16-662: Robot Autonomy Project
Bi-Manual Manipulation
Final Report

Oliver Krengel, Abdul Zafar, Chien Chih Ho, Rohit Murthy, Pengsheng Guo

1 Introduction

1.1 Goal

We aim to use the PR2 robot in the search-based-planning laboratory to help with unpacking groceries from a grocery bag. The setup involves having a table in front of the PR2 robot with a grocery bag on one side of the PR2 and a fridge on the other side. Our proposed activity involves identifying object inside the grocery bag, picking an object with one arm from the bag, transferring the object to the other hand and finally placing the object in the fridge.

1.2 Motivation

Human use two hands to perform many tasks, including open the can, pass the objects or lift the heavy stuff. Without a second hand, it's difficult for human to perform such tasks. To perform similar tasks on the robots, it's important to teach robots to use both hands smoothly. This increases the variety of tasks that robots can perform as well as increase the performance of the robot.

1.3 Hardware Platform

The PR2 has two 7-DOF arms, a mobile base and a spine for height adjustment. It also has 2 parallel-jaw type end-effectors which can be used to grasp objects. The robot has two embedded computers onboard which are equipped with ROS Groovy. The robot also has a Microsoft Kinect which is attached externally to enable better perception. It also has a couple of 2D LiDARs and forearm cameras which we will not be using.

1.4 Challenges

There are several challenges in performing the entire task due to the large number of modules that need to be working together. The toughest challenge is that of regrasping. This is because it requires feedback to compensate for possible errors in the initial motion planning as well as ensuring that we choose a grasp that can perform the place task easily. This requires online correction of offline grasps which could prove very complicated.

The other challenge which is typical of a system is the integration of the different modules. The state machine described in the next section needs to function correctly to ensure that the different modules work well together.

2 System Architecture

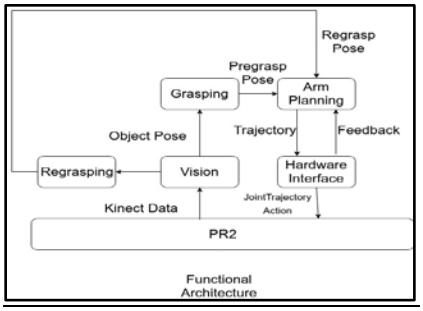

Fig. 1. Functional Architecture

2.1 Vision

Purpose:

- Provide object's pose (position + orientation) to facilitate arm planning.
- Identify scene for collision checking and high-level decision making.

Description:
Vision module will receive RGB image data and extract pre-stored AprilTag information to output the location and orientation of the object. It will also use the AprilTag information to determine the location of the scene object including table and refrigerator.

Code: Integrating different ROS packages and defining custom messages

2.2 Arm Planning

Purpose:

- Planning a collision-free trajectory to move the arm to desired location.

Description:
Arm planning module takes the desired pose (position and orientation) of the end effector and output the trajectory, which includes the desired joints' values and velocities at different timestamp.

Code: Planner and all other environment functions will be written and integrated using MoveIt! framework.

2.3 Grasp and Regrasp Planning

Purpose:

- Provide a collision-free and stable single grasp configuration to grasp a single object
- Provide a collision-free and stable dual grasp configuration to exchange an object between two hands

Description:
Grasping module will generate a set of valid grasps for each possible object offline. During single arm grasping, module will receive input including type of object and environment constraint (on ground or not) and output desired pre-grasp and final-grasp poses (position and orientation) relative to the object. During dual arm grasping, module will receive input including type of object and environment constraint in order to output desired pre-grasp and final-grasp poses (position and orientation) of the free arm.

Code: Using OpenRAVE for generating valid grasps inspired by first homework assignment.

2.4 Hardware Interface

Purpose:

- Execute planned trajectory and grasping on physical robot.

Description:
During planning, hardware interface will receive a trajectory includes the desired joint configurations at different timestamp. The interface will use low-level API to control the robot along the trajectory. During grasping, hardware interface will receive a pre-grasp pose and final-grasp pose. The interface will use low-level API to move the end-effector to final grasp position and close the finger. After all executions finished, the interface will also notify the high-level module to proceed to next step.

Code: C++ ROS node written from scratch

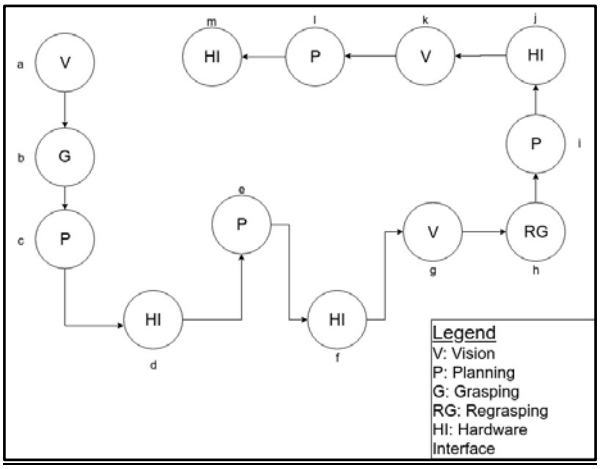

Fig. 2. State Transition Diagram

2.5 State Machine

The above figure shows the state machine that we will be implementing to step through the entire system process. It starts with the vision system (labelled as ‘a’) which detects an object using raw data from a Kinect. It then sends the object pose to the grasping module which identifies the best valid grasp and publishes this value to the arm planner. The planner identifies a trajectory of waypoints and passes this to the hardware interface which enables the robot to execute the movement. Then the arm planner plans to a predefined location to transfer the object to the other arm. The vision system helps identify the object pose for the other arm to regrasp the object. After the transfer is complete, the arm plans to the destination and places the object either on the table or in a fridge. There is also a higher-level decision that decides the destination of the object.

3 Subsystem Description

3.1 Arm planning

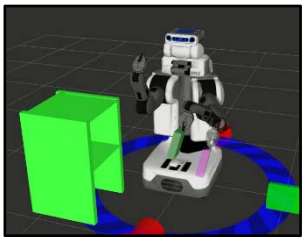

Fig 3. Planning in MoveIt!

3.1.1 Algorithm/ Implementation

For the purpose of generating feasible motion plans for the bi-manual manipulation of an object, the planning problem was decoupled into two

separate single arm planning problems instead of one combined dual arm planning. Consequently, the planning problem then boils down to generating motion plans for 7 DOF arm that avoids self-collision and collision with the objects in the environment while respecting the kinematic constraints of the robot. We utilized the OMPL library under the Moveit! framework to carry out planning. Specifically, we employed RRT-Connect algorithm for motion planning of a single arm coupled. For environment representation and collision checking, we used the APIs provided in the Moveit! framework such as '*move_group'* and '*move_group_interface'*.

3.1.2 Challenges

Although we did not face any insurmountable challenge in implementing the sub-system alone, however, one challenge that we did stumble on was running the planner on the PR2 network. Due to unsynchronized times and time zones between the planner-running machine and the PR2 CPU, the network was flooded with ROS errors pertaining to 'tf' package. Additionally, we also observed that the virtual state of the robot maintained by the 'move_group' seemed to be affected by the ROS PR2 msgs being published by PR2 robot. A work around regarding this issue involved planning and writing the planned trajectory to a file offline. This file was then parsed by the Hardware Interface module to execute it on the PR2 robot while being connected to the PR2 network.

3.1.3 Trade Study

Two approaches that we considered for carrying out the arm planning are search-based planning approach and sampling based planning approach. For implementing the search-based planning approach, SBPL library (Weighted-A*) was intended to be used for which a planning environment needed to be implemented from scratch along with implementation of informative heuristics (BFS-3D) to guide the arm to the goal configuration. Contrary to this, implementing sampling based planning approach involved less implementation time as the RRT-Connect planner implementation was already available as an OMPL plug-in in the Moveit! framework. Although search-based planning generates intuitively better motion plans for the arms as compared to sampling based planning, however, keeping in view the scope and requirement of the project, both approaches were viable and fulfilled the required functionality of the project. Hence, we decided on going with the sampling based approach due to faster implementation time.

3.2 Hardware Interface

3.2.1 Algorithm

The PR2 controllers use *actionlib* to receive requests for execution from the software in the form of *FollowJointTrajectoryAction* control messages. This is a very flexible interface which allows for a trajectory of points to be passed into the interface and the corresponding controller is able to generate a control strategy to follow the trajectory. To make this work we wrote a C++ ROS node that can accept a trajectory and pass these into the required controller. We have implemented this module and tested it on the PR2. An example demonstration can be found at this link.

3.2.2 Challenges

This module was the first one that we implemented thus we still weren't well acquainted with the PR2 at the time, when we started writing this module we were using the wrong *actionlib* to communicate with the robot. We were using *JointTrajectoryAction* instead of *FollowJointTrajectoryAction* which was resulting in the robot not responding to our commands. Once we fixed this we were able to overcome this challenge.

3.2.3 Trade Study

There were no alternative methods to implement this module. The interface with the robot is fairly well-defined and requires us to proceed in exactly this way. The reason for that is that the SBPL has existing nodes which listens to certain topics published in ROS Indigo and convert them to ROS Groovy. This helped us finalize the hardware interface early without much ambiguity on what method to choose.

3.3 Perception

3.3.1 Algorithm

For this component we are using an April Tag detection based method to get the object pose (position and orientation). The Microsoft Kinect that is attached to the PR2 gives raw image data which the *apriltag_ros* and *ar_track_alvar* package is able to interpret and calculate the pose of the Apriltag. We have implemented this package and are able to get object pose information in real-time from a number of objects.

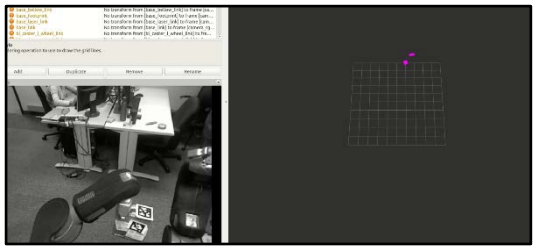

Fig 4. AprilTag Detection

3.3.2 Challenges

The first challenge is the synchronization of the camera info and camera images topics. Every second camera info topic publishes more than a hundred messages, but the camera image topic only publishes 2 messages. ROS cannot synchronize both topics so the perception system won't detect the Apriltag reliably. The second challenge is that the field of view of the robot is limited, so the robot might not be able to see the bottle if it doesn't move its head. In order to move PR2's head, we have to write the script to make it look at its grippers so that the Apriltag on the objects will appear in the field of view of the robot. The third challenge is that the transformation between the objects to the PR2 *base_link* is hard to get due to the time synchronization issue between the PR2 and the local machine. The Apriltag node running on the local computer can publish the transformation between the camera and the objects. The PR2 can know the transformation between the *camera* and the *base_link*. However, PR2 cannot give the transformation between the objects to the *base_link* because the timestamp between the PR2 is different from the timestamp of the local machine. We can only manually write the script to get the transformations between *base_link* and *objects* from both PR2 and local machine.

3.3.3 Trade Study

There are many Apriltag implementations online in ROS Indigo version, but none of them use ROS Groovy version, so it can only run on the local computer which causes the time synchronization problem. We used the Apirltag package from personal robotics lab at CMU. It provides Apriltags-cpp and Apriltag implementation to detect the 36h11 family tags. Even though it cannot synchronize the time perfectly, the system is still able to detect the Apriltag at a frequency of 1Hz.

3.4 Grasp and Regrasp Planning

3.4.1 Algorithm

The goal of grasp and regrasp planning is generating collision free and natural grasping pair for both arms relative to the object. In our demonstration, we use left arm to pick the object from ground and

right arm to regrasp and put the object in refrigerator. A set of valid grasps for every object are auto-generated from *OpenRave GraspingModel* package offline. For each of the object, there will be approximately 50 valid grasps available.

When grasp planning module starts, it will firstly filter out the list of single-arm grasps based on two criteria, grasp stability and prior knowledge. Grasp stability refers to grasp quality metrics and in our case, we use volume of ellipsoid in wrench space to measure it. Prior knowledge refers to grasping object's initial pose as well as regrasping requirement. For the left arm which picks object from ground, we will filter out all grasps coming from bottom. For the right arm which picks object at the exchange point, we will restrict grasp from side in order to avoid arm colliding with table.

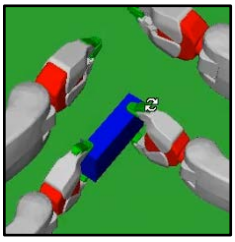

Fig. 5 Regrasp Example

After getting a set of filtered grasps for both arms, a double grasping list are generated by combining every two valid left and right grasps. We then test left and right arm's grasps together in order to get a collision-free double grasping. The pairs are also ranked by both stability and distance. Stability refers to the minimum stability score for a single arm calculated by grasp quality metrics. The pregrasp poses relative to the object are generated from the first-ranking pair from double grasping list.

The regrasp pose is determined based on the pregrasp poses. The right arm end effector pregrasp pose is predetermined and fixed in order to restrict it approach object from right direction. Based on the regrasp poses between left and right end effector, left end-effector pose will be calculated relative to right end effector. Thus, the pose for left end-effector in robot-base frame will be determined. The interchange location will be offset by 10 cm from left-effector's pregrasp pose along grasping direction.

3.4.2 Challenges

The unexpected challenge faced in double grasping module is unsynchronized time between robot on-board computer and our laptop. Due to this issue, even though our laptop is connected to ROS master launched in robot, TF information between robot internal links and pregrasp poses cannot be linked together. In order to solve this problem, we decided to subscribe to robot internal link and get *camera_link* to *base_link* transformation as tf_1, object to robot *camera_link* as tf_2, end effector to object transformation as tf_3. Then we manually calculate end effector transformation to *base_link* and publish the TF information.

3.4.3 Trade Study

The only trade study we have done is the method used to evaluate single grasping quality. However, all the several methods including volume of ellipsoid, minimum singular value of G, radius of largest centered sphere have generated similar results. The reason comes from the simplified object model as a cube and 2-linkage design of the gripper. We decided to use volume of ellipsoid as it provides descent results for all three objects.

4 Evaluation

Evaluation of individual subsystems in our project was mainly through simulation and objectively testing whether it performs the task that we are attempting. The arm planning module was simulated using fake controllers in MoveIt! and visualized in RViz. The grasping module was tested in the OpenRAVE simulation.

Testing out the entire system in simulation would have required a large overhead of setting up the environment and sensors in Gazebo. Thus we chose to test the entire system on the robot itself. This also allowed us to improve the pipeline based on the difficulties faced by interfacing with the robot itself.

The result of the evaluation was the successful implementation of the system on a particular object - Cheezits box. The video of the successful demonstration can be found here. While the video shows a successful execution, it required a few attempts to get it perfect because of minor changes in the environment affecting the performance.

We achieved the main goal that we set out to do as stated in our midterm report. We were hoping to successfully perform regrasping of an object allowing the robot to pass objects from one hand to another and were able to achieve this. This fits in well with the motivation for pursuing this project which was to give robots the ability to use both arms smoothly thus allowing robots to perform a wider set of actions.

The main limitation of the system currently is that the pipeline is not completely seamless. There is still some work to be done with the message passing from one node to another. The other limitation is the ability (or lack thereof) to perform high-level decision-making during the arm planning and grasping pipeline. This is crucial to allow replanning and online grasp calculations. It will also allow for grasp verification since there is a chance that the grasp is executed wrong which would result in an incorrect grip between the robot and object.

5 Conclusion

5.1 Work Division

This project included several modules that required extensive development work. Hence we were able to divide work based on three broad functions:

- Perception: Chien Chih Ho
- Grasping Module: Pengsheng Guo and Oliver Krengel
- Arm Planning and Hardware Interface: Rohit and Abdul

Integrating these modules together was made easy because of the modularity offered by ROS. By publishing relevant messages from each node, we were effectively able to integrate the different modules.

5.2 Unexpected Delays

We faced a peculiar difficulty in interfacing with the PR2. We expected this to be the least difficult portion of the project but ultimately caused us to change our system integration. The system time on the PR2 was slightly different from our system time which was causing TF to give errors that flooded the system resulting in many dropped messages. TF is integral to converting the grasp pose from the object frame to the robot base frame.

This problem was also the reason that the planned path being generated from the planner was not being subscribed by the hardware interface node. As a result, we had to find a workaround by parsing the trajectory into a file and then reading from it while sending to the PR2.

5.3 Challenges

The first major unexpected challenge was before the midterm report due to a faulty power terminal which resulted in it being torn out. This set us back around a week till we were able to order a new part and replace said power terminal.

Another major challenge was the hardware interface. While we figured out the correct way to use *actionlib* later on with the PR2, it posed a big challenge for us in the initial parts of the project. After learning the correct messages to interact with the PR2, this part was a lot smoother.

5.4 Lessons Learned

An important lesson that we learned is that it is crucial to fully understand and explore a robotic system before using it. This will help mitigate the main risk of unknown delays to the project, especially near the chaotic end of the project.

We had initially planned on performing the regrasping online and our testing with the limited system shows that this was the right idea. Small mistakes in gripping the object could change the orientation of the object which would result in the regrasping pose being wrong. Thus we should make small changes in the regrasp pose based on the final position of object.

5.5 Possible Improvements

If we had to do the project again, we would attempt to test our different interfaces with the robot first before proceeding the rest of the development. This is because in order to manage the risks effectively, the main risks should be mitigated as early as possible.

We would also have liked to have better modeled the environment early on in the project to make development work a lot easier. A lot of the arm planning code needed to be redone because of changes in the environment.

Appendix

The final compiled video can be found at this link.