Group Zeta
lab_9

writeup

This lab is the final lab of the semester, and is the culmination of our work.
Our goal is to have a baxter robot (Merry) and a custom navigation robot (Jinx) travel together (Merry is bolted to the top of Jinx) from a starting location, over to a stool (at a well known location in the room), identify a block sitting on the stool, pick up the block, and bring the block back to the starting location.

We subdivided this problem into three subsystems:

1) Navigation
2) Object Detection
3) Object Manipulation

To control the robot, we have a "client" program that will command the combined robots to perform the following steps in sequence:

1) Navigate to the stool and stop (from the starting location)
2) Find the block on the stool
3) Pick up the block on the stool
4) Back up (so we don't knock over the stool when turning to go home)
5) Navigate back to the starting location and stop (from the stool)

Our client is found in this repository as lab_9/lab_9_path_client and the source file is found at src/lab_9_path_client.cc

This repository also provides several launch files required to bring up Merry and Jinx.
They are as follows:

1) launch_on_merry.launch -> this file launches all nodes that are required for the Object Detection and Object Manipulation subsystems to work. It (as it is named), should be run on Merry.

2) launch_nodes.launch -> this file launches all nodes that are required for the Navigation subsystems. It can be ran on Merry or an arbitrary machine that has Merry as the ROS master. Note that these nodes need to run in real time, so a strong connection is suggested if not running directly on Merry.

We had to make several design choices to get the Object Detection and Object Manipulation subsystems to
work together to correctly pick up the block. Since Merry is equipped with a ReThink gripper, the orientation
of the gripper fingers matters (the gripper cannot pick up the block by its minor axis (short side)).
The Object Manipulation subsystem defaults to aligning the gripper fingers with the minor axis of the block,
and the Object Detection subsystem will return the major axis of the block if it finds the block.

To correct for this, we had the Object Detection subsystem return the minor axis of the objects that it finds
as opposed to the major axis, so the Object Manipulation subsystem would now align the gripper fingers with
the major axis of the block and pick it up.

We also had to incorporate a new gripper interface into the Object Manipulation subsystem, which we placed
in the "zeta_object_grabber" package.

Finally, we had to create the functionality for the Navigation subsystem to allow it to back up.
This was not a trivial task, as the Navigation subsystem makes two crucial assumptions:

1) The robot wants to always move forward.
2) Navigation always knows where the robot is (and will not move by other control other than Navigation).

By the second assumption, we cannot "open loop" the robot backwards without turning off Navigation. This
would be possible except for that we want to only run the client once, and would therefore be required to relaunch the Navigation subsystem from within a running client; something we weren't keen on doing.

Therefore, we decided that Navigation would provide an additional "special" service that would only drive the robot from its current point to a provided point. Note that it is completely possible to have the robot drive a path backwards using this architecture, but separate service calls would be required for
each waypoint in the path (as opposed to giving a full waypoint path to forward navigation).

With this functionality, we were able to get Merry and Jinx to successfully complete its goal.

Some problems that we encountered during development (and our competition run):

1) Sometimes inverse kinematics of the right arm of Merry fails. This can happen at arbitrary times during
movement, sometimes even if there is a valid path to follow.

The most common of these errors is that once Merry places the gripper over the block, she will be unable
to find a path to raise her arm and move her hand to her side. This is remarkable since that is where her
arm started, so if she was able to move her hand over the block, she had to have found and executed a valid path from the now goal location.

The second most common error was Merry would find a path to execute, and then be unable to execute it mid-way.

2) Sometimes the gripper does not close (or open) when we tell it to. Commands that we give seem to either
run when we give them, or run at an undetermined later point in time.

3) Sometimes the Object Detection subsystem does not find the block on the stool. This is likely due to the
stool not being a horizontal surface (the stool is tilted). Therefore, when looking in a small rectangle in space, the block might be mistaken for part of the table, or vice versa due to the orientation of the stool and the pose of the block on the stool.

This is the commands we ran for our demo:

1) (On Merry) baxter_master && roslaunch launchers start_baxter.launch
2) (On Merry) baxter_master && roslaunch launchers start_jinx.launch
3) (On Merry) baxter_master && roslaunch lab_9 launch_on_merry.launch
4) (We ran it on Merry but you don't have to) baxter_master && roslaunch lab_9 launch_nodes.launch
5) (We ran it on Merry but you don't have to) baxter_master && rosrun lab_9 lab_9_path_client

Our code is located at:
https://github.com/aew61/eecs376