SMOOTHIEBOT: Using Fetch for Gourmet Purposes

Aryeh Zapinsky, Geffen Huberman, Leyla Mizrahi
aryeh.zapinsky@columbia.edu <ayz2103>, gh2434@columbia.edu, om2243@barnard.edu

Abstract:

In this paper, we introduce SmoothieBot, a Fetch robot that is meant to simulate the simplest kitchen tasks (grasping and moving objects to a blender) upon receiving verbal commands from a user. Our approach is focused on setting up a robust, reliable environment for robot navigation using the ROS framework and relevant ROS packages: Gazebo, MoveIt!, Tele-op, fetch_ros, Columbia University's GraspIt!.

Introduction:

The kitchen as a playground for robotics presents many computational hurdles. Perceiving and discriminating objects in a cluttered workspace, safely manipulating kitchen objects, and navigating a small environment are all separate tasks that present unique challenges.

Our goal in this project is to set up a simplified kitchen world for a Fetch robot. SmoothieBot, a smoothie making robot, would receive verbal commands specifying fruits to put into a blender to make a smoothie. It would then navigate to the table, pick up the specified fruit, move it to the blender, and start the smoothie making process. This a voice controlled pick and place task.

After we successfully navigate SmoothieBot with the pick and place task, we would enable SmoothieBot learn the fruit composition different types of smoothies. SmoothieBot will learn the composition of sweet and savory smoothies, as well as user preferences over time. SmoothieBot would then be able to act on requests for a "sweet" smoothie, or the "Tuesday morning" smoothie, and respond to verbal commands requesting either type.

Related Previous Work:

Extensive work has already been done in the field of grasping and moving.[1] Kehoe, et al. presented an architecture to use cloud based object recognition for robot grasping.[2] Kehoe et al. used a PR2 robot. They sent images of objects to a cloud based Google object recognition engine.

[1] Andrew T Miller and Peter K Allen. "Graspit! a versatile simulator for robotic grasping". In: IEEE Robotics & Automation Magazine 11.4 (2004), pp. 110–122
[2] Kehoe, Ben, et al. "Cloud-Based Robot Grasping with the Google Object Recognition Engine." *2013 IEEE International Conference on Robotics and Automation*, 2013, doi:10.1109/icra.2013.6631180.

They used the result of the object recognition to determine what object was in view of the PR2. Kehoe et al. queried a database of object models and determined candidate grasps based on the object models.

Bollini and his team laid out an understanding of the kitchen as a semi-structured proving ground for robotics.[3] It inspired the presentation of different kitchen tasks (cutting, chopping, etc.) as a state diagram for a PR2 to follow. It also demonstrated the importance of integrating natural language processing--in this case, parsing recipes--into software for humanoid robots to emphasize their ability to interact 'naturally.'

Both of these works, although insightful and good starting points, proved insufficient for our problem. Firstly, there was an abundance of old code that was used with the PR2. The robots lack proper maintenance and are no longer supported for development. Secondly, none of the papers provided working codebases. As a result, when mapping their implementations to our problem, we lacked the resources to understand where specific issues were coming from. Therefore, although these papers provided general design frameworks, we had to change our implementation and start from the basics again.

Method:

First we will create a gazebo simulation with a Fetch robot in the simulated environment. We will download meshes of fruit from TurboSquid and 3D warehouse to have a blender and fruit in our simulated world. We will add these meshes to a world manager server to be able to move the Fetch's gripper to the proper pose. SmoothieBot will wait for a command served through Amazon alexa to determine which fruit to attempt to grasp. We will use point cloud segmentation and iterative closest point algorithms to discern the proper object in the scene. SmoothieBot will use the meshes from the point cloud segmentation in GraspIt! to determine possible grasps on the meshes. We will attempt to grasp the object with these grasps. After a successful grasp, SmoothieBot will move its hand to the blender and place the fruit from the hand into the blender. After placing all the desired fruit into the blender, SmoothieBot will use a similar process to cover the blender.

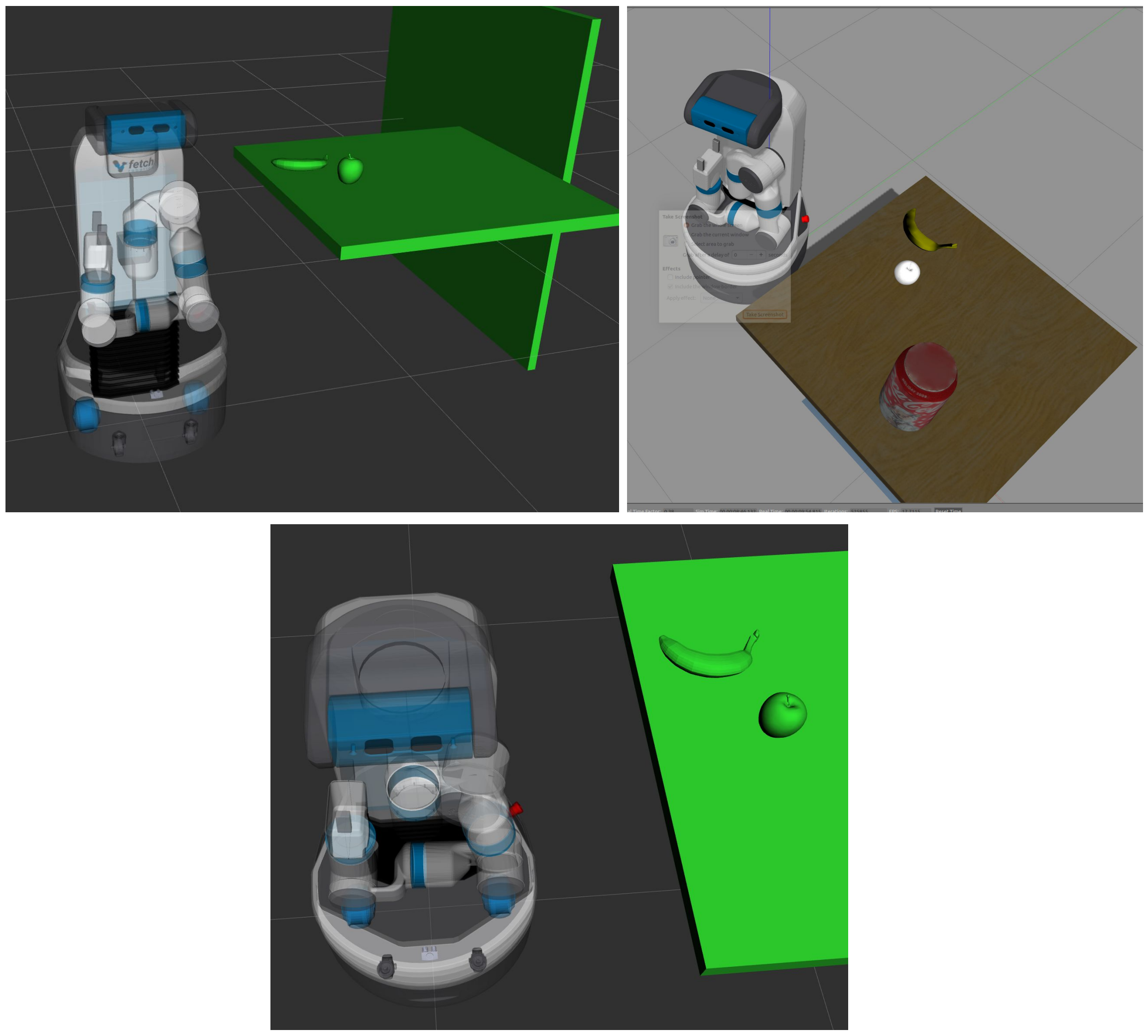
Progress and Procedures:

1. Robot Simulation with Gazebo

We create a Gazebo simulation with a Fetch robot, a cafe table, and several custom fruit models including an apple and a banana. In order to create our simulation world with fruit we had to go through several steps. First, we downloaded meshes from TurboSquid and 3D Warehouse. Initially, when we placed these object meshes into the simulated environment, they were far too large. We used MeshLab to scale our fruit to reasonable sizes. Second, in order to import custom meshes into Gazebo, we had to copy the Collada files into the local Gazebo models directory and add two more files, a config and an sdf (spatial data file), to register the objects. Third, once we were

[3] Bollini, M, Tellex, S, Thompson, T, Roy, N, Rus, D (2012) Interpreting and executing recipes with a cooking robot.

able to import the custom meshes into Gazebo, we placed the objects in simulation and saved our simulated world with custom models.



Displaying Fetch and fruit meshes in Gazebo and RViz

2. Voice Control with Alexa

This approach was adapted from CRLab's alexa_ros_controller-master repository. We used Amazon Developer Portal to set up an Alexa JSON intent scheme. The specified json is sent to a ROS network through an ngrok server and reaches a local port which interacts with the ROSBridge Websocket server. ROSBridge then creates an ROSTopic that receives this json. Finally, we would ideally integrate this ROSTopic into our grasping code (which we did not get the chance to do). This would determine which fruit is grasped in the scene.

3. Visualizing

We use RViz in order to visualize the simulated Fetch robot. We faced several issues when trying to implement the visualization. We used CRLab's world_manager package in order to manage meshes and transposes in RViz.

Initially, we placed the fruit meshes in the Fetch's base_link frame. This caused the meshes to translate and rotate when we moved the Fetch in Gazebo. We tried adding a planning map and adding the fruit to that frame, but this did not solve the problem of rotating fruit. We tried adding the fruit to a world frame and tying the world frame to a fixed point in the map, which again proved unsuccessful. Finally, we tried computing the transform from the base_link to the fruit and updating the transform as the robot moved, but again this did not solve our problem. We then realized that although the meshes rotated in visualization, the pose transforms did not. Consequently, we solved our problem by only adding the meshes to visualization after rotating the Fetch to be in front of the table.

4. Motion Planning and Grasp Planning

We successfully position the fruit in the visualization at hardcoded poses.

Initially we tried to use CRLab's CURPP repository, which provides a wrapper for MoveIt!'s pick and place pipeline. This necessitates planning a grasp with graspit, transforming it into a MoveIt! Grasp, analyzing whether it is reachable, and then attempting to execute that grasp. We verified that SmoothieBot was facing the table and was close enough to the fruits to grasp them. We were even able to manually place SmoothieBot's arm to be directly above the fruit, as if about to grasp it. However, when analyzing the grasp reachability based on our planned grasp SmoothieBot determined that every grasp, regardless of the object's pose, was unreachable.

Next, we tried to use MoveIt! path planning directly, but planning from a tuck position to the table position proved too challenging for the Fetch. We set the arm in a raised position to be able to come down and grasp the fruit.

Currently, we use the ground truth positions of the objects in the scene. The robot follows a predefined sequence of poses specific to each object in the scene. In the next steps, we would replace the ground truth positions with positions returned by point cloud segmentation and object recognition.

We are using a simple heuristic grasp on the object to determine that we are in the proper location. In the future, we would pass a mesh to Columbia University's GraspIt! to find possible grasps on the fruit.

Conclusions:

Future Work

In the future, we still envision SmoothieBot coming to life in domestic environments. We are still enthusiastic about implementing a classifier that would distinguish among different smoothie types based on different fruits and different times of day. We did not have time to implement these features because we spent the majority of our time attempting to resolve issues with grasping and ROS. Furthermore, our ideal final product would have a more sophisticated, more flexible implementation of grasping with MoveIt! And GraspIt!.

Lessons Learned

Systems/Infrastructure are hard: Setting up and acclimating to the ROS environment has a steep learning curve. The different dependencies required for packages ended up delaying us on a number of occasions. We did not have similar workspaces and environments. This led to use of better tools such as gitman, a package manager, and and node manager, a debugging tool for visualizing ROS nodes, to synchronize our workspace.

Use currently supported tools as often as possible: We also had to shift our configurations from PR2 to Fetch after discovering that the PR2 robot was down. We attempted to use many outdated packages and methods. It is important to use current tools and current solutions to not face the same issues that people have previously faced and not solved. If there are deprecated packages that solve the problem, first check if there are other solutions, and if all else fails, then try to debug and fix unsupported and deprecated code.

Division of Labor

We divided the work based on three tiers - Tier 1: Robot world simulation and grasping, Tier 2: Verbal commands, Tier 3: Machine learning aspect for smoothie preference.

Aryeh worked on grasping and robot world simulation in Tier 1.

Geffen worked on Tier 2 and grasping.

Leyla was meant to do Tier 3 but we did not get to its implementation.