Color Detection And Recovery Robot

Jun-Jee Chao, Ren Jeik Ong, Khai Nguyen, Grant Udstrand
Course: CSCI 5551: Introduction To Intelligent Robotic Systems
Advisor: Prof. Junaed Sattar

December 15, 2019

Abstract

This project set out to create a robot with color-detection capabilities combined with basic self-navigation. The robot was to identify targets based on colors, and approaching to targets. Self-navigation with a pre-built map was implemented to reach specific area. This was accomplished using the Turtlebot. Simulator was used for initial algorithm development.

1 Introduction

In the realm of robotics, it is of prescient moral demand that robotics are used to solve problems which adversely affect the ability of humanity to continue living their lives safely and by convenience. One common application to increase the living standards of all humans is to replace dangerous occupations with automated participants so as to risk circuitry rather than human life. One of these occupations is search and rescue. Beyond this member of the problem space, there are also factory jobs which deteriorate the human condition that could be solved by automation such as carrying a box from one part of a factory to another and identifying the drop zone based on a color pattern on the floor. These techniques for color detection are thus useful and coordinating them in a robotic programming sense with some form of mobile direction could prove useful in a variety of environments. The purveyors of this project concede that the machine in this experiment is rudimentary and incomplete, but the concepts that were explored in the creation of this robotic system could be extended by other researchers to solve a broad range of problems that are currently solved through human expenditure in time, effort, and physical exertion, and often all three in a variety of risky environments.

2 Methodology

2.1 Turtlebot and Simulator

In this project, we implemented our method on both Turtlebot and simulator. For Turtlebot, we are using Turtlebot2 along with ROS indigo. The simulation is implemented with Gazebo, ROS kinetic, and Turtlebot3. OpenCV2 is required for both version in order to run our object approaching node. We also created a red ball model to simulate the object that the Turtlebot is suppose to find and approach.

2.2 Map Building and Navigation

In order to self-navigate to a specific location. We have to first build a map of the surrounding. Our initial pitch was to use Velodyne VLP-16 as our lidar sensor to build the map with SLAM. However, after setting up VLP-16 and visualizing the data in Rviz, we found one of the package's version isn't compatible with

our SLAM package when doing Cmake. After few attempts of changing Cmake version, and building some package from source, we decided not to go deeper. Moreover, the power of VLP-16 is another problem since it has to be plugged into wall socket and we don't have a battery that is available. Finally, we ended up implementing the Turtlebot navigation gmapping package and control the bot with keyboard teleop to build the map of the room. We further load the map with Rviz, and the Turtlebot is able to navigate itself to the specific position input by the user.

In addition to build the map using a real robot, we also created a map in the simulator. Turtlebot3 and House were launched in Gazebo for this project. Similar to what we did with Turtlebot2, we launched the SLAM package which is using gmapping method, and moved the Turtlebot3 around to build the map. Also, Turtlebot3 is able to navigate itself to the location that is selected by the user in Rviz.

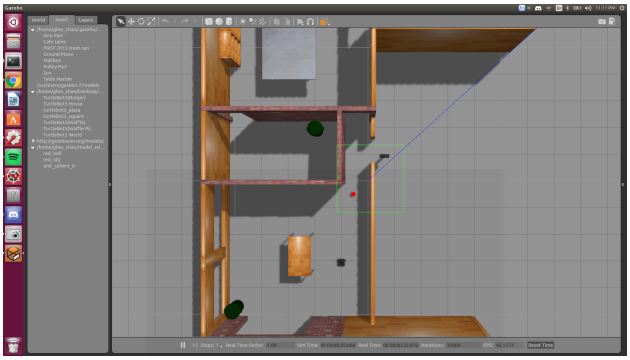

Figure 1: Turtlebot3 house in simulator.

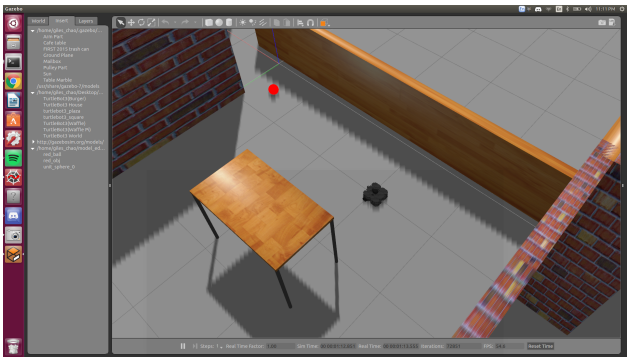

Figure 2: Ball model created for the representation of the target.

2.3 Object Detection and Approaching

After reaching to the location input by user, our project aims to approach to the desire object, which we assume the desire object is a red ball in this project. We built a ROS package that contains a node that subscribe to a RGB image topic, and publish message to `cmd_vel` to move the Turtlebot.

2.3.1 Find Target Coordinate

In order to identify the desire object from a RGB image, we built OpenCV2 from source and add it to package dependency. `cv_bridge` was used to convert ROS images message to OpenCV images. We first transform the image from RGB to HSV color space since the RGB value in a digital image is correlated

with the lighting condition which makes it difficult to separate different colors. It is more accurate for us to set the upper and lower bound to identify red in the image after converting to HSV space. We further created a mask to represent all the red pixels that lies in the range. Erosion was applied to get rid of noise after obtaining the mask, and dilation was applied after erosion to enhance our target. Then we applied findcontour to get all the contour in the mask, and with the assumption that our desire target has the biggest area in the mask, we find a circle to fit the contour that has the biggest area. Finally, we are able to get the radius and the center coordinate of our target.

2.3.2 Approaching Strategy

Now we have the specific coordinate (u ,v) in the image that we want to approach. Inverse projection was applied to find out the angle to align camera center with the desire coordinate. As shown in figure 3, for each point (u ,v) in the image, we can find a 3D ray that points to the real 3D coordinate (X ,Y, Z) from the center of the camera. Image projection can be represented as equation 1, where K is the camera intrinsic parameter and could be found either by camera calibration or from the sensor_msgs/CameraInfo message in ROS. Given K and (u ,v), we are able to find out the 3D ray by inverse projection in equation 2. Since Turtlebot only moves on the ground plane, we project this 3D ray to the camera's x-z plane. And by calculating the dot product between camera's z axis (0, 0, 1) and the normalized projected 3D ray, we can get the value of $\cos\theta$, where θ is the angle difference between camera facing direction and the target direction. We further implemented proportional control with $k_p = 1$ by setting the vel_msg.angular.z= $\pm\theta$. The sign of θ can be determined by checking the X axis value of the 3D ray. After aligning the camera center with the target, we check the detected object radius and the threshold we set to determine if the Turtlebot has to go forward or not. Since we are using only RGB image, the distance is simply represented as the ratio to the object and the image size. Also, this algorithm allows Turtlebot to follow a moving object.

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, K = \begin{bmatrix} f_1 & 0 & C_x \\ 0 & f_2 & C_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \lambda K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2}$$

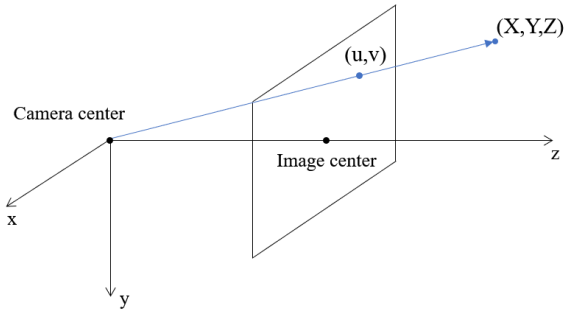

Figure 3: Image projection.

3 Result

3.1 SLAM Mapping and navigation

In this project, we successfully built the map of the teaching lab and the house in the simulator. And the Turtlebot could navigation itself to the specific location input by the user in both scenarios. We can easily tell that the simulator has less noise than the real environment, which make simulator a very useful and convenient tool to test the prototype of algorithm, and the real environment is where we can test the robustness of the algorithm. Video of all the results are attached.

Figure 4: (a) SLAM map of the teaching lab. (b) SLAM map of the house in simulator.

3.2 Object Detection and Approaching

After reaching a specific location, the Turtlebot will look for target by spinning itself. Once it found and object, it align itself with the target then move close to it. The ROS node we built is able to read an image message from /camera/rgb/image_raw topic, then convert the image message to OpenCV image message. We further created a window that shows the view from the Turtlebot. As shown in the left of figure 5, it allows user to see from the Turtlebot's view, and the target will also be circled with blue line to indicate the object it's aiming for. This node publishes two message. The mask after applying image processing method is published to image_mask, as shown in the right in figure 5, it allows user to check if certain threshold is suitable for detecting desired objects. Linear velocity along x and angular velocity along z are published to /cmd_vel according to the vision-based computation result.

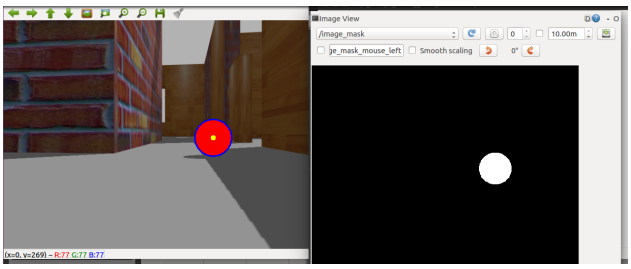

Figure 5: Turtlebot view and mask after image processing.

4 CONCLUSIONS

The design of the algorithm was ultimately a success as it was found that the robot could perform object detection and approaching appropriately. Object tracking was implemented. SLAM mapping and naviga-

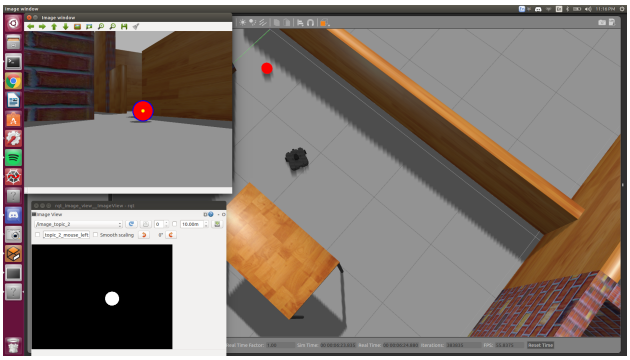

Figure 6: Object searching and approaching in simulator.

tion was also successfully implemented in this project. Several image processing methods were applied to locate the object in the image accurately. Vision-based navigation was successfully implemented as inverse projection was applied to find out the target direction in 3D space with only 2D image, and the angular velocity for turning is directly calculated by the angle between two vector in 3D space.

5 Future Work

This project can be further develop into a delivery robot. By combining the navigation and object approaching algorithm. The robot is able to deliver things such as coffee or mail from one room to another. After approaching to a certain area, people can trigger the robot by showing a QR code so that the robot can carry something back to where it came from. There is still room for the object detection algorithm to improve, since pure color based object detection won't be stable in complex environment. Machine learning based detection could be applied to improve the stability. Besides object detection, the robot must be able to avoid obstacle efficiently. This could be achieved by using RGBD camera along with LIDAR sensor. Moreover, a gripper could be attached to make the robot capable of finishing more complex mission. Forward and inverse kinematics could be applied to move the gripper to certain position along with camera. Hand-eye calibration could be implemented if there's a camera attached to the gripper.

References

[1] Park, H. S. (2019). Camera model [PowerPoint slides]. Retrieved from https://www-users.cs.umn.edu/~hspark/csci5561_F2019/Lec27_CameraModel.pdf

[2] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).

[3] Sural, S., Qian, G., & Pramanik, S. (2002, September). Segmentation and histogram generation using the HSV color space for image retrieval. In *Proceedings. International Conference on Image Processing* (Vol. 2, pp. II-II). IEEE.