Tidying with Robots, a.k.a. OCD-bot

Nick Alessi, Shyam Lenna

Abstract

Our plan for the robot is to have the PR2 start in an office space, determine if it is organized or not, and organize it if necessary. Specifically, the goal is to have it primarily push in chairs under desks aligned with monitors. It will push in chairs by navigating behind them, moving arms to a pushing position, and driving forward to the goal.

Introduction

If you had a robot in your office or home, what would you want it to do? Traditionally, people think of menial chores such as cooking, cleaning, or fetching objects. Fetching objects has been done plenty of times before, so we decided to address the former. We decided to try having a PR2 organize a room. Imagine leaving your office for lunch, letting the PR2 in as you leave, and coming back to see the room all neat and tidy. We want to continue the dream of having robots make our living space nicer, by addressing the task of pushing in chairs.

Approach

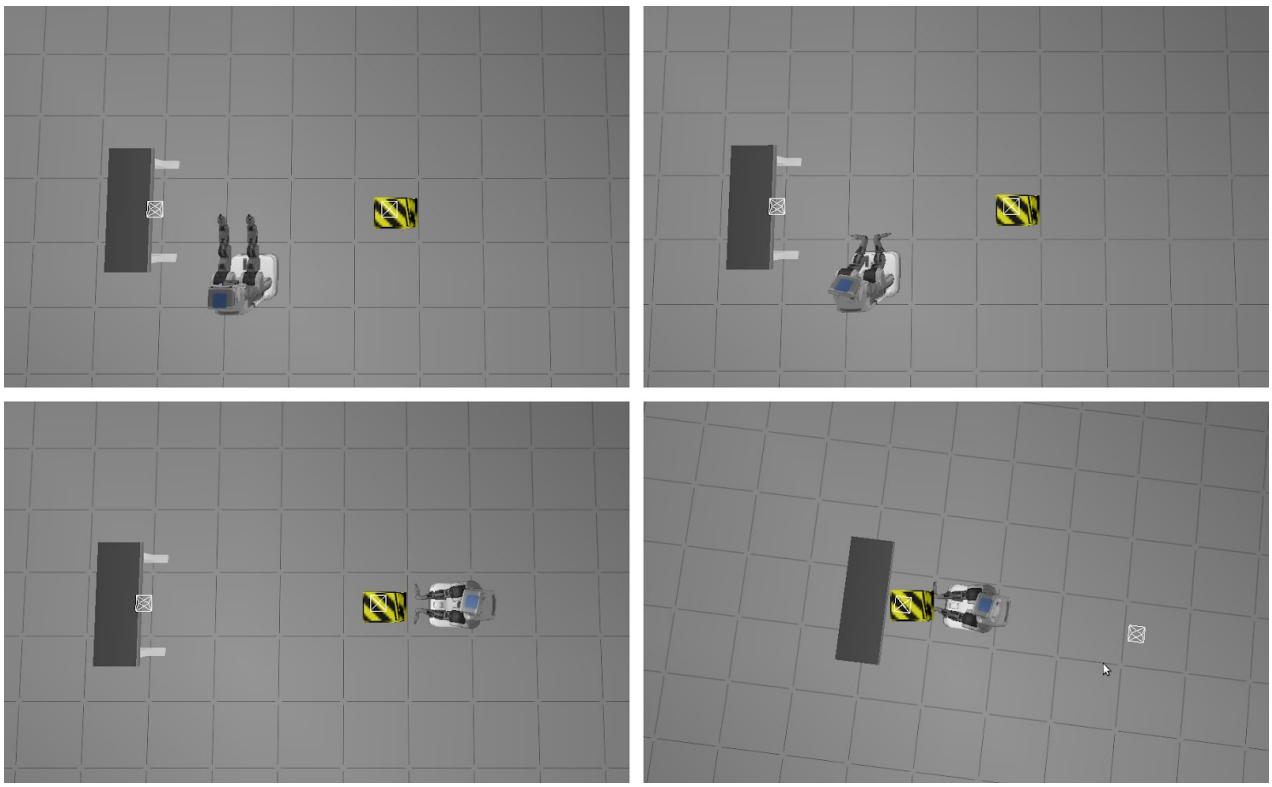
Our project is split into two primary phases - detecting chairs and tables, and moving chairs. We initially hoped to have the robot start with moving about the room to find objects, but without a kinect attached, we restricted it to the case for when the initial view contains both a chair and a table, and it will just push in one chair. Our code is divided into two ros nodes which are designed to act as independently as possible, interacting only through ros topics.

The first node perceives the environment to find both the chair and where it should be so that the environment is “neater”. We implemented a ros node that listens for labeled point clouds from the semantic label node and publishes the location of one of the chairs it detects and where it should be placed. We isolate an individual chair by filtering a copy of the point cloud to only include points of one of the chair types. Then we run a euclidean clustering algorithm from pcl that creates a cluster only if enough points are in the chair (to prevent erroneous labels from causing the robot to try and do something crazy).

We then choose one of the clusters and assume that it corresponds to an individual chair. Then the centroid of these points is found and used as the chair location. We then used that to find the nearest location to the chair that is a good positioning. For now this is just the closest table point, but ideally we would like to find a better method for this. Both of these points are then published for other nodes to use.

Our second node handles controlling the robot by reading these published points. For simulation, we have the robot move its arms to a pushing position - at about chair base height.

In real PR2 experiments this caused hallucinated obstacles to appear in front of the robot. So we changed this to tucking in its arms before moving; which solved the problem nicely. It then uses the default navigation stack to move to the correct place - near the chair, and facing the goal. Then it just drives straight until the goal. This ignores chair orientation. There are a few sample pictures below, and we also have a video up at http://youtu.be/lDEstURjObk.



PR2 pushing a chair in simulation.

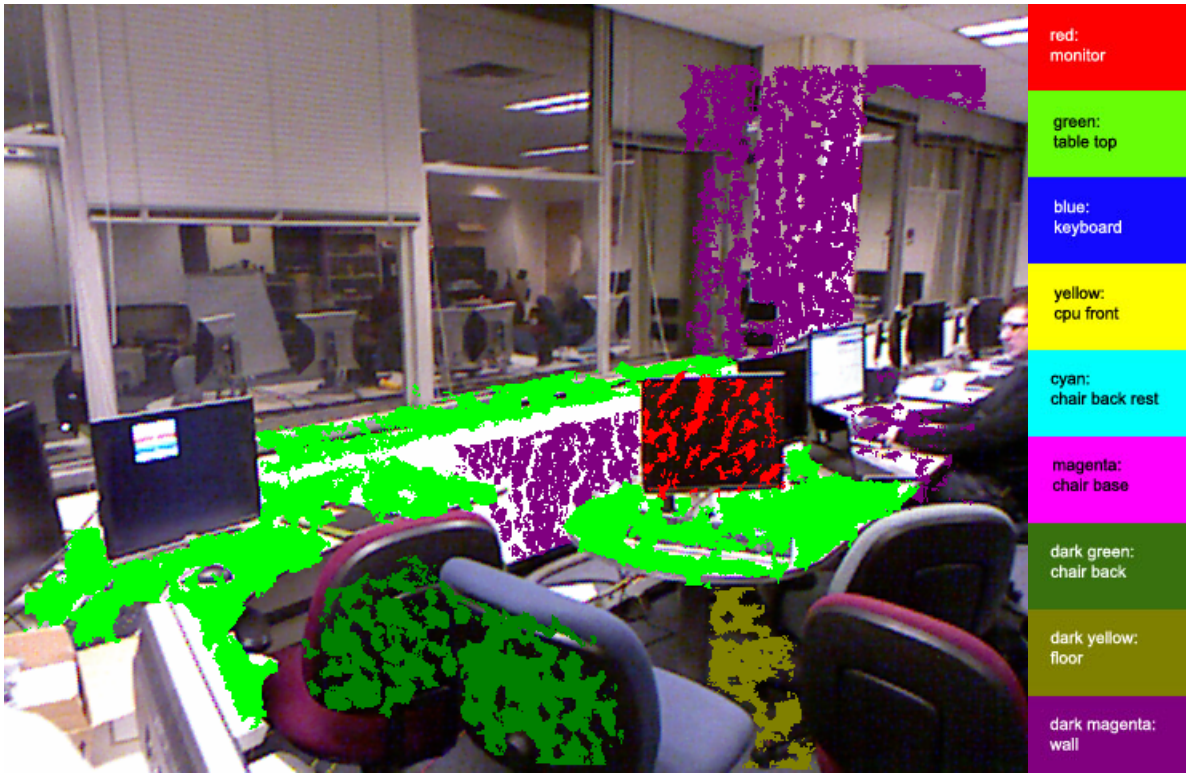
Future Work

If we were to continue this project, the first step would be to connect a kinect to the PR2. With this, we could modify the chair finding to be more robust. It could potentially have the PR2 navigate about, find multiple chairs and tables and push in all of them, instead of just a single one. Lastly, some work could be brought into designing a process to decide what order to push the chairs, and to what tables.

Another important direction we would look at is adding more intelligence to the method of finding a chair to push in. We would first like to add functionality to choose a chair based on how poorly placed it is at the time. We also only have a heuristic for neatness; given enough time we would like to learn a good model that would be more effective. This would also ideally take into account the other things that could be detected (e.g. computer towers, keyboards, etc), but it would have to be robust to the possibility that some of these may not be detected in the labeled scenes.

Experiments

First, we evaluated the effectiveness of the provided scene labeling code. To do so we setup a kinect sensor at a height closer to that of an actual robot (about 5 and ½ feet by my best guess). We then moved chairs, monitors, and keyboards into the shot to be included in the data, and moved them around between some point clouds.

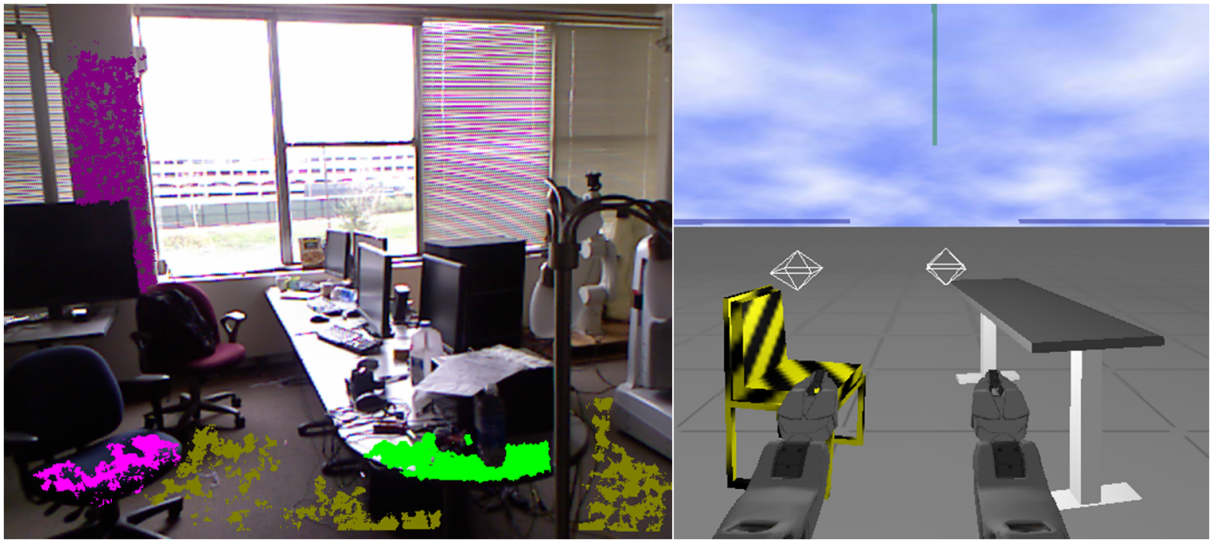


An example of the data we collected in the lab.

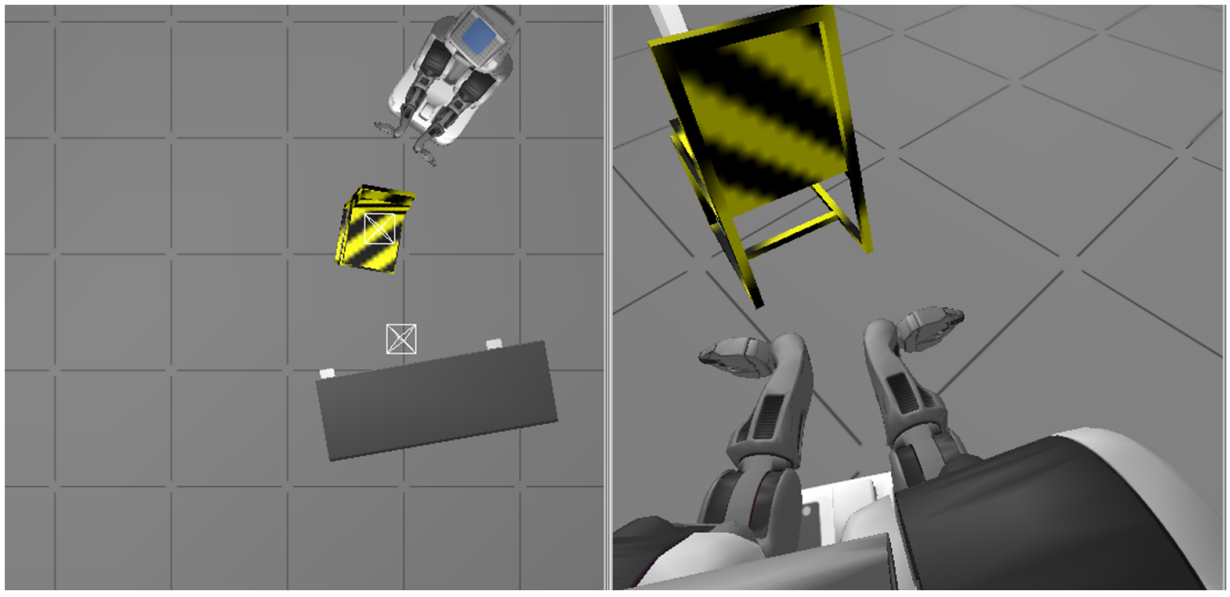
Evaluating the pictures (colored to represent labels) we observed that the tabletop was correctly classified 94% of the time. Because our setup did not allow us to bring the monitors closer so we got poorer results of 12% correct. The chair recognition involves three different components: the chair base, chair backrest, and chair back. The chair back is the worst classified only being correct 23% of all cases. The chair base is better recognized at 71% correct. Finally, the chair backrest is the best classified being correct in 82% of all cases. Keyboards (at least in setups we could create given our environment) were never classified correctly, but I believe we could spot them if the sensor were better positioned.

We tested the pushing of chairs in simulation with chair and table locations manually inputted. We tried many different angles and distances to ensure accuracy and correctness. After fixing a couple mistakes with angles, the robot performed fairly well in simulation, succeeding approximately 90% of the time when the goal was within 3 meters. Further than that and it would tend to get slightly off, but most rooms would not have the chair and table that far apart, so we considered this to be reasonable.

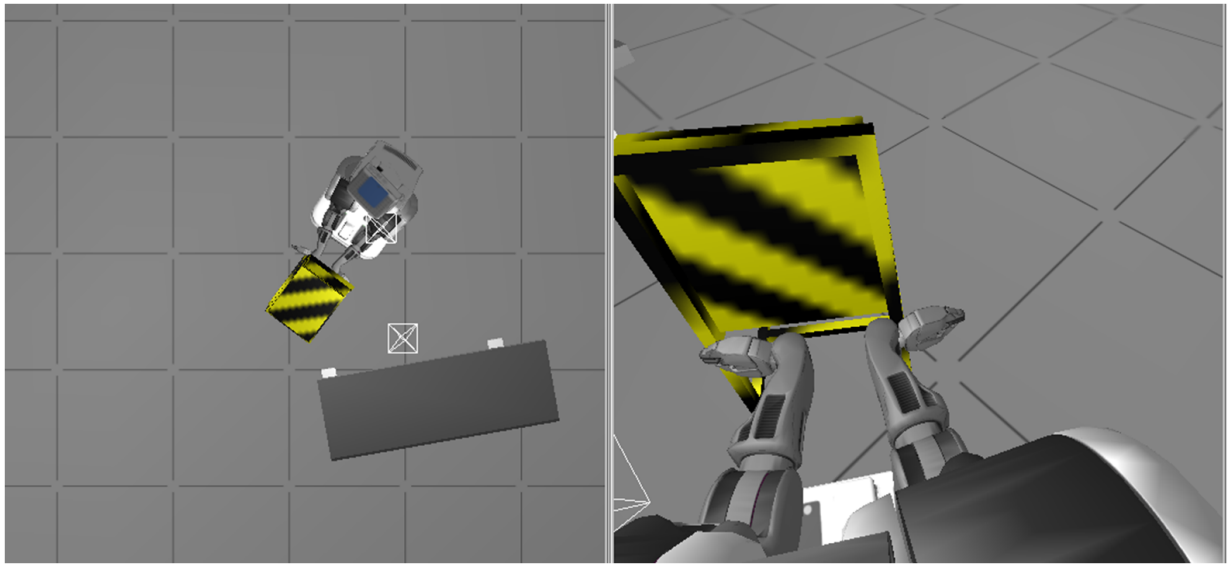
Then, we created a trans-realital simulation. We constructed a real world and simulated world to mimic each other. Without having coordinates manually inputted, we used a kinect and semantic labelling to find the locations of a chair and table, and then the PR2 in simulation pushed in the simulated chair. The pictures are below. Note that the simulated table isn't as long as the real table, and we didn't detect as much of the table as expected, so it seems slightly off, but the results are very acceptable.



See that only the edge of the real table is correctly marked green, as we only detected that part.



After navigation finished in the simulation.



We correctly moved the simulated chair to the edge of the real table.

Finally, we ran the moving code on the actual PR2. Due to time constraints and lack of having a kinect on the PR2, this was only run with chair and goal coordinates manually input. We found that the default navigation stack would frequently be the main cause of failure in these experiments. Most of the time, it would not approach the chair even though the path was very clear, in order to gain more info about its surroundings. When this happened we stopped it for safety reasons, because it would wonder in the opposite direction very close to obstacles. We believe that the PR2 would eventually reach the correct location if it were in a more open environment.

When it did get past this stage, it almost never failed at the chair-pushing stage. Overall, the accuracy was success about 20% of the time. Here are some pictures of the overall process, and we also have a video up at http://www.youtube.com/watch?v=2o2eAQU5irs.

Tucked in arms in order to navigate.

Finishing up navigation using the built in navigation stack.

The chair pushing position for the arms.

Driving forward to push the chair to the goal.

Acknowledgements

- Scene Understanding by Hema Swetha Koppula, Abhishek Anand, Thorsten Joachims, and Ashutosh Saxena
- ROS by Willow Garage
- Multiple ROS tutorials by Willow Garage