
A Robot to Collect Objects by their Color

BLG456E Project Report by Merdobot

Mehmet Barış Yaman - 150130136

Muhammed Burak Buğrul - 150140015

Pelin Hakverir - 150140031

Kadir Emre Oto 150140032

Mertcan Yasakçı - 150140051

Hüseyin Aküzüm - 150140103



Table of Contents

A Robot to Collect Objects by their Color	1
BLG456E Project Report by Merdobot	1
Table of Contents	2
Abstract	3
Introduction	4
Background and Problem Review	5
Problem Solving Design	6
Implementation	7
Requirements	13
Results & Conclusion	14
Video Demo	14
Reference List	15

Abstract

In this report, all of the progress we made in our project is explained in detail. A brief introduction/statement of our project is made. Problem background and a similar academic work supplied and explained. Designs made before implementation and their connection to problem are stated. Technical details are explained in implementation section for our project. After all, a brief conclusion added to the report.

Introduction

Greatest feature of our robot is go into places that humans cannot reach easily and collect necessary objects and bring them back according to given commands by humans. We believe that this is an important problem because there are many places like this in our world. Places like damaged nuclear reactors, mines and unsafe buildings are basic examples to that kind of places. Since we do not have enough time or budget to accomplish big tasks like that, we aimed to solve a “garbage” collection problem in our project. In our project, we aim on a robot that can take necessary commands from a human and then search these places. We are going to use TurtleBot with a gripper attached to it. This project is interesting because we know a little about robots however; we are ambitious to do this kind of project. We believe that this project is an interesting one because it contains various parts. The main approach is going to be first add the gripper to TurtleBot and then add necessary components to it to understand and react to humans’ commands.

Background and Problem Review

In this project we aimed to solve a collecting problem. In our context, we designed our system to collect objects according to their colors. Our problem is a general problem that can be specialized to certain areas. For example, garbage collecting problem, selective collecting/separating problem etc.

While working on our project, we decided to make a literature review in order to guide us in this project. For this purpose, we found a paper named “Evolving non-trivial behaviors on real robots: A garbage collecting robot” (Nolfi, 1997).

In this paper, analyses and explanations for a “garbage” collecting robot are stated. Basic approach in project that is stated in this paper is to staying away from arena walls while searching for a “garbage” (Nolfi, 1997). After such an object is found, purpose of the robot in the project that is described in the paper named “Evolving non-trivial behaviors on real robots: A garbage collecting robot” is to take that garbage object out of the arena (Nolfi, 1997).

We thought that, our project has similar sides with the paper’s project. Robots used in both project has similar capabilities and features except their sizes. We used Turtlebot whereas in the paper’s project a miniature mobile robot named Khepera was used. Main differences between the projects are controller and algorithm. In the paper’s project a neural controller and genetic algorithms are used to accomplish this task. However, we did not use such controllers or algorithm because of our limited time and resources.

To solve our problem, we used a gripper attached to the Turtlebot. Gripper actually used to imprison object. So that objects will move with robot. For modelling the gripper, Blender program used. After designing and drawing of the gripper, URDF files used to connect gripper with the robot.

Our motivation for this project shaped mainly by the paper we reviewed. We aimed to implement a problem solving robot in a simulation environment. We saw that our project can be extended to various types to solve different problems. Also techniques used in the paper are all interesting and creative subjects to us. All of these combined, gave the motivation we need to implement our project.

Problem Solving Design

In order to solve the problem explained in previous sections, we decided to use a modified version of Turtlebot. Our robot is basically a Turtlebot with a gripper attached.

We decided to place different colored objects in a maze-like environment for our robot to pick them up. Also, we decided to use voice commands to collect specific-colored objects.

We designed a system where our robot waits until a valid command is given to it. After receiving a valid command, it goes to a journey. Commands can be color names. While our robot is in a journey, it searches for a specific-colored object to pick up. When it finds such an object, it returns that object to starting point.

This design solves our stated problem which is a garbage collection.

Different components are implemented individually and integrated all together in the end of the project.

Implementation

Gripper Integration and Gripping

For modelling and creating a solid base for gripper and its components, blender software was used. Blender is software that is designed for modelling, rigging and rendering purposes (Blender Foundation, n.d.). It has a powerful set of tools even for motion tracking and game creation. In our project, blender is used to create gripper components. Also blender is free and open source software that can be used commercially or for education.



Figure 1 – Turtlebot with gripper

For detailed editing and Gazebo integration, first we aimed to use .sdf file type. However, since Turtlebot's model is made using .urdf and .xacro files, we decided to go with .urdf and .xacro.

After designing gripper parts one by one, we exported these blender designs in a format that Gazebo can understand. Using these mesh files we created by blender, .xacro and .urdf model of the gripper implemented. In our gripper model file(urdf.xacro) we have 3 links that corresponds to the parts that our gripper has(left arm, right arm, base). Since our gripper consists of 3 parts we need 3 joints to use it functionally.

```

<link name='right'>
  <collision>
    <origin xyz="0.2 -0.15 0" rpy="0 0 -0.25"/>
    <geometry>
      <mesh filename="package://turtlebot_description/meshes/gripper/gripper_right.stl"/>
    </geometry>
  </collision>
  <visual>
    <origin xyz="0.2 -0.15 0" rpy="0 0 -0.25"/>
    <geometry>
      <mesh filename="package://turtlebot_description/meshes/gripper/gripper_right.stl"/>
    </geometry>
  </visual>
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0.2 -0.15 0" rpy="0 0 -0.25"/>
    <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz="0.01"/>
  </inertial>
</link>

```

Figure 2 – Right arm part in urdf.xacro file for gripper

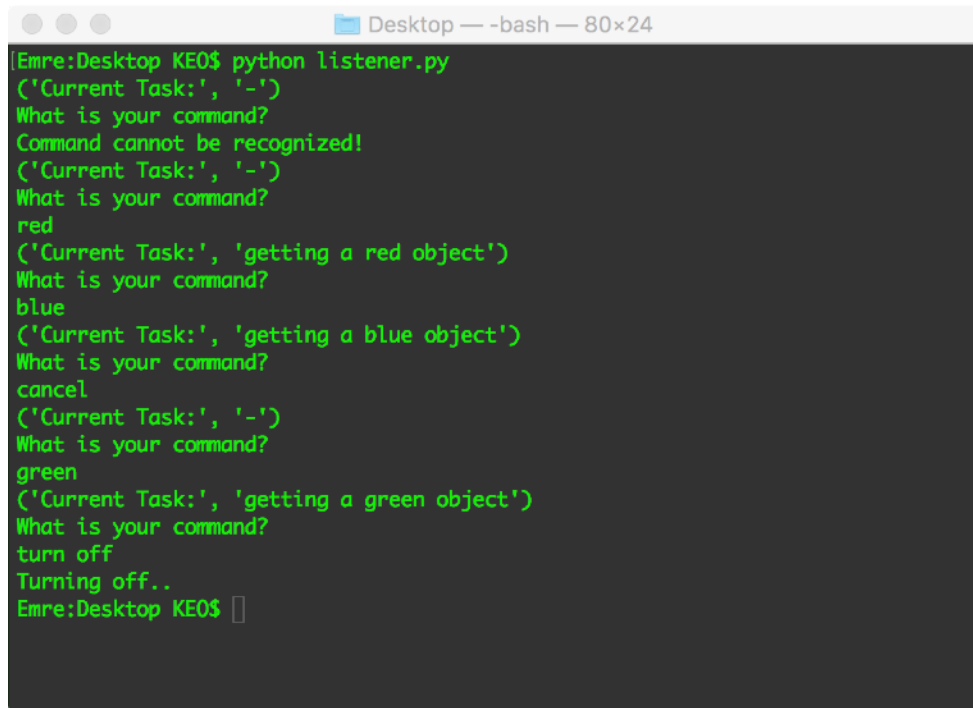
Arms are connected to the base via revaluating joints. Because arms needs to rotate in order to imprison objects. However, base of gripper connected to Turtlebot via fixed joint. Because we do not want a gripper base that can be moved.

Connecting gripper to Turtlebot is not enough. It must be controllable by programs. To achieve that, we used joint_state_publisher console command to apply force to our movable gripper arms. Using these command we are able to open or close our gripper's arms to imprison objects.

Speech Recognition and Commanding

We decided to take voice commands from user. So that, we need a speech recognition module to accomplish this task. We designed our system to wait until a command is supplied. After a command is received our robot will act according to that command.

To transform a voice to a command, we used a python module, which is called SpeechRecognition. This module supports several famous apis such as CMU Sphinx Api (works offline), Microsoft Bing Voice Recognition Api, and Google's Speech Recognition API. We preferred Google's Speech Recognition API. Recognition program works in threaded mode and transforms voices supplied via microphone to command strings. Available commands for our robot are the color names. When such a command is supplied to the robot, it goes to a journey with a hope to find an object with that color. Also at any moment "cancel" command can be given to cancel any command and "turn off" command will terminate the program.



```
Emre:Desktop KE0$ python listener.py
('Current Task:', '-')
What is your command?
Command cannot be recognized!
('Current Task:', '-')
What is your command?
red
('Current Task:', 'getting a red object')
What is your command?
blue
('Current Task:', 'getting a blue object')
What is your command?
cancel
('Current Task:', '-')
What is your command?
green
('Current Task:', 'getting a green object')
What is your command?
turn off
Turning off..
Emre:Desktop KE0$
```

Figure 3 – Example Output of Listener

Object and Color Recognition

Even if we first thought to use point cloud to recognize object, we implement our object recognition script with scipy along with numpy and PIL. Because, point cloud has a hard building process and its documentation contains many deprecated functions. That's why we choose scipy to recognize objects.

First, object recognition script takes an image (screen capture taken from turtlebots camera) to process and opens it with PIL. Second, script counts occurrences of colours in the image except greys. Because grey pixels are belong to environment. Third, it detects edges in the image. But this detection is not a proper one. Third, script completes the empty parts in the edges. and counts number of objects in the image. Fourth, script determines the colour of the object if there is any object. Fifth, it calculates weighted averages of the determined pixels of determined colour in the x axis. Finally, it calculates the occurence rate of the determined colour in the image.

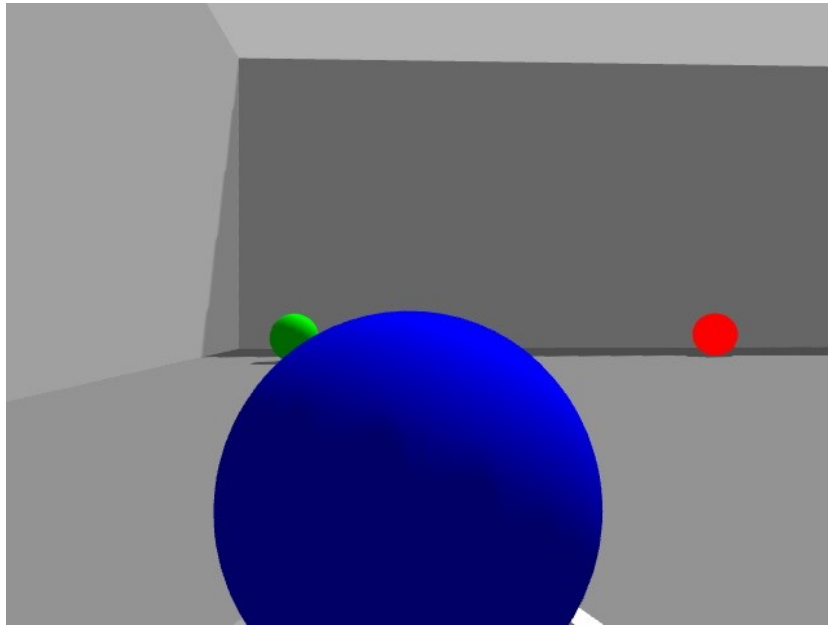


Figure 4 – Screenshot taken from TurtleBot

World Building

The world for the robot to travel around is built in Gazebo, via Gazebo's Building Editor. The spheres for the robot to grasp are created in this world, with Gazebo's Model Editor. Weights of the objects are 0.001kg, radius of the spheres is 0.15m and their color differs between red, green and blue. The created world can be seen in Figure.

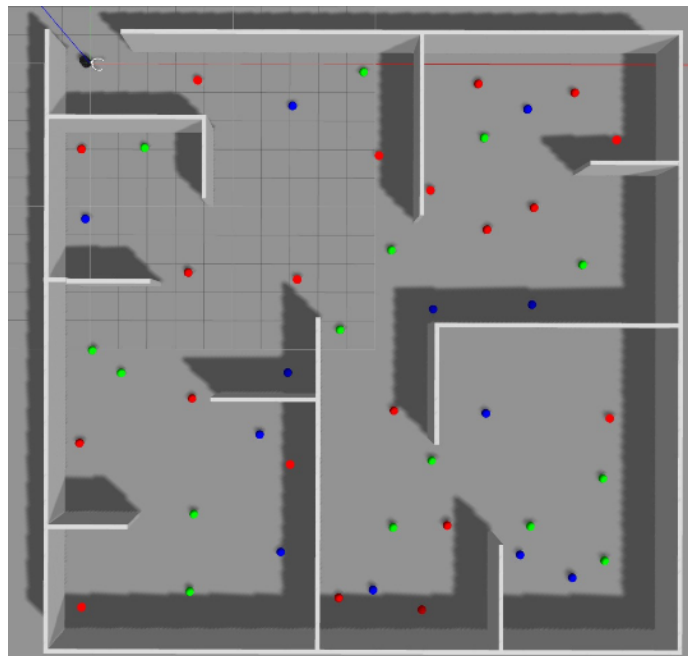


Figure 5 – Merdobot World

Mapping

Generally Mapping process is done using AMCL or Gmapping. Since we do not need special coordinates to make our robot reach, we chose Gmapping for mapping process.

Since the map that our group considered is large enough, adding a Laser to the robot could make the Mapping process be easier. Therefore, we searched about how to add a Laser to the Turtlebot. The laser should include “ray” instead of “gpu ray”. After a one week study, we added a Laser to the Turtlebot. The laser has the movement angle of 270 degrees and sees at most approximately 30 meters different than the normal Turtlebot. Thus we can accept that that is only true in working with Simulation, not in real robots, when it comes to reality.

But in that time, so many project files such as turtlebot.gazebo.urdf.xacro should be changed in terms of the laserscan topics and that may cause conflicts in Rviz program, when we run the robot environment. That can be visualised by using rqt-graphs but we could not see the problem on that graph. Because of that we needed to reinstall ROS again and that caused to lost in Laser.

Reinstalling ROS again made mapping process the hardest one for our group. The robot had lost many times and many times we rerun Gmapping again.

The created map afterwards is:



Figure 6 – Obtained tutorial.pgm

Path Planning & Returning

In this part, it is aimed to use robots' move base and send appropriate goals to bring it back to initial pose. To do this, it is assumed that the initial coordinates of the robot is $x=0$ and $y=0$. Firstly, robot takes commands from the user and starts searching for the appropriate ball in the map. If it finds the ball with the desired colour, the robot grasps the ball and returning part takes place. Firstly, we create a goal object with assumed initial position and appropriate rotation, then send this goal to move base to be achieved. We set 120 seconds duration for the goal to be achieved. If the robot reaches to the goal, gripper commands comes in. It opens up and we create a new goal with parameters $x=0$ $y=-2$. Therefore, robot leaves the ball to the initial position and steps back from the ball, then turns back and waits for the new command. If it can't achieve the goal, we assume either it hit to a wall or a package broke during this phase. Everything told above is working just fine with the default turtlebot (without gripper part, since it does not have it). However, we have a problem with customized turtlebot with gripper extension. Move_base of the robot cannot transform to map and ros kills the move_base process. Therefore, move_base cannot receive any goal since it is killed and we cannot estimate robots' location in the map and cannot send goals to move base. We have a demonstration with default robot that goes to any place(coordinate) we want in the map but not with the one with the gripper. The related code about returning to the initial position with the turtlebot without the gripper is shown in the Figure.

Integration

After all of our components are finished, we started to connect them. In order to do this, we created new files that are combining different components together.

To generate our custom world and spawn our custom robot in Gazebo environment, we created a launch file. In this launch file, our custom world file is supplied to turtlebot_world as a parameter. Also to make our custom robot fully functional, joint_state_publisher node is added. With these implementations we were able to see our robot in our custom world, however, we also need to see what our robot see. In order to achieve that we added rviz launcher to the launch file.

In order to control our robot we combined other components into a .py file. In this file we make necessary checks and give decisions according to these checks in order to move our robot carefully and successfully.

Orient & Grasp

After the object is detected and confirmed that the color of the object matches with the color from the command, it starts to orient to the object. Firstly, the robot corrects its position according to the detected angle between the object and the robot. Then, it opens the gripper to get ready for grasping the object, and goes through the object. After it arrives in front of the object with the gripper being around the object, it closes the gripper. And the object is finally grasped. The next goal is to bring the object to the initial point.

The information for the object's detection is taken from `ball_control` function implemented to the python code.

The orientation is done with `motor_command`.

Opening and closing the gripper is done with `os.popen()` function, sending the recommended commands from the gripper to the console.

Requirements

Following terminal commands should be executed for speech-recognition and object recognition features.

- ❖ `pip install SpeechRecognition`
- ❖ `sudo apt-get install portaudio19-dev python-all-dev python3-all-dev`
- ❖ `sudo pip install pyaudio`
- ❖ `pip install scipy`
- ❖ `pip install numpy`
- ❖ `pip install Pillow`

Results & Conclusion

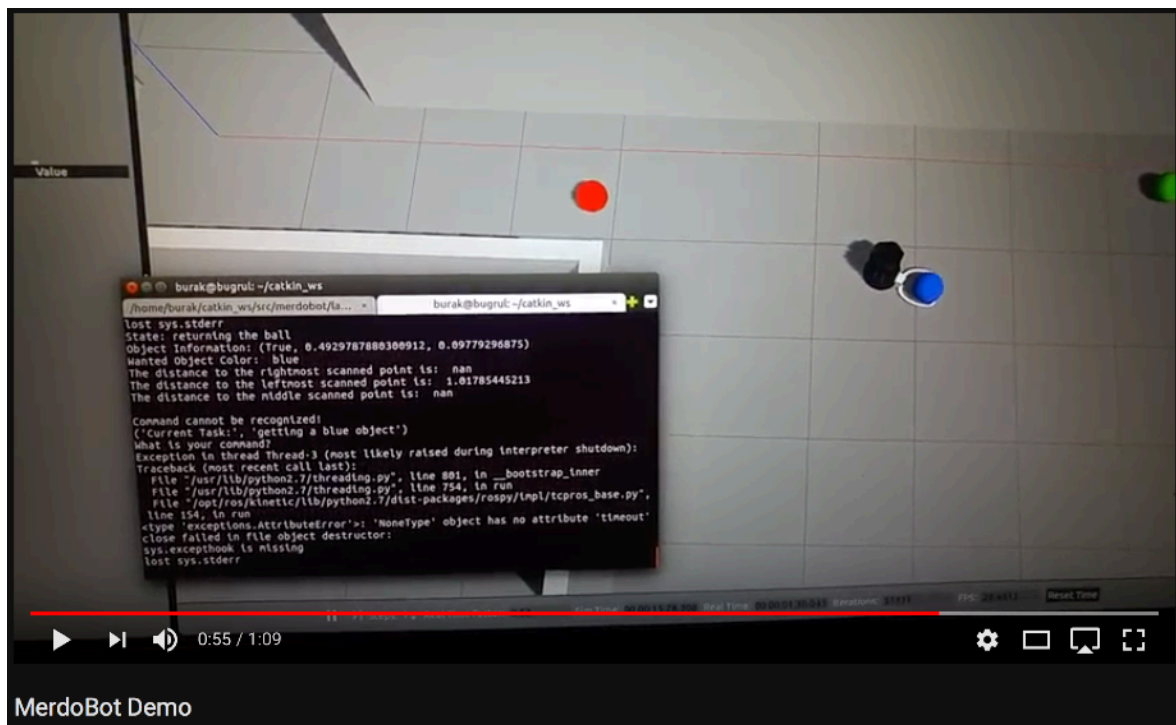
In our project, we successfully implemented our custom world and flawlessly spawned our modified Turtlebot to that world. Giving command via voice and identifying objects according to their colors and shapes works correctly.

In our original proposal we designed our gripper to actually grip objects. However, in implementation we faced some serious problems and changed our gripper's purpose to imprison objects. Robot and gripper works fine while taking commands, finding and grasping the object. However, "returning back to initial position with robot and leaving the ball" part does not work smooth as stated in proposal. Problems according to returning part explained in returning part. Apart from this problem, project implementation and run went well

Our project can be executed with the following commands consecutively:

- ❖ `roslaunch merdobot merdobot.launch`
- ❖ `roslaunch merdobot run.py`

Video Demo



<https://www.youtube.com/watch?v=ChJdR-KP6KU>

Reference List

Nolfi, S. (1997). Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robotics and Autonomous Systems, Volume 22(Issues 3–4)*, Pages 187-198.

Blender Foundation (n.d.). About. Retrieved from <https://www.blender.org>

Joshi, B. (2016). Adding Hokuyo Laser Finder to Turtlebot in Gazebo Simulation. Retrieved from <https://bharat-robotics.github.io>