EXPERIMENT 4: TRAJECTORY PLANNING

The purpose of this experiment is to learn methods of computing trajectories that describe the desired motion of manipulators in 3D space. The following topics will be studied in this experiment.

Topics Covered

- The notion of trajectory
- Cubic and higher-order polynomials
- Cartesian space trajectory planning

Prerequisites

- The robot has been setup and tested. See the Quick Start Guide for details.
- You have access to the User Manual.
- You are familiar with the basics of MATLAB® and SIMULINK®.

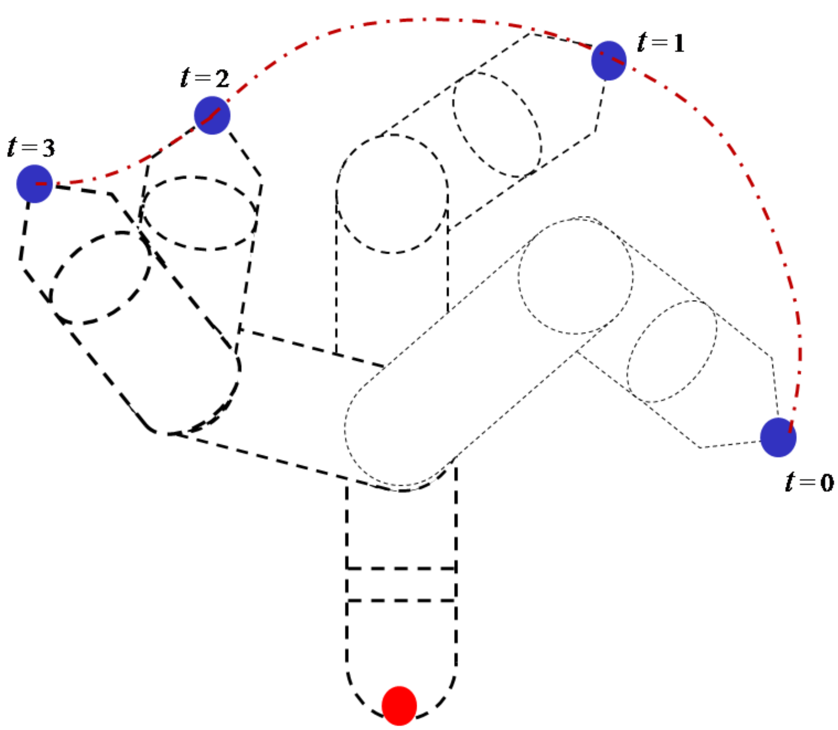

Figure 0.1: Robot trajectory is designed such that the manipulator moves from its initial position to the desired goal position in a smooth manner.

1 Background

In the context of robotics, *trajectory* refers to the time history of position, velocity and acceleration for each degree of freedom, in either the joint space or Cartesian space.

In various robotics applications, the end-user cannot be involved in determining complicated functions for the robot motion. Therefore, the robot program should be able to create the motion path of the robot using limited information such as the desired goal pose (position and orientation) of the robot end-effector, the attached tool, or a set of way points (via points) along the desired path.

In this laboratory you will learn how robot trajectories can be computed, represented, generated and applied to robots.

1.1 Path Description and Generation

As mentioned earlier, the end users of robot manipulators are mostly concerned about the motion of the tool frame of the robot attached to the end-effector, relative to the base or station frame. Therefore, in this lab we focus on the Cartesian-space trajectory planning, although these methods can be directly applied to joint-space schemes.

The motion of the robot includes both position and orientation. Since we are dealing with a 4-DOF robotic manipulator, if we want to control the position of the robot end-effector in space, we require three degrees of freedom. We use the rotation of the last joint (wrist) of the robot as the remaining degree of freedom of the robot that can be specified and controlled. For complete control of the 6 possible degrees of freedom of the end effector, the manipulator would need to be equipped with a complete 3-DOF wrist.

In most cases, it is necessary to specify the movement of the robot in a detailed way to achieve a specific task. Consider a factory in which a robotic arm has to reach a part without touching other obstacles around it. The user needs to provide a sequence of **way points** that represent intermediate points between the initial and final points. Depending on the design scheme, each way point can include different information. In a Cartesian scheme, each way point describes the position and orientation of the robot. In joint-space scheme, each way point represents all of the joint angles.

Along with the spatial constraints each way point usually contains some temporal attributes, such as the time elapsed between the way points. Having a smooth motion is also usually a constraint that is placed on the motion and controlled using various planning techniques. Usually, a trajectory function that is continuous and has a continuous first derivative will generate a ”smooth” trajectory. For a ”minimum jerk” trajectory, a continuous second order derivative is desirable. Minimum jerk motions minimize wear on the robot mechanisms and are desirable for many applications including accurate image analysis for robotic vision inspection.

1.2 Cubic vs. Quintic Polynomial Trajectory Planning

In order to achieve the desired smooth motion *between* any two consecutive way points, say $\mathbf{X}_0$ and $\mathbf{X}_1$, we need to satisfy the following constraints

1. $\mathbf{X}(t = 0) = \mathbf{X}_0$
2. $\mathbf{X}(t = 1) = \mathbf{X}_1$
3. $\dot{\mathbf{X}}(t = 0) = \mathbf{V}_0$
4. $\dot{\mathbf{X}}(t = 1) = \mathbf{V}_1$,

where the last two constraints are necessary for smooth operation between the way points and $\mathbf{V}_0$ and $\mathbf{V}_1$ should be zero for the initial and final way points. In order to satisfy the above four constraints, a cubic polynomial (that has for parameters) is required. As mentioned earlier, for a minimum jerk trajectory, it is desirable to have a continuous second-order derivative which yields the following two constraints

5. $\ddot{\mathbf{X}}(t = 0) = 0$
6. $\ddot{\mathbf{X}}(t = 1) = 0$.

In order to satisfy the above six constraints a quintic polynomial is required.

2 Pre-Lab Questions

1. Open the MATLAB script called mico4DOF_generate_cubic.m and explain what the code is doing. Be sure to identify the key portions of the script.
2. Run the script with the default way points and plot the trajectory over time. Does the trajectory go through the way points? Is it smooth?
3. Describe how the trajectory is generated and plot the planar motion of the robot (y , z planar motion).
4. How can you show that the derivative of the trajectory is zero at the way points? What does this mean when you apply the trajectory to the robot?
5. Change the way points so that the robot moves from $x = -0.4$ to $x = 0.4$ while $y = +0.2$, $z = 0.4$ and $\theta_4 = 0$ (taking 0.1 steps for x; you will need 9 way points). Tune the variable speed (between 0 to 0.01) and plot the trajectories until you find a very smooth motion. Assuming that the robot begins to move at 2 s and completes the movement after 60 s, what is the ideal speed?
6. Do you think quintic polynomials are needed for the above way points? Discuss your conclusions. Open and run the mico4DOF_generate_cubic.m polynomials for the above way points with various speeds and discuss your findings.

3 Lab Experiments

3.1 Simulation

The QUARC model for this exercise is ”MICO_Trajectory_Planning_Simulation.mdl” a snapshot of which shown in Figure 3.1. This model works with the two trajectory generation MATLAB scripts, mico_generate_cubic.m and mico_generate_quintic.m.

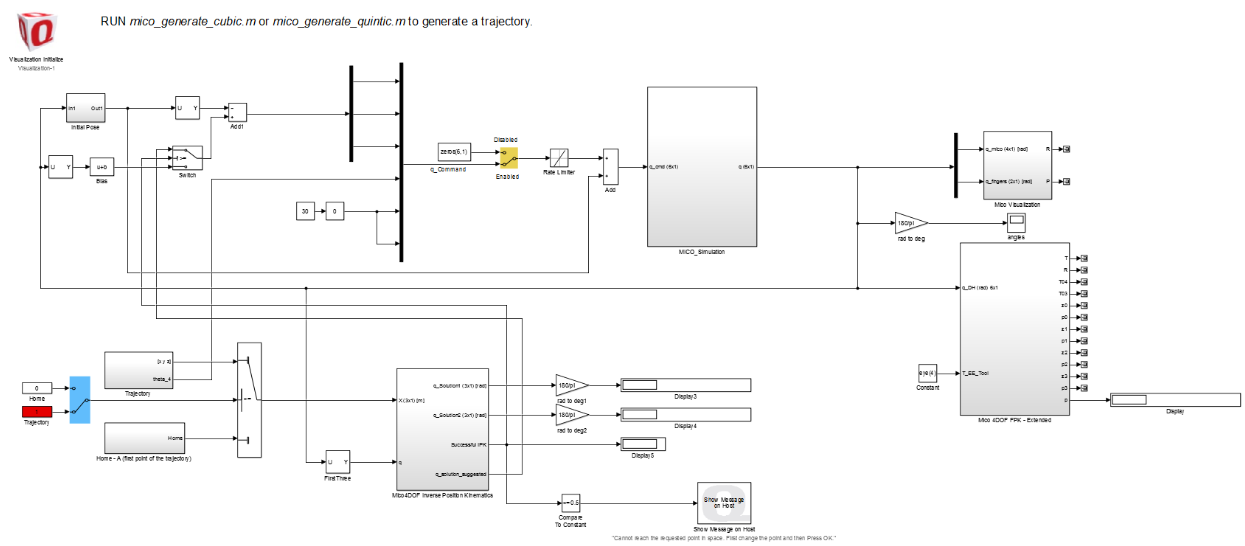

Figure 3.1: Snapshot of the controller model ”MICO_Trajectory_Planning_Simulation.mdl”

mico_generate_cubic.m generates a smooth trajectory based on cubic polynomials given a set of way points (or via points) and the corresponding velocities. **mico_generate_cubic.m** generates a smooth trajectory based on quintic polynomials given a set of way points and the corresponding velocities and zero accelerations. These two scripts generate a trajectory in the MATLAB workspace called ”Traj”. The QUARC model ”MICO_Trajectory_Planning_Simulation.mdl” applies this workspace variable to the model of the robot arm.

The MICO Simulation block, receives the joint angles and computes the position of the end-effector. Compile and run the model, then follow the procedure outlined below. Make observations on the motion of the robot, and answer the associated questions.

1. Use the default way points that were generated in the Pre-Lab for circular motion of the arm.
2. Open the ”MICO_Trajectory_Planning_Simulation.mdl” model, compile and run it making sure the yellow and blue switches are set to *trajectory*. Observe the motion of the virtual robot.
3. Use the set of way points you created from the Pre-Lab exercise section and run the robot. Compile the QUARC model ”MICO_Trajectory_Planning_Simulation.mdl” and run it again. Describe the motion of the virtual robot.

3.2 Experiment

In this section, you will experimentally evaluate the trajectory generation algorithms on the robot arm. During the experiment, always keep a safe distance from the robot and evaluate your results on the virtual robot first before applying your commands to the actual robot platform.

The QUARC model for this experiment is called ”MICO_Trajectory_Planning_Experiment.mdl” shown in Figure 3.2 which also works with the two trajectory generation MATLAB scripts that were studied before (cubic and quintic).

Before running the model, manually move the robot arm in a comfortable pose where the robot end-effector is away from the table (preferably a elbow-up pose), and turn on the robot to hold its pose. Make sure that the robot is disabled (the yellow manual switch is set to ”Disabled”) and the blue manual switch is set to ”Home”.

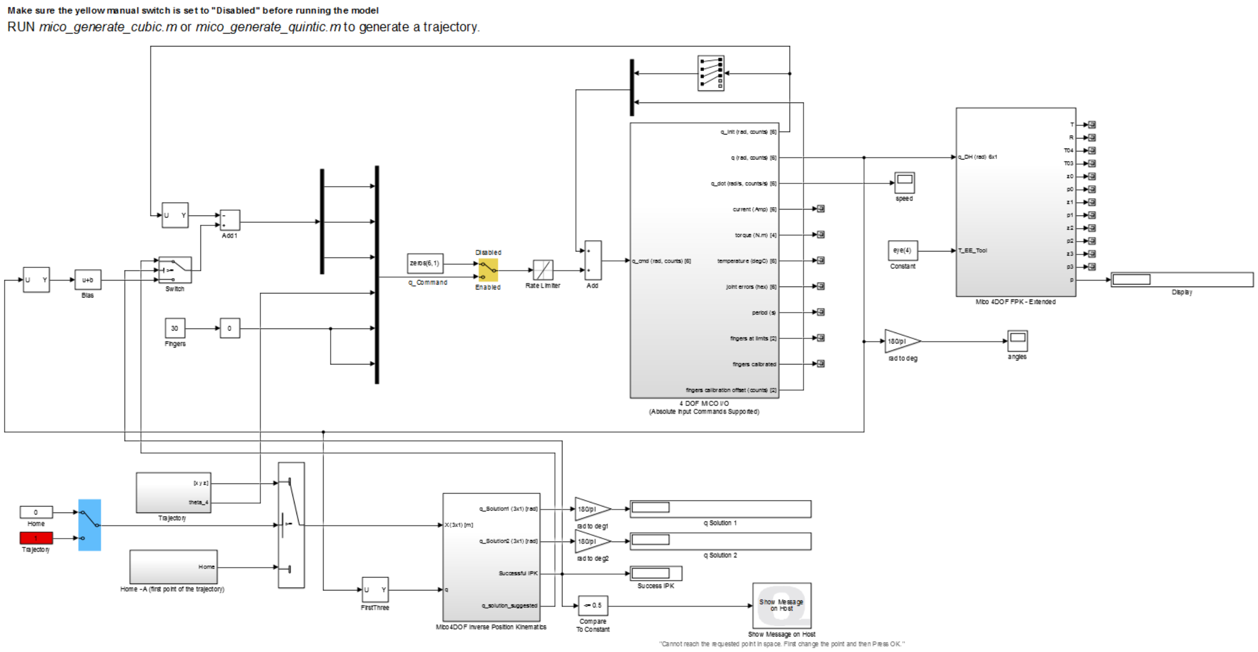

Figure 3.2: Snapshot of the controller model ”MICO_Trajectory_Planning_Experiment.mdl”

Go through the steps below and answer the questions.

1. Implement the way points that were created for a circular path. From the plots, confirm that the desired motion is achieved.
2. Open ”MICO_Trajectory_Planning_Experiment.mdl”, compile and run it making sure the blue switch is set to ”Home”. When the robot is in the home position, toggle the blue manual switch and observe the motion of the robot.

Caution **Be sure to set the Ports in the 4-DOF MICO I/O block to the correct ports for your serial card. For more information, refer to the User Manual.**

3. Use the set of way points you created from the Pre-Lab exercise section, and evaluated with the virtual robot in simulation. Run the trajectory generation mathscript, and make sure that the desired motion along the x axis is achieved (look at the plots).
4. Compile the QUARC model ”MICO_Trajectory_Planning_Experiment.mdl”, connect to the target and run it. Describe the motion of the robot.
5. How would you use trajectory generation for assembly line automation in a factory?

Quanser Inc.
119 Spy Court
Markham, Ontario
L3R 5H6
Canada
info@quanser.com
Phone: 1-905-940-3575
Fax: 1-905-940-3576

Printed in Markham, Ontario.

For more information on the solutions Quanser Inc. offers, please visit the web site at:
http://www.quanser.com