Feroze Naina M Dheen Mohamed Ismail
Team D - HARP
Teammates – Alex Brinkman, Rick Shanor, Abhishek Bhatia, Lekha Mohan
ILR04
Nov. 13, 2015

Individual Progress

Over the last two weeks, I worked on designing the PCB schematic for our suction gripper controller and planning the PR2 arm trajectory using the MoveIt library.

I created the initial Eagle schematic after discussing about the circuits required for the pressure sensor with Alex. We decided to use through-hole components for easier soldering. I looked through the datasheets to find the power consumption of the relays and indicator LEDs to calculate resistor values. I then handed the schematic over to Rick who swapped the relays for a cheaper unit available on Amazon. He also designed the board layout, finalized the enclosure and changed the connectors to DB-9 for convenience.

We started using the MoveIt planner for PR2 arm as it enabled us to quickly switch and test with a number of geometric and control-based planners. We will eventually switch over to the search-based planner plugin being developed by the SBPL. We were compelled to use a deprecated version of the MoveIt package as the robot was running on ROS Groovy. There was no clear indication in MoveIt documentation about which features were missing in that version. This caused us issues while using the MoveIt tutorials and documentation.

The MoveIt planner has three interfaces – rviz, C++ and python. Initially, I worked with the rviz interface to understand the internal architecture of the MoveIt pipeline. In MoveIt, we have to define the robot model, planner, start state and the goal state. I first worked with the rviz version. I worked on switching planners, setting the goal pose and visualizing the trajectory. We settled on using the C++ interface as it had the maximum flexibility and also because the python interface was missing certain function bindings.

For our project we required a service node which calls the MoveIt planner. This is done so that it can be interfaced with the smach state controller. The move_arm_server.cpp file was created which advertises the service /move_arm. The service request consists of a PoseStamped message. PoseStamped was used instead of Pose in order to keep track of time while debugging. The orientation was defined in quaternion format. The service response was a Boolean value to indicate if the planning was successful.

I also created a move_arm_client.py node for testing purposes. It calls the move_arm_server with a PoseStamped request.

In figure 1 (a), you can see the trailing path of the trajectory generated by MoveIt planner as the robot moves from the default pose to target pose. Figure 1 (b) shows the final configuration of the arm after it has reached the target pose.

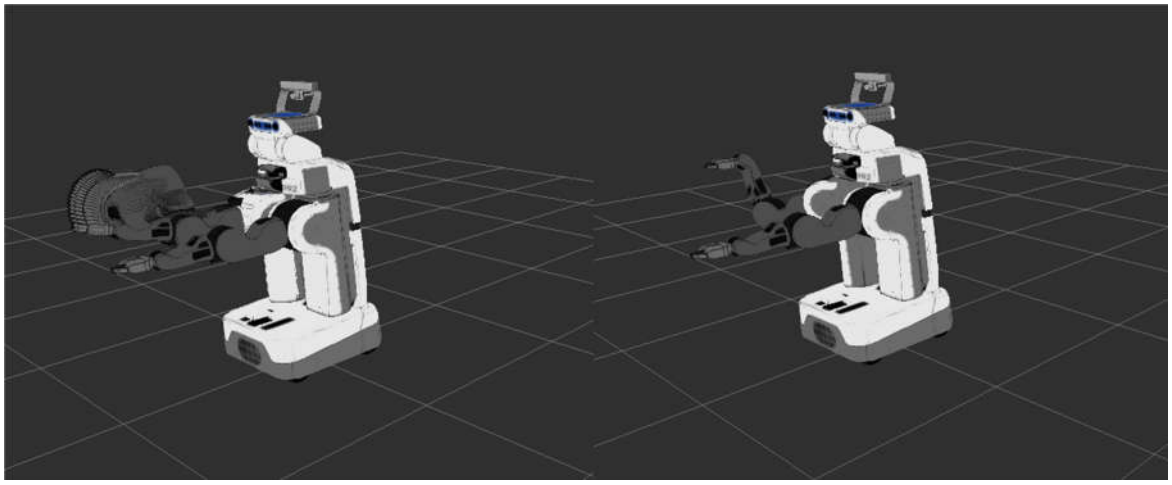

Figure 1: (a) Generated MoveIt trajectory; (b) Final goal state

Last week, I successfully tested the Inverse Kinematics based arm trajectory planner on the actual PR2 robot. The program was loaded onto the PR2's computer and executed.

Challenges

The biggest issue I faced was to understand how multi-threading is used inside ROS and MoveIt frameworks. I spent a lot of time trying to convert the MoveIt C++ code into a ROS service node. The MoveIt planning service was a blocking call. This meant that it had to be called inside its own thread or it would be blocked by the ROS service function call. To fix this, I had to initialize an asynchronous spinner in ROS. I also learned to correctly use member initialization lists in C++ and when they are required. I am grateful to Allard for helping me with these topics outside the TA office hours.

Another minor issue was the PR2 demo launch file had the starting arm pose outside the boundary limits. This prevented MoveIt from planning any path from the default PR2 pose so I had to move the right elbow joint into the boundary. Later, Alex helped me find and change the boundary value in the URDF file.

I had also tried getting collision objects to work but could not follow the MoveIt documentation as their listed function methods were not implemented in my deprecated version of MoveIt. I'm going through their code examples to find alternative methods to add collision boundaries to the planner.

Team Work

Alex found that messages from a ROS Indigo MoveIt node were not compatible with the PR2 running ROS indigo. Figure 2 shows the custom gripper added to the PR2 URDF and visualized in Gazebo by Alex. He also added the collision model for the gripper.

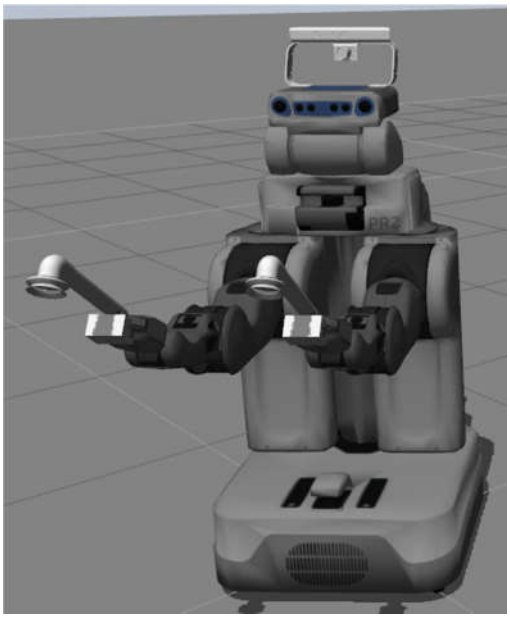

Figure 2: Suction gripper was added to the URDF

Alex created a node to publish the pose of the interactive marker in rviz. This would allow us to graphically set and visualize the target pose while testing the system. He also worked on moving the PR2 base using the move_base package.

Rick worked on creating the PCB board layout and made changes to the schematic after getting feedback from the TAs.

Alex, Rick and Bhatia worked on iterating and improving the mechanical designs for the suction gripper. We purchased a shop-vac and designed the suction tube to fit the shop-vac. They also designed a mechanism for hardware-in-the loop testing which would allow us to test and calibrate the gripper position without damaging our actual suction gripper.

Abhishek, Rick and Lekha worked together on the perception system. An algorithm was developed to remove the shelf from the point cloud. A Euclidean clustering algorithm was implemented and the complete pipeline for perception was created. The current algorithm identifies the grasp position within 3cm in over 56% of the attempts. They will be working on improving the accuracy to 70% for the Fall Validation Experiment.

Plans

For this week, I will be working on developing a simulation of the entire planning pipeline. In simulation, we will programmatically make the PR2 base move along one axis, move the arm to the center of a shelf bin and then move the arm to the top of the order bin. This would be repeated for each of the 9 shelf bins. I will be working with Alex on this task. I will also be soldering the PCB for suction gripper controller.

For the perception subsystem, we will procure the laptop for running the Kinect2 ROS node and test the perception algorithms using real-time data from the Kinect2.