Lab 8: Jinx & Merry Integration

Connor Wolfe, cbw36@case.edu; Xiaoye Qian, xxq82@case.edu Sasha Belotserkovskaya, avb27@case.edu; Kristina Collins, kvc2@case.edu

I. Introduction

This assignment integrates the motion control implemented in Lab 7 with the object finder code from PS9 and the object grabber code available in the learning_ros code repository. It demonstrates the integration of complex systems to perform a task: Jinx carries Merry to the stool, where a red block is identified with input from Merry's Kinect. Merry picks the block up, and Jinx returns to the starting position. All major systems are used: Both the Atlas and Jinx's computer run requisite ROS nodes, based on sensor input from Jinx's odometry and lidar and Merry's Kinect, and actuate Jinx's base and Merry's arms and gripper.

II. Theory of Operation

A. Coordinator

The coordinator node handles the high level steps of the lab, of which there are four:

1) Jinx drives a programmed path from the starting square, marked on the floor with tape, to the location of the stool. Linear steering with odometry feedback is used, but AMCL isn't needed for a distance this short. This is handled by the **mobot_pub_des_state** nodes.
2) Merry identifies the block's position with Kinect input.
3) Merry moves her gripper above the block's major axis and picks up the block.
4) Jinx navigates back to the starting square, which necessitates moving backwards so that the lidar unit doesn't get caught in the stool.

There are two relevant coordinator nodes, the command bundler and the primary client. The latter monitors the robot at the highest level and calls commands from the former, which itself calls the navigation and vision nodes.

B. Navigation

The navigation is an extension of the system developed in PS7, where Jinx runs off the Mobot simulation nodes. A specific path client, **acquire_block_client_final**, requests the path. For backwards navigation, we added a new function in **traj_builder** with a backwards trajectory, using negative velocity and acceleration. We created a new function in **mobot_pub_des_state** which called this trajectory. The client would call **append_path_queue** to move forward, and the new function to back up.

C. Object Finder

The **object_finder** node from Part 5 of the learning_ros repository can be used with few edits.

D. Object Grabber

We followed the edits in the Canvas announcement to control the gripper. First, we added **include <object_grabber/object_grabberAction.h>** to the coordinator node, then initiated the publisher: **ros::Publisher gripper = n.advertise<std_msgs::Bool>("close_gripper", 1);** We initialized a Boolean variable to indicate whether the gripper was open or closed; in order to close the gripper, we published to the topic for 3 seconds.

III. Testing

Baxter is set as the ROS master.

- On Jinx, turn on the thermal fuse by pushing the black tab up.
- Wait 10 seconds for the pre-charge of the capacitors on the output stage of the Roboteq.
- Hold down the on switch until the LED ring and battery voltage monitor turn on. (If the voltage is under 24V, charge the robot for a while.)
- Check the ESTOPs.
- Turn on the computer and log in: user/1lmiagmc!
- Turn on Merry using the switch at the base; wait for the green halo to turn on.
- Run **baxter_master** on Jinx. This script must be run in each terminal window.
- Run **roslaunch launchers start_jinx.launch** on Jinx.
- In another terminal, run **baxter_master** and **roslaunch launchers start_baxter.launch**.
- In another terminal, run **baxter_master** and **rosrun rviz rviz**. Grasping the capacitive switch depressions on Merry's gripper arm, move it to the Kinect's viewable area, and check whether the transforms are correct in RVIZ, based on whether Merry's arm is "painted" with points. If not, adjust them in the Baxter launch file.
- In another terminal, run **baxter_master** and **roslaunch coordinator_final coord_vision_manip.launch**.

It's a good idea to run **rostopic list** and make sure the robot is connected, particularly the ESTOP topic. Make sure to place Merry in the tuck position and shut off Jinx after use.

IV. Testing & Results

Our code may be accessed on Github from the following address: **https://github.com/qianxiaoye/teambacon**.

We were able to demonstrate the navigation and block finding/grabbing operations independently. We had some difficulty running them together, partly because of merge conflicts with other groups' code in the shared repository.

The video of successful navigation can be found at this address: **https://www.youtube.com/watch?v=e8XpqY9KqjI**.

The video of successful object identification and grabbing may be accessed at the following link, and is much funnier: **https://www.youtube.com/watch?v=TVLlgKWXJAI**.