Basic Autonomous Robot Control for a Simulated UR5E

Charlie Caputo

***Abstract* – I was tasked with moving a UR5E robot through four points using autonomous control, starter codes, and simulators. In this paper I will discuss my methods to finding a solution and the roadblocks that I ran into. Throughout the process, I learned techniques that helped me debug code and think of new ways to arrive at a solution.**

I. Introduction

For this project we were given the task of making a simulated UR5E robot move. Specifically, the goal was to ensure that the simulated robot would make controlled and slow movements through each of the points. There are four points that the robot needs to flow through as depicted in (Fig. 1).

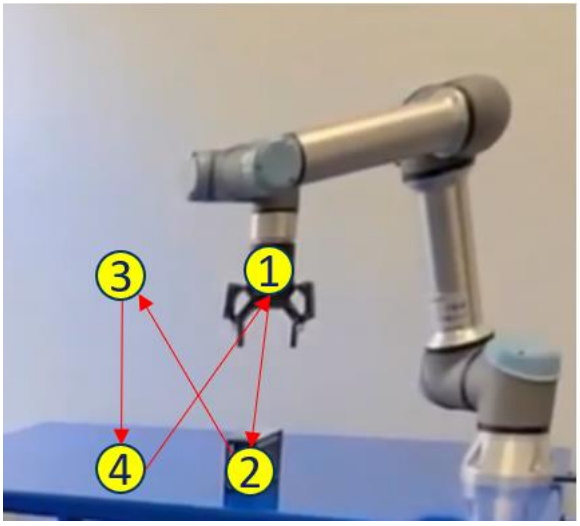

Fig. 1: 1) starting at a safe initial position, 2) Reach a position to pick up the object, 3) Move to a save spot above the target position, 4) Place the object at the target position [1]

In this paper I will first discuss the tools that were given to achieve this goal. Then how the problem was approached and solved. Finally, the results of the task will be revealed along with an in-depth discussion of the problems that ensued and what could have been done better throughout the process.

II. Methods

During the ISE 360-CSC 475/592 lectures we were shown a multitude of resources such as starter codes, a simulator, and ROS. The starter codes were written by Dr. Hamed Saeidi both in C++ and Python. The first important code is the "manual_initialization.py" which is used to set the robot to an initial position by sending the joint motors specific positions. While the code is not necessary for the other codes to run, it is still useful as it starts the robot in the same position every time. This file specifically sets all of the joint positions to zero except for elbow joint and shoulder lift joint which were set to -pi/4 and pi/4 respectively [2].

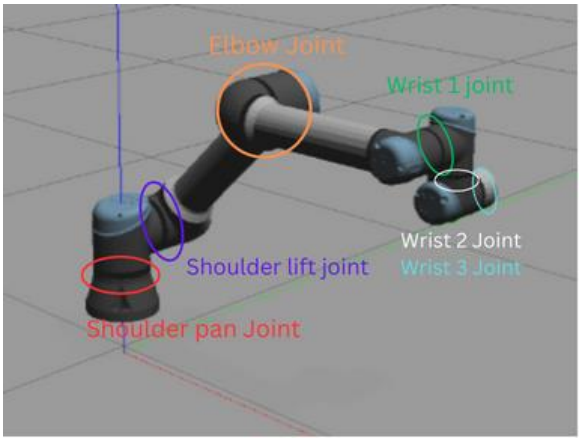

Fig. 2: The joints and their names which are color coordinated. The tip of the robot is designated as the end-effector and is the metal flange after wrist 3 joint

The next important code is the "ur5e_controller.cpp" which allows us to control the robot. It defines a chain of links and joints using the Denavit-Hartenberg (DH) parameters [2]. These parameters essentially tell the code the specifics of the robot based off the base of the robot 'zeroed' position). These are usually given by the company that makes the robot. These parameters allow us to use matrix multiplication to move the robot to desired positions. After defining the DH parameters, the code then goes on to get the initial joint positions, then computes the kinematics (received from "task_space_traj.cpp") of the desired position, and finally publishes control commands to the robot in order to achieve the desired end-effector position (the piece where the tool is mounted) [2].

The next code that is important is the "task_space_traj.cpp". This code is used to receive a set of planned coordinates and orchestrate a trajectory by planning the steps of the trajectory [2]. It then sends this planned trajectory to the "ur5e_controller.cpp" [2]. This code also ensures that the UR5E moves in a smooth and controlled manor by using the "Reflexxes Motion Library" which is an opensource software [3].

The last Important code is the starter code that we were given called "simple_planner.py" which I renamed to "point_planner.py". We were given this code to alter in order to make the robot function. It creates a "Plan" topic which allows us to plan points by using the following notation: "plan_point#.linear.A = #". The first # symbol is the point number which ranges 1-4 for this specific case since we are only trying to move the end-effector through 4 points. The "A" represents the coordinate we are using, cartesian and polar, such as x, y, z, roll, yaw, pitch. The second # represents the specific coordinate we want the robot to reach for or what angle we want to achieve with specific wrist joints. The code uses one of these per for each "A" value available [2].

We were also given a simulator named Gazebo. This simulator allows us to test the codes and see how the robot reacts which will be useful for ensuring that we don't break the real one when we eventually get to it.

The last tool we were given is integrated in ROS as a program called "rqt_gui" this allows us to manually send the robot position commands. This was useful to me as it was the first tool that I used to solve this problem.

I started solving this problem by finding the points that I wanted to send the end-effector to by using "rqt_gui" to do trial and error in order to understand how each value affected the robot's movement. After figuring out each of the desired points/ coordinates, I used the "point_planner.py" to send the four points to "task_space_traj.cpp". After interpreting the plan point, it then sent instructions to the "ur5e_controller" which in turn controlled the simulated robot and moved the end-effector to the specified points. The interaction of these modules can be summed by (Fig. 3):

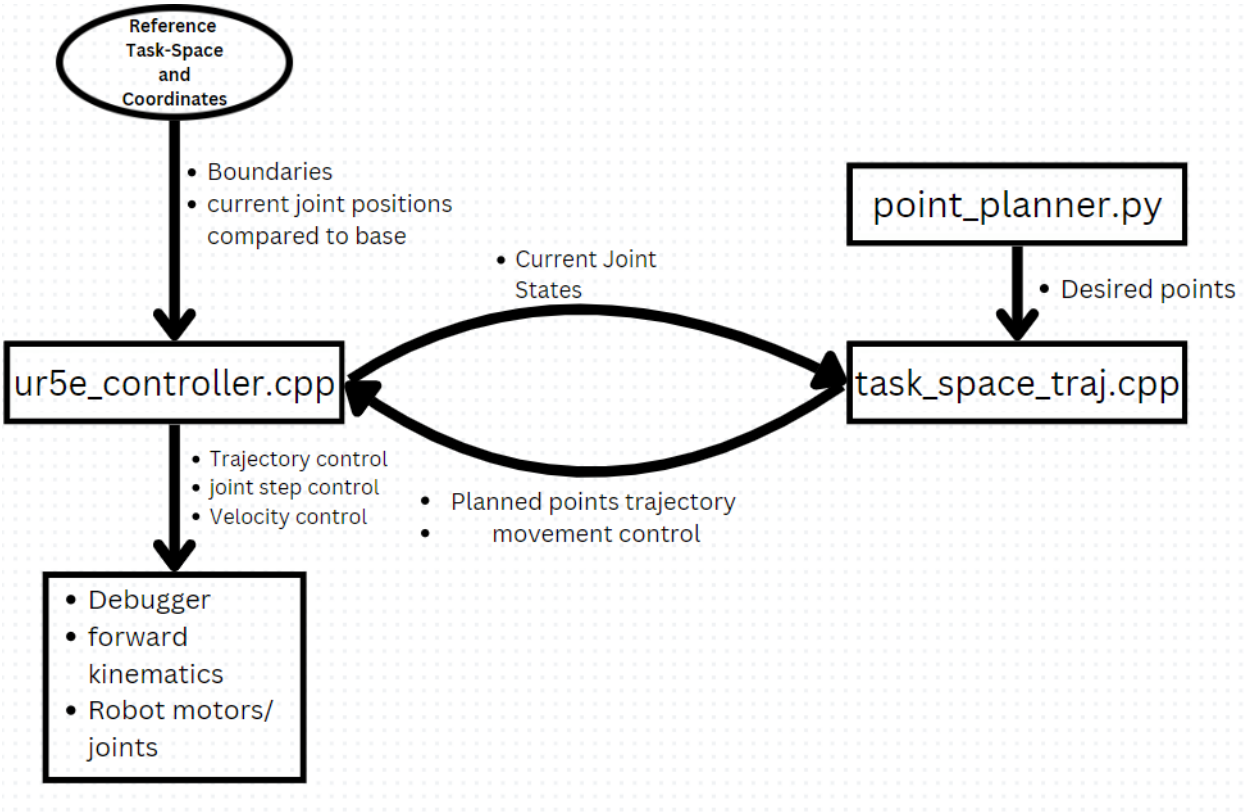

Fig. 3: A diagram of the work flow and how all of the codes are connected

III. Discussions and Conclusions

Throughout the process there were a couple of hiccups. The first problem was that the simulator would crash any time I sent a coordinate that was out of reach for the robot. After the first time and taking 10 minutes to re initialize the program, I decided it would be smarter to take baby steps from known positions so that the program wouldn't crash again. The second problem was that I didn't realize that I needed to make the point_planner.py in a ROS package so that it could be ran properly with all of the other modules. I had realized my mistake as soon as I ran it the first time and this was a quick fix. I used the "catkin_create_pkg" and made the same messages that my professor used in the starter file. All in all, the project was successful and I was able to get the robotic arm to smoothly move through the four points.

References

[1] H. Saeidi, “ISE 360-CSC 475/592: Robotics Reports and Presentation for Phase: Robot Modeling and Control”

[2] Saeidi, H (2022) ur5e_control [Source code]. https://github.com/hsaeidi-uncw/ur5e_control.git

[3] Reflexxes Motion Libraries: The Reflexxes Motion Libraries – An Overview. Reflexxes.ws. http://reflexxes.ws/software/typeiirml/v1.2.6/docs/page__reflexxes_motion_libraries.html