Rescuebot
Computational Robotics Fall 2014: Search Challenge

Authors: Christina Fong, Cypress Frankenfeld, Victoria Preston
Time spent: ~

TABLE OF CONTENTS:

I. Project Overview

Our project, rescuebot, is a case study in autonomous reconnaissance for ground robots. Given an unknown environment, the rescuebot must explore, find items of interest, and log information about the scene. At the end of an exploration run, it should be able to communicate the locations of items of interest with relative accuracy. This project unites elements of mobile robotics approaches to mapping and movement with computer vision topics in object recognition.

In the class-wide search challenge competition, the items of interest (red, blue, yellow, and green kickballs) were located in CC300, a multi-room area. The robot had access to a 360° LIDAR laser scanner, an optical camera, and a set of built-in bump sensors. It used these to successfully navigate the rooms, identify the location and color of each ball, and display these on its map of the environment.

Our minimum viable product for this project was to have a robot that could navigate the STAR Center, create a (not necessarily accurate) map, and find and identify one ball and locate it on

the map (not necessarily in the correct place). Our stretch goal was to create an accurate map of the STAR Center with accurately located objects, correctly identify all colors, and win the competition.

Our learning goals included:

- Gaining a deeper understanding of SLAM algorithms
- Gaining more experience with OpenCV and computer vision development, especially conceptual and dealing with environmental changes
- Gaining more experience with ROS best practices (eg: visualizing things with rviz, testing with bagfiles, writing launch files, incorporating built-in packages, writing testable code)
- Practicing good coding hygiene and techniques (especially with multiple people and codebase integration)
- Practicing mapping techniques

II. Navigating the STAR Center

We considered many approaches to navigation throughout the course of this project, including: creating a map with SLAM and using it to path-plan, using pre-built packages like hector_exploration, implementing a random walk, moving along a space-filling polygon, and simple behaviors like wall following and obstacle avoidance. Although we initially pursued a hector_slam + hector_exploration approach, as it has been used to great effect in the RoboCup Search and Rescue competition, we abandoned that due to difficult implementation and lack of documentation.

Ultimately, we went with a finite state machine approach, with states for ‘room centering’ , ‘wall following’, and ‘obstacle avoidance’.

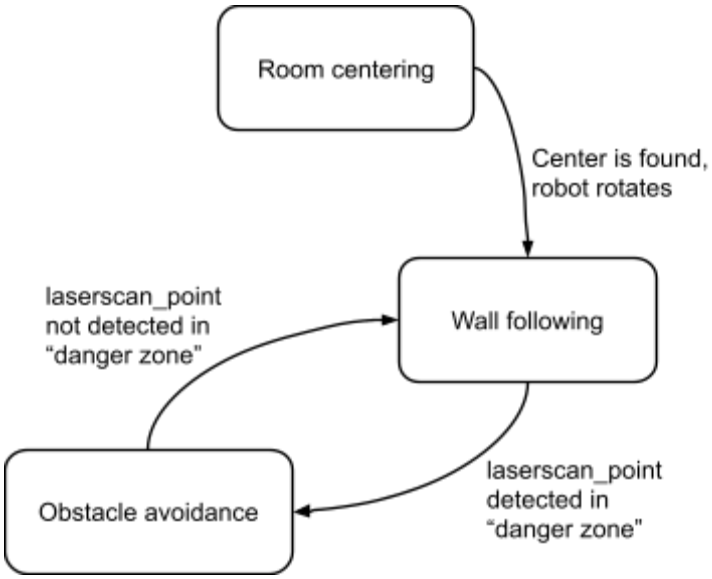


Figure 1: State machine diagram

The robot begins in the room centering state. After successfully centering itself and rotating, it enters the wall following state. It remains in the wall following state unless it detects something in its “danger zone”, at which point it transitions into obstacle avoidance. Once there are no points in the danger zone, it returns to wall following.

Room centering

In the room centering state, the robot travels to the center of a room and slowly spins 360° once it is there. This allows it to quickly and efficiently observe an entire, moderately sized room with its camera. If a target ball is located somewhere in the room, this should allow us to locate and identify the ball without having to traverse the entire room.

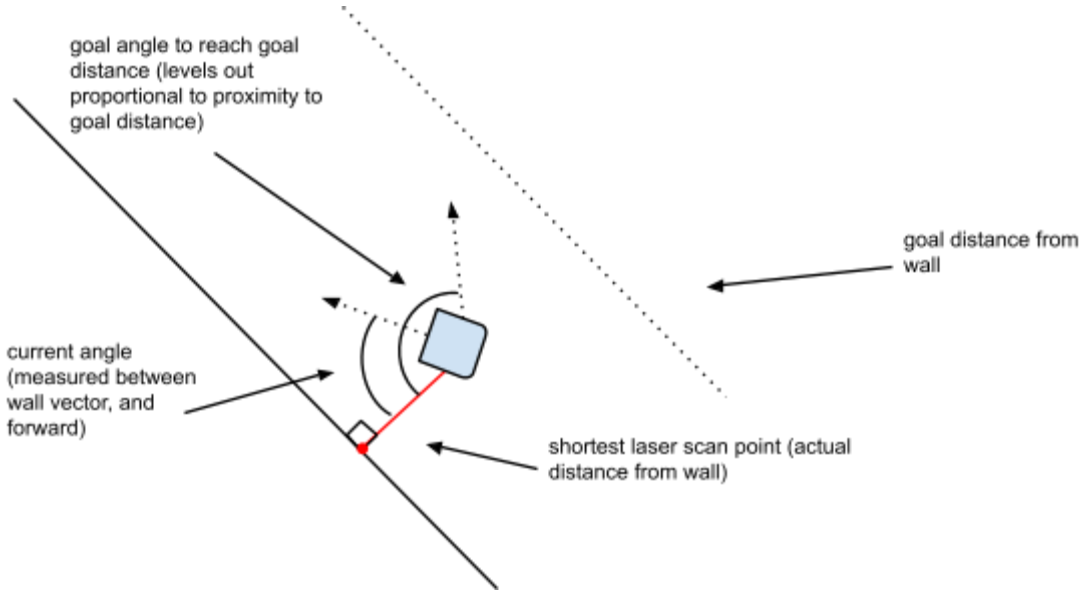
The robot identifies the center of the room as the point where the standard deviation of all its laserscan data is low (below parameter ROOM_CENTER_CUTOFF). Theoretically, a low standard deviation means that the robot is equidistant from all points. (Note: this approach is more prone to error if there are many obstacles in the room.)

After going to the center and rotating, the robot automatically enters the wall following state. Note: there is currently no way to return to the room centering state after entering wall following. This is suitable for the STAR Center problem, as the environment consists of one large room with one corridor and one smaller, narrower room. In different environments, such as those with multiple large rooms, it may be desirable to return to the room centering state.

Wall following

In the wall following state, the robot identifies the closest wall and moves so that it is parallel to the wall and a desired distance away from the wall. This allows the robot to continuously follow the wall while dynamically correcting if it gets off-angle or too close/far away. It controls the distance from the wall by measuring

```
diff_distance = desired_distance - actual distance
desired_angle = proportional_constant * diff_distance
diff_angle = desired_angle - current_angle
angular_velocity = proportional_constant * diff_angle
```



Obstacle avoidance

The robot begins to obstacle avoid if it detects objects in its danger zone. The danger zone consists of a parabola in front of the robot. The robot slows down its linear velocity proportional to the proximity of the closest point in the danger zone. If the robot detects a point very close in the danger zone, it will halt. Obstacle avoidance doesn't change the angular velocity. The wall following code was robust enough to orient the robot to drive parallel to all obstacles, but the obstacle avoidance code was used to slow the robot down and allow it enough time to turn away from things like corners.

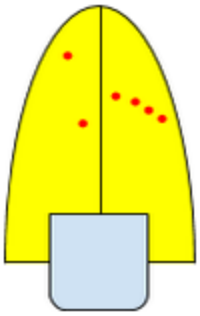


Figure 2: Danger zone used for obstacle avoidance, with example point data shown

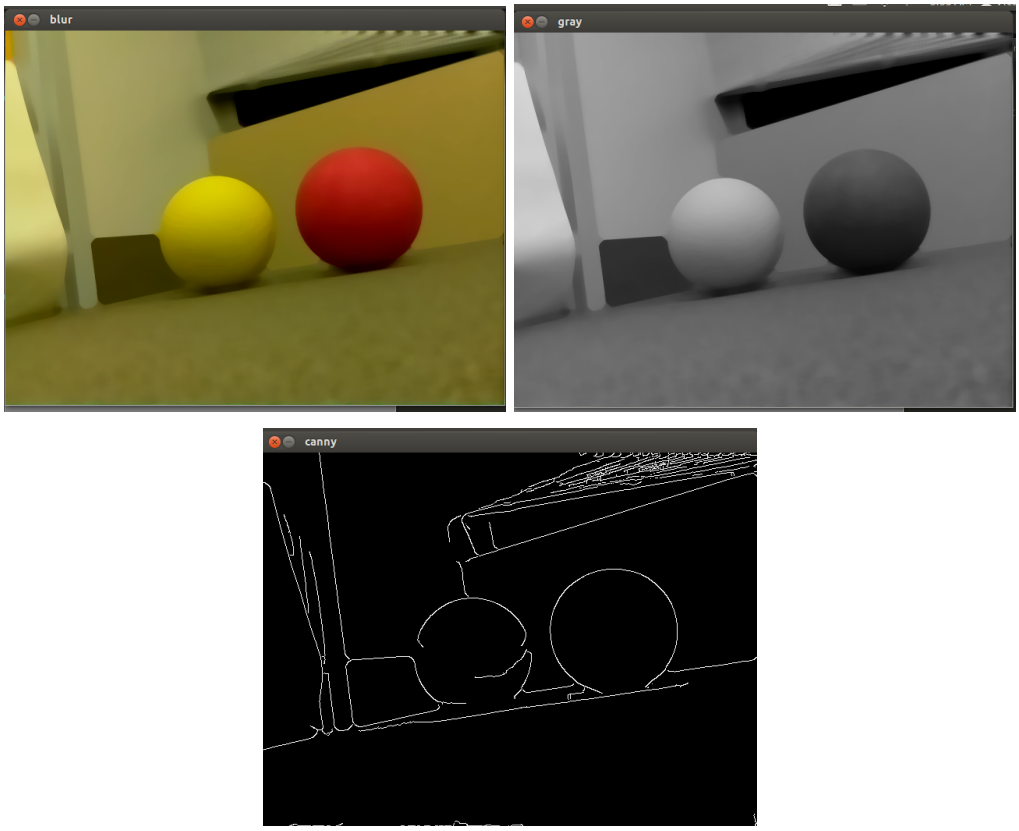
III. Recognizing target balls

Object recognition is conducted in the image_converter node (initialized in seer.py). It consists of two main steps: finding the circles, and then identifying what colors they are.

Finding circles

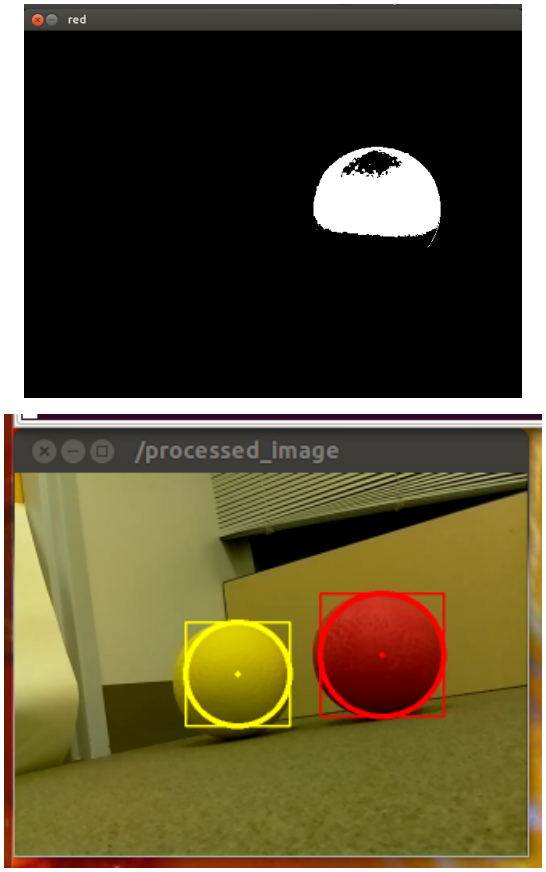
We begin by pre-processing the raw camera data in the callback function to reduce noise (using medianBlur), convert to grayscale, and apply Canny edge detection. We then apply the Hough

circle transform function to the pre-processed image to identify possible balls in the scene. However, the Hough circle transform has a high rate of false positives at times. This means that there must be a secondary check on the system.



Identifying colors

We cross-reference the Hough circle transform results with HSV image masks for each ball color (red, blue, yellow, green). This allows us to both reduce/eliminate the number of false positives and identify what color each ball is. To address variation in color caused by lighting conditions, we compared the average value of the HSV mask taken over the identified circle, and if that is greater than some calibrated value, assume the identified circle is the appropriately colored ball. Additionally, we were able to change the camera settings in order to have a consistent image no matter the time of day or ambient light in the room.



We chose to identify circles before colors, even though Hough circle transform had such a high false positive rate, for decreased computational load. With this order, we only have to create and process the HSV image masks for areas identified as circles. If we were masking/filtering for colors first, we would have to create four separate image masks for the entire frame size, for every frame. We added the HSV values to the slider gui so we could adjust for varying conditions each time we run.

IV. Mapping and output

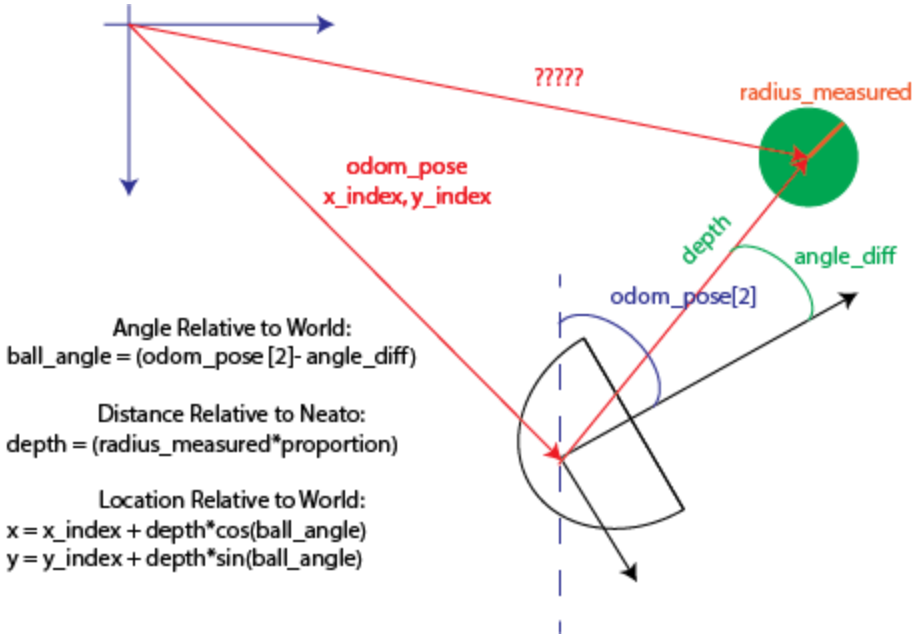
We have decided to map both ball locations and a map of the room during exploration for ease of understanding locations and debugging.

Map of the room

In order to map the room, we created a class called OccupancyMapper which subscribes to the laser scan data from the neato, and plots the viewed walls or obstacles onto an image. It uses a probability distribution to determine what is a wall, and what is not. Consistently viewed objects clearly have a higher probability of being walls, so they are marked. The robot is also tracked about the room. The odom_frame is referenced on the map in order to plot the robot's position. The orientation of the robot is not captured in this mapping style.

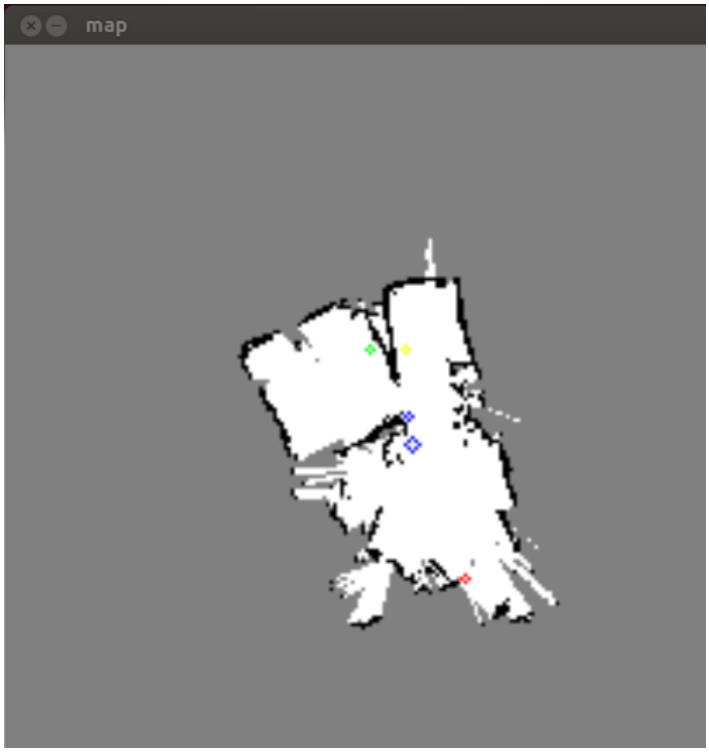
Adding the balls

In order to add the balls to the map, the reference frames must be considered. Of the world, base_link, and odom frame, we selected the odom frame to map the balls relative to the orientation and location of the robot. At the end of a run, we could then convert our mapped coordinates to physical locations in space to compare with a master map of the world.



The 'Real World'

We also apply a small transform to get the coordinate of the world with relation to the start point of the robot. Given some preliminary tests, these have been acceptably accurate. An example of a generated map is provided.



V. Dynamically reconfiguring parameters

As the challenge rules allow us to modify our system's behavior through interfaces (while the robot is in the recovery area), we implemented a set of dynamically reconfigurable parameters using the dynamic_reconfigure ROS package. The dynamic_reconfigure package allows parameters to be controlled through the use of PyQT sliders. We used this to modify:

- maximum velocities (MAX_LINEAR_SPEED, MAX_ANGULAR_SPEED)
- size of danger zone (DANGER_ZONE_LENGTH, DANGER_ZONE_WIDTH)
- scaling multiplier for danger zone points (DANGER_POINTS_MULTIPLIER)
- desired distance from wall for wall following (WALL_FOLLOW_DISTANCE)
- number of points to consider for room centering (room_center_number_points)
- maximum standard deviation for room centering (ROOM_CENTER_CUTOFF)

Although initially implemented for competition, being able to modify these parameters easily while the code was running was incredibly valuable for testing.

VI. Challenges faced

Hardware

Motion blur and inconsistent lighting/exposure on the camera were the main hardware challenges we faced. These were resolved by changing the Raspberry Pi cameras to sports mode and matrix metering mode for exposure.

Additionally, drift in the Neato odometry data was exceedingly problematic in acquiring coordinates for the balls and creating a map of the environment. Some Neatos had worse drift than others, but it was a universal problem that we faced.

Software

- Lack of documentation on third-party ROS packages -- we spent a lot of time trying to implement hector_navigation, but eventually gave up due to lack of documentation
- Wall following -- in general, robust wall following was a challenge algorithmically (especially with corners and doorways)
- Coordinate frame references - we had trouble converting the data into the appropriate coordinate frames. This was eventually resolved with a little bit of trig and critical thinking. There are still improvements that could be made to the robustness of the mapping and granularity
- Circle detection - Hough transform worked well, but wasn't necessarily consistent in diameter detection, even in still frames. This definitely impacted the readings and mappings that were made. Potentially adjusting the values, or the Canny settings could be done in future improvements

VI. Future improvements

There are many ways we could extend this project or make it more robust, with additional time and effort. These include:

- Improving navigation to optimally search in likely locations and/or efficiently move towards desired locations (higher-level path planning as opposed to simple wall following behaviors)
- Improving robustness of room-centering method and wall-following (if still using those techniques), to better deal with obstacles
- Improving state machine triggers
- Ability to autonomously self-calibrate HSV color masks under different lighting conditions
- Cross-referencing LIDAR and image data to get more accurate ball locations
- More robust mapping methods
- 'Smart' object recognition - a system that could immediately pick out the 'out of place' objects (like brightly colored balls in a stark space)
- SLAM techniques on mapping to help with consistent navigation

VIII. Applications to future robotics projects

This project used elements of mapping, exploration, and computer vision. All of these in their own right have many applications in robotics projects - from developing topographic maps of otherwise dangerous spaces, to optimally navigating a set mission. This particular challenge has very close ties with similar robotic applications in search and rescue, autonomous navigation and identification, and environmental monitoring.