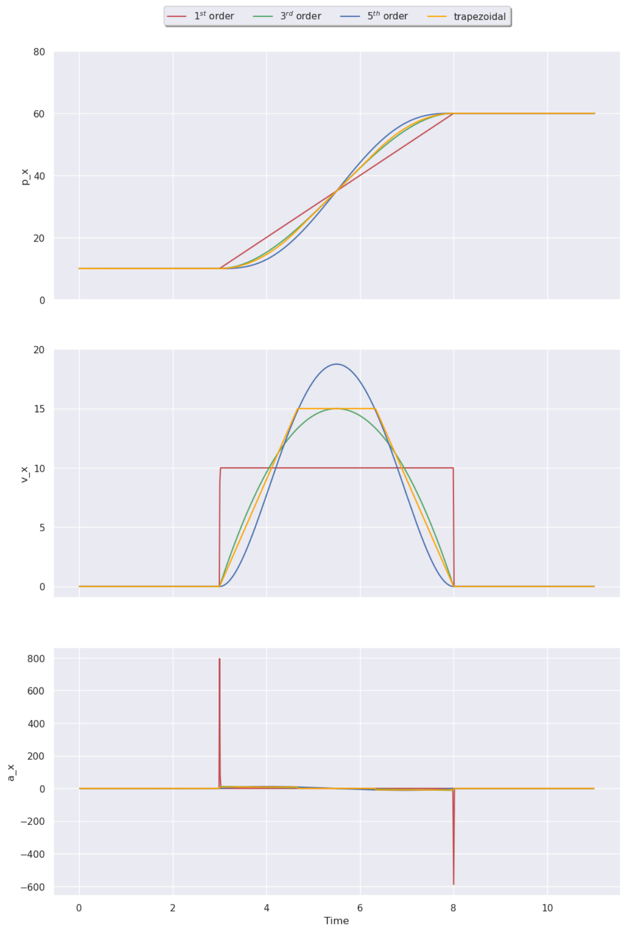


Figure 15. Position, velocity, and acceleration of the manipulator's endpoint on x-axis from $P_a(10, 20, 40)$, $t_0 = 3$ to $P_b(60, 50, 70)$, $t_f = 8$ for all interpolation functions.

From the position plot, it can be observed that the additional interpolation functions have the same s-curve trend as the 5th-order polynomial. Moreover, during the beginning of the movement, the linear interpolation is the fastest, while the 5th-order polynomial is the slowest. After that, all of the interpolation functions meet at the halfway of the total distance, which occurs at $\frac{1}{2}$ of the total movement time after the first movement. Then, 5th-order interpolation leads the way, whereas the linear interpolation slows down and becomes the very last to reach the final position.

If we look at the velocity profile of each interpolation method, it can be deduced that the linear interpolation stays at its maximum velocity the longest, followed by the trapezoidal motion law. On the other hand, the polynomial functions possess a parabolic curve, with the 5th-order's peak being higher than the 3rd-order. This means that the 3rd-order motion law requires a smaller maximum velocity to achieve the same displacement for the same period (trapezoidal motion too, as it has the same peak as the 3rd-order). Moreover, the 5th-order motion takes the longest time to accelerate but is able to achieve the highest velocity among the other profiles. This observation explains why the 5th-order is late in the first half but is the first to reach the end point.

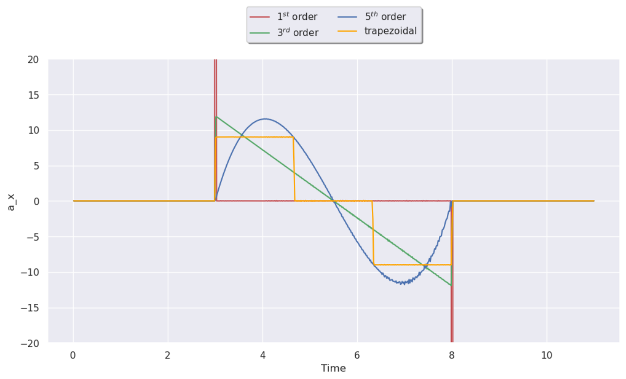


Figure 16. Detailed look of the trapezoid, 3rd-order, and 5th-order interpolation acceleration.

Lastly, we observe the detailed look of the acceleration plot shown in Figure 16. It can be seen that, unlike the 5th-order motion law, the 3rd-order motion law does not show a curvy plot, instead, it jumps from zero to a certain value (quite similar to the linear motion law but with much smaller amplitude), then constantly decreases until it jumps again to zero. Similarly, the trapezoidal interpolation has a leap at the beginning of the movement, but instead of descending, it remains constant (acceleration phase); then it falls to zero and remains constant twice (constant velocity and deceleration phases) until it surges to zero. Moreover, the trapezoidal movement requires a smaller acceleration to achieve the same displacement for the same period. It is interesting to note that if we increase the constant velocity phase time interval to be equal to the elapsed time, it will become the linear motion law (higher acceleration value is required to achieve the same displacement); whereas if it is reduced to zero, it will behave in the same manner as the 5th-order motion law (considerably high amplitude, but not curvy).

Overall, there are trade-offs behind the benefits of each type of interpolations. For polynomials, the higher the order, the smoother the velocity and acceleration profile. However, higher-order means more coefficient, which might result in costlier computation. Vice versa, the trapezoidal motion law requires a smaller number of coefficients but the movement is slower than a high-order polynomial function.

5. Conclusion

This experiment taught us how to simulate a manipulator (robotics arm) using the Processing IDE. We investigate the relationship between the joint angles and the end-point position by implementing the inverse kinematics principle. Finally, we observe the four trajectory path interpolation functions: linear, 3rd-order polynomial, 5th-order polynomial, and trapezoid.

References

[1] *Processing*. URL: https://processing.org/ (see pp. 1, 4).

[2] Xiaoyu Wang and Luc Baron. "Topology and Geometry of Serial and Parallel Manipulators". In: Apr. 2008. DOI: 10.5772/5363 (see p. 1).

[3] Salazar Jose V. "Robotic Arm Geometry Simple Trajectory Planning". In: 2022. URL: https://drive.google.com/file/d/1GPyetVl7jfSt9QEo3bdX2_oSOA66m_LE/view (see pp. 1, 2, 4).

[4] Antonio Barrientos. *Fundamentos de Robótica*. Jan. 2007 (see p. 2).

[5] Luigi Biagiotti and Claudio Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Softcover reprint of hardcover 1st ed. 2008. Springer, Oct. 2010 (see p. 3).

[6] Salazar Jose V. "RobotSimulator.zip". In: 2022. URL: https://drive.google.com/file/d/1-NbZSrIeYjkM3Sytt2pZs9K8QsZZ1oFa/view (see p. 4).

[7] Robot Academy. *1D polynomial trajectory*. URL: https://robotacademy.net.au/lesson/1d-polynomial-trajectory/ (see p. 7).