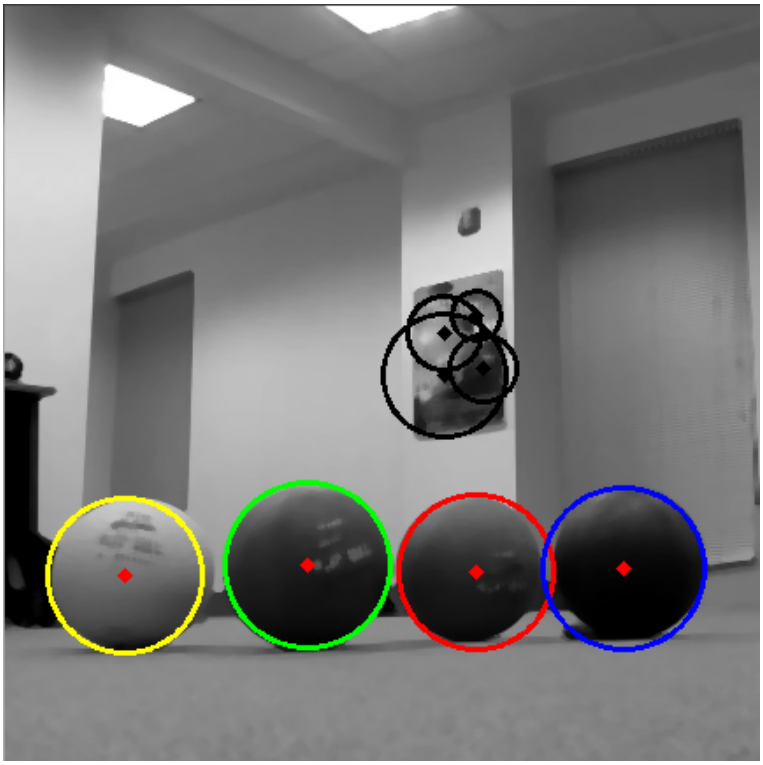
Computational Robotics : ERRbot
Emergency Rescue Robot

Written by

Amanda Sutherland, Claire Diehl, and Jasper Chen



December 18, 2014
Olin College of Engineering

Contents

List of Figures

1 Project Goal

The goal of this final project was to use many of the skills that we learned this semester such as robotic navigation and computer vision to create a search and rescue robot. This was in the form of a simple competition between several groups using the Neatos and their sensors. To achieve our goal, the robot must identify objects in an unknown environment and return their locations. The objects are a set of four colored balls as seen in figure 1.

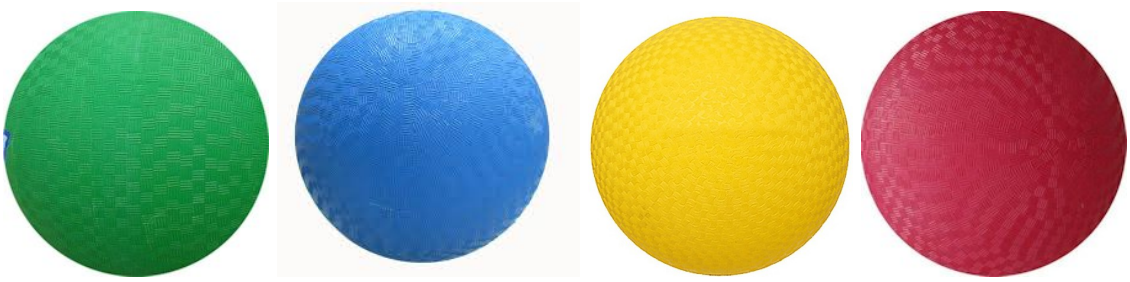


Figure 1: The ball colors.

2 Method - Problem Solving

To create the ERRbot we divided the problem into two smaller ones. The first was navigation. Moving around in an unknown environment is relatively difficult and a problem that, though it has been done, has much room for improvement. The second was the object detection using computer vision. Though other groups had done this in the past, all our computer vision had to be written and so this was far from a solved problem. These two issues came together in the map. The map depends on the LIDAR data and requires the robot to move around most of the space, but also requires the locations of the objects so as to place them in their respective locations. This was achieved using RViz and hector_slam.

3 Code Structure

The major code structure was split into map making, path planning, and vision, as in figure 2. Each of these components was a script that we ran using a launch file. They communicate by publishing and subscribing a set of messages.

The code begins with the LIDAR and camera data coming in from the robot to the appropriate topics. At this point the computer vision subscribes to the image and finds the balls. Path planning code runs using the LIDAR data to create potential paths. These two portions of the code do not interact, though in a more advanced version they might. In addition to the path planning, hector_slam uses the LIDAR data to create a map of the area and then plot the balls on this map.

Our github repo can be found here:
https://github.com/AmandaSutherland/ERRbot

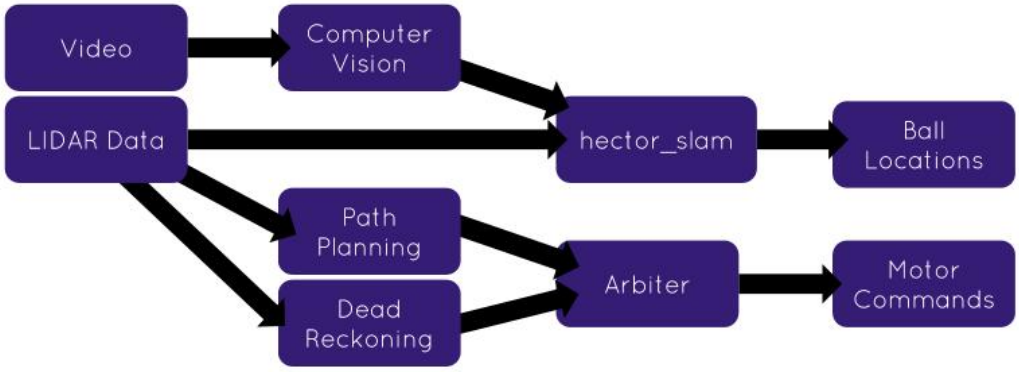


Figure 2: System Structure

3.1 Map Making

To create the maps, we continued to use hector_slam. By creating the map in RViz and adding the balls here, we did not need to create our own map making code and rather only create the connections.

One of the main issues with hector_slam is its tendency to lose its map frame when the robot moves quickly. As it was entirely unusable at the speeds the packages we used involved, we needed to account for this issue. Initially we attempted to dig into hector_slam and determine where the problem stemmed from in there, but changing hector_slam was both too complicated and apparently not useful to change. The main reason the robots cannot have sudden movements is that the LIDARs have too low of an rpm to not skip measurements when spinning. Therefore, we recognized we could do little to improve hector_slam and therefore concentrated our efforts in our path planning.
A map example is seen in figure 3.

3.2 Path Planning

Path planning is a combination of dead reckoning and other path planning packages. These packages work fairly well on their own to navigate to individual goals, though several additions needed to be made.

The largest and most time intensive change was to reduce the speed of the robot that was being output. By lowering the robot's velocity, especially its angular velocity, we could prevent the robot's position from being lost. This task was particularly time intensive as the parameters for speed change in packages we were dealing with were extremely well hidden and often did absolutely nothing to change the speed.

After this change the path planning worked well. However to actually use these packages, waypoints needed to be created for the robot to aim to move towards. For these waypoints we simply created a random set of x, y coordinates and reset these each time the robot reaches a waypoint or gets stuck in one location for too long.

The planned paths appear on the map as red lines as seen in figure 3.

3.3 Vision

Using the team who completed Neato fetch on the last project as inspiration, we created a computer vision script that uses a combination of Hough transform circle detection and color masks to identify the balls and decide what color the ball are as well as decide how far away from the Neato the balls are located.

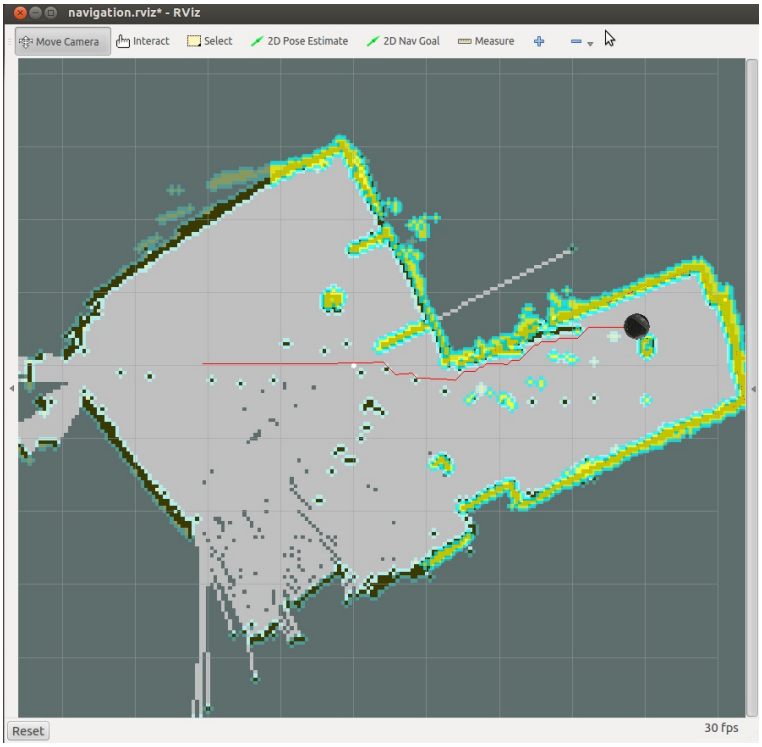


Figure 3: Mapping and Path Planning

The Hough transform locates balls and compares the circles to the four color masks. When there is a circle of the appropriate color the distance and angle of the ball in reference to the Neato based on the size and location of the circle in the image, as seen in figure 4.

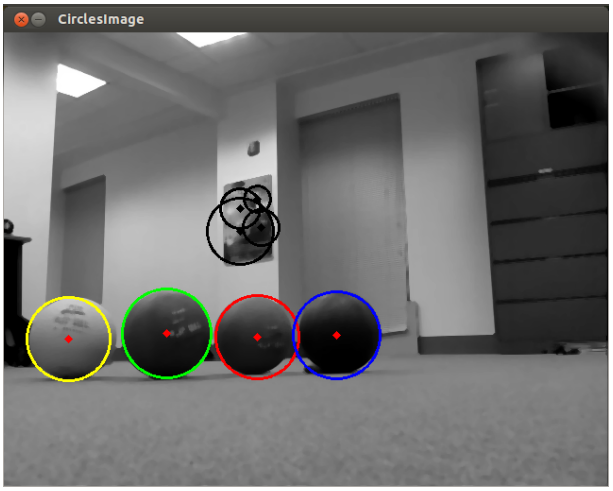


Figure 4: Ball detection using Hough transform circle detection and color masking.

3.4 Plotting

To plot the balls in a map the team implemented the ROS package using_makers which plots simple shapes into RViz. The package is unfortunately in C++, but due to how complicated implementation would be without this package the team decided to use it. The script basic_shapes was rewritten to python so we could make significant changes. In its edited state, the script subscribes to the four Vision publishers and plots the four differently colored balls to the map. The map is used only to display the shapes and not as part of path planning.

4 End Result

The end result of this project is a robot that can more around a space autonomously and find balls. It is a reasonable proof of concept. Though the portions of the code where the locations of the ball are put of the map are non-functioning and not calibrated, all of the sections - vision, path planning and map making - work exceedingly well. There is very little error in the map making (as seen by the precise shape of the balls in the map) and if it is confident enough in the map, it can correct it's potential position. Path planning is more effective than random walk because it decides on the path to take based on what is around it. Even if the waypoint is not the best final location, the information it gathers on the way there is what is important. The vision is well calibrated so there are very few false positives and the balls detected almost all the time. We are very pleased with this end result.

5 Design Decisions

Originally we were considering doing exactly what ROS was pretty much built to avoid - threading the different scripts in parallel as the processing should all be done at the same time. Thankfully we realised this mistake and switched to using nodes as they were designed to function.

In addition to simply using the nodes, we made a specific design decision regarding communication with the user. Originally to communicate the locations of the balls we were publishing a list of lists using a custom message. However, by splitting the publisher into four publishers, one for each color, we were able to use Vector3 and not deal with custom messaging. This was much simpler and easy to get working rather quickly. This decision made the C++ code that we were editing slightly more difficult, but was still faster then creating a new message type.

6 Challenges

Originally we started working with a package, hector_worldmodel, that would have allowed us to easily path plan, keep track of objects and map. Due to the simplicity possible if we could determine how to use it, we wasted a lot of time attempting to get this package that had virtually no documentation to work. Eventually we were forced to move on. Rather then using a package that would have allowed us to do all of these things at once, we instead used a set of packages that together, about equaled the single package.

This was the first time we were really using nodes to run our code other then just collecting data or pushing motor commands. It took a little while to figure out how to create our own messages.

This project was perhaps the first in this class where scheduling became a real problem. As we entered finals and the end of the semester, the amount of time that we could all meet together was greatly reduced. Though this was a significant challenge, we adapted to work more individually then we have done in previous projects as a team. This is perhaps less productive as a certain amount of communication was lost, but relatively effective.

7 Future Improvement

Better waypoint generation would be one of the first issues to resolve in the future. Though there is relatively excellent path planning happening to get to each set location, the way that we generate the actual goals for the path planning is perhaps not the most optimal as currently it is random.

In addition, more precise computer vision would be helpful. At this point the balls are recognised very easily, but the data for their physical location is not accurate. The Hough transform circle detection could be calibrated more effectively but it was deemed good enough for this iteration. In the future, the location data of the balls could be much improved.

8 Lessons Learned

We learned a lot more about the actual structure of ROS in this project then in past projects. In both the navigation and vision projects, for the most part, we had a single script that subscribed and published to the robot, but most of the communications for that were written for us. In this project we needed a set of nodes and packages to run in parallel, as it was far too complex for a single script. To accomplish this it was necessary to publish and subscribe to messages that we created ourselves. At one point we even ventured into creating a new kind of message. Though we did not end up deciding to create a new message type, this was explored deeply enough that it is understood and could probably be implemented.

In addition to the message tangent, there were a whole bunch of very different parts of ROS that we did a decent amount of exploring into. For example, there is so much more to hector_slam then we originally thought. For example, there is hector_worldmodel, which does almost the entire project. It path plans in an unknown environment, keeps track of and plots objects that are passed to it and maps the area around it. With this package the only thing we would have had to do is use computer vision to find the obstacles and pass them to hector_worldmodel. We contacted the creators of the stack but did not have any success, and thus could not acquire any further documentation.

However, our use of "would have" brings us to our next lesson learned. Lack of documentation for packages in a language we don't know is not easily resolved by spending time on figuring out how the package works. A better use of the time is to find packages that are easily used and create the extra code to work with them, which is what we ended up doing.

We also learned to write launch files, which is a highly relevant skill for all of us.

References

[1] A. Stentz 'Optimal and Efficient Planning for Partially-Known Environments' (1994)

[2] G. Oriolo, G. Ulivi, M. Vendittellig 'Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environment' (1998)

[3] A. Stentz 'Optimal and Efficient Path Planning for Unknown and Dynamic Environments ' (1993)

[4] P. Das, R. Laishram, A. Konar 'Path Planning for Mobile Robots in Unknown Environment' (2010)

[5] T. Errson, X. Hu 'Path Planning and Navigation of Mobile Robots in Unknown Environments' (2010)