

2019 Spring Robotics Project

Term Project Report

2019. 6. 27

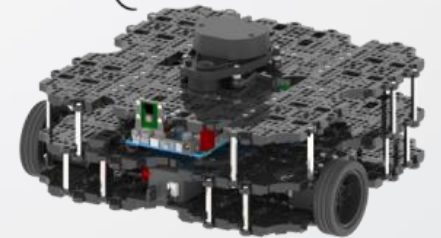
Myungjin Park (Integrated course, IIT, GIST)

Geonhyup Lee (MS course, IIT, GIST)

Yongwoo Lee (Undergrad course, GIST)

Jinseo Hong (Undergrad course, GIST)

TurtleBot3
Waffle Pi



TURTLEBOT3

GIST

CONTENT



Overview



- Turtlebot 3 with Open Manipulator

Onboard sensors : 3D gyroscope, 2D LiDAR

Open manipulator : 4 DoF RRRR robotic arm.

Reference : <http://www.robotis.com/>

- Overhead 2D camera

RGB color detect

Methodology



Detection & Navigation

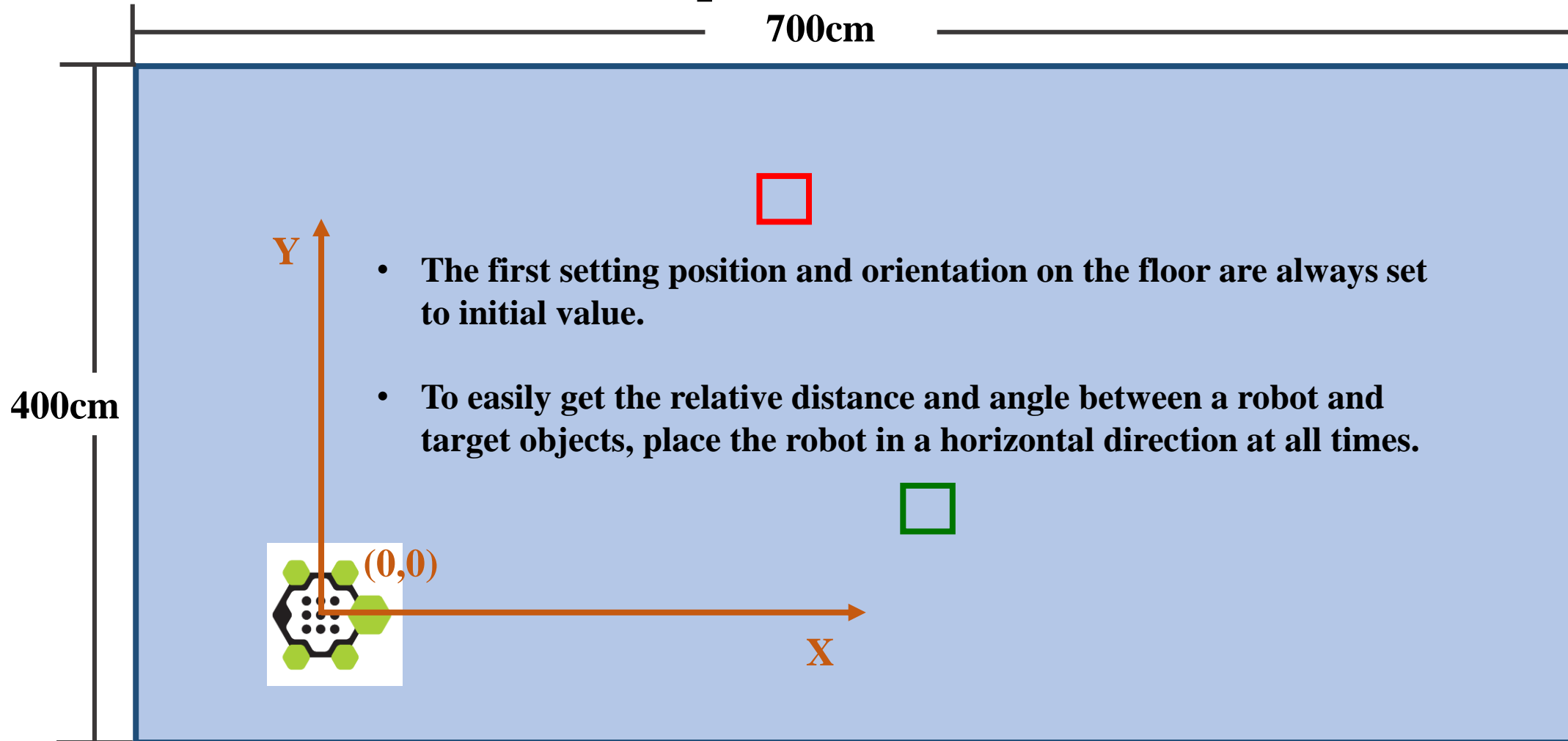
- Past - roughly find location of robot and other object with overhead 2D camera & precise task(manipulate, localization of object) with 3D depth sensor
- But there was a problem to associate 3D depth sensor with ROS and it was hard to solve in a month. So we changed our strategy to simplify the task
- Now – Find location as precisely as we can, keep calibrating robot's path and line up robot and object with 2D overhead camera. After alignment

Manipulation


- First we tried to use a package given in ROS. But it also had a problem about connection, we solved inverse kinematics of Open Manipulator by hand.
- The pick & drop task is performed by inverse kinematic method with known position data of objects.

Results – Task 1

Set initial position & orientation



Results – Task 2



▪ Steps of Object Detection

We used Overhead USB camera and OpenCV.

1. Restrict the area of detection

(To prevent the effect of edge sides.)

2. Detect the objects first and remember each object's position

(Using Color filter in OPENCV)

3. Detect the turtlebot's position continuously and calculate distances from objects to turtlebot.

Results – Task 2

The screenshot displays a Linux desktop environment. On the left, a vertical dock contains various application icons. The main workspace features a terminal window on the right and a recording application window on the left. The recording application window has a red rectangle highlighting its settings, including 'Enable recording hotkey', 'Hotkey: Ctrl +', 'Total time: 0:00:04', 'Preview frame rate: 10', and 'Start preview' button. The terminal window shows a list of coordinates for 'black position' and a command prompt at the bottom.

Recording Application Settings:

- Enable recording hotkey: ☒ Enable sound notifications: ☐
- Hotkey: ☒ Ctrl + ☐ Shift + ☐ Alt + ☐ Super + R
- Total time: 0:00:04
- Preview frame rate: 10
- Note: Previewing requires extra CPU time (especially at high frame rates).
- Size in: 1366x768
- Size out: 1366x768
- File name: simpl...9.mp4
- File size: 241 KiB
- Bit rate: 46 kbit/s
- Start preview

Terminal Output:

```
black position is133.667 125.184
black position is133.705 125.156
black position is133.799 125.211
black position is133.721 125.269
black position is133.749 125.222
black position is133.727 125.16
black position is133.74 125.227
black position is133.746 125.215
black position is133.71 125.235
black position is133.708 125.187
black position is133.68 125.251
black position is133.788 125.257
black position is133.628 125.25
black position is133.736 125.246
black position is133.667 125.241
black position is133.771 125.183
black position is133.614 125.237
black position is133.782 125.198
black position is133.646 125.285
black position is133.748 125.233
black position is133.638 125.195
black position is133.724 125.239
black position is133.728 125.196
black position is133.66 125.215
black position is133.692 125.245
black position is133.716 125.279
black position is133.662 125.184
black position is133.688 125.215
black position is133.701 125.194
black position is133.689 125.219
black position is133.677 125.279
black position is133.725 125.265
black position is133.785 125.22
black position is133.715 125.209
black position is133.731 125.209
black position is133.675 125.198
black position is133.651 125.208
black position is133.692 125.255
black position is133.775 125.27
^C
woody@woody-N450-PE50K:~/Desktop$ ./multicolor
```

Results – Task 2

▪ Detectable objects

It was easy to detect color objects, but it was difficult to detect white and transparent objects.

Before competition, we can easily calculate $100 \text{ pixel} = 1.15\text{m}$, so we can convert the scale from pixel to meter.

Then we send meter information to Turtlebot.

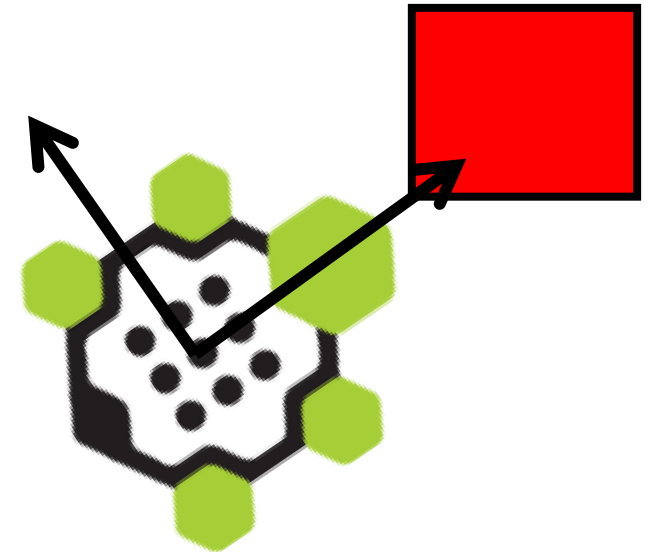
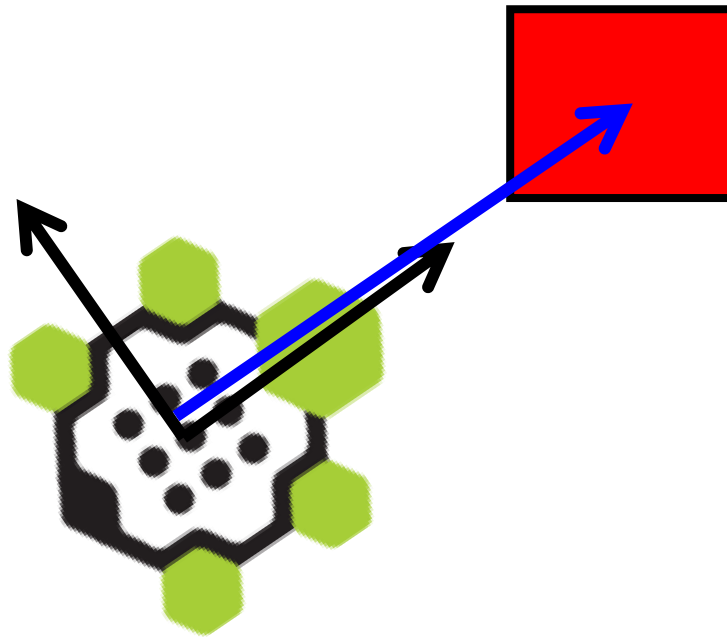
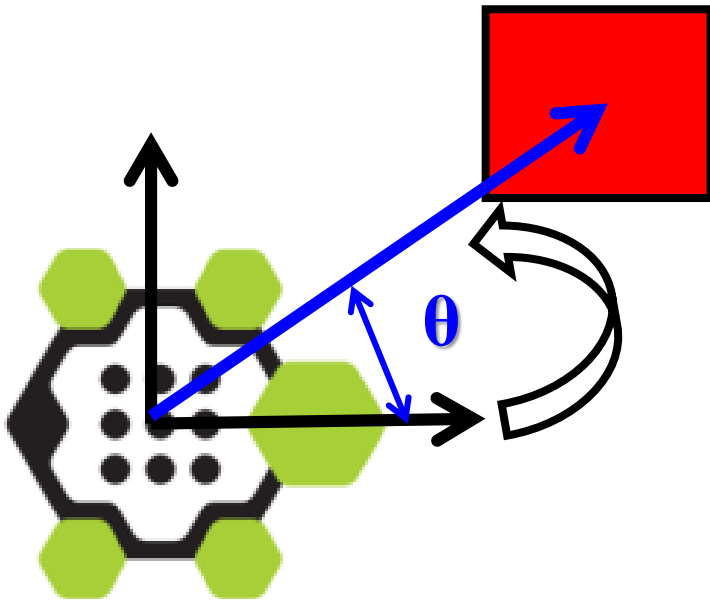
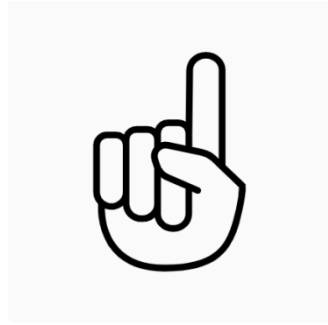
We tried to use Canny Edge Detector in OpenCV, but it has problems.

(1) Camera was too far from objects. Sometimes objects have no difference with other noises.

(2) Square blocks were also detected as circles. It seems to happen because of distance from camera.

However, it might be possible if we spend some time to find best edge thresholds.

Results – Task 3



Results – Task 3

- If x-axis matches with the direction of object, Turtlebot can reach object using only x-direction moving.
=> We match the x-axis with direction of object by rotating yaw-axis.

- From the rostopic `open_manipulator_with_tb3_tools/odometry`, we can read odometry of turtlebot. However, it is quaternion based orientation expression. Thus, we need to convert Quaternion to Euler. We can calculate y-axis tilt using x,y,z,w in Quaternion.

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \text{asin}(2(q_0 q_2 - q_3 q_1)) \\ \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

```
// yaw (z-axis rotation)
double siny_cosp = +2.0 * (q.w() * q.z() + q.x() * q.y());
double cosy_cosp = +1.0 - 2.0 * (q.y() * q.y() + q.z() * q.z());
yaw = atan2(siny_cosp, cosy_cosp);
```

Results – Task 3

1. Rotate until matching Turtlebot's face with object direction

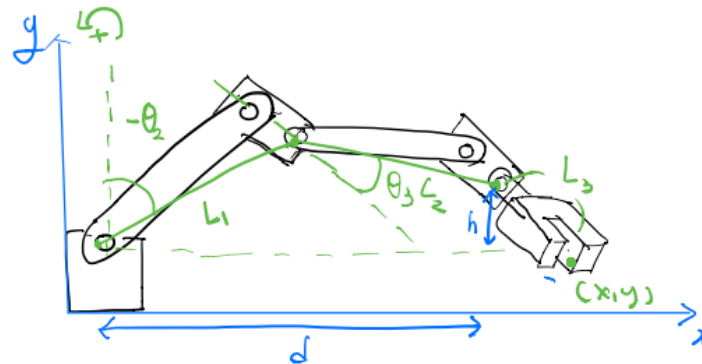
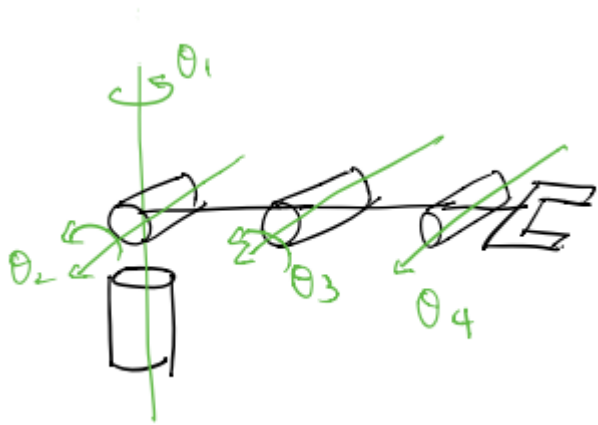


2. Angle compensation while reaching the object
3. Angle compensation while reaching trashbin.



Results – Task 4

Arm Manipulation 수식



$$d^2 + h^2 = L_1^2 + L_2^2 + 2L_1L_2 \cos\left(\frac{\pi}{2} + \theta_3\right)$$

$$\theta_3 = \sin^{-1} \frac{L_1^2 + L_2^2 - d^2 - h^2}{2L_1L_2}$$

$$-\theta_2 = \frac{\pi}{2} - \left(\cos^{-1} \frac{L_1 + L_2 \cos(\frac{\pi}{2} - \theta_3)}{\sqrt{d^2 + h^2}} + \tan^{-1} \frac{h}{d} \right)$$

$$\theta_2 = \cos^{-1} \frac{L_1 + L_2 \cos(\frac{\pi}{2} - \theta_3)}{\sqrt{d^2 + h^2}} + \tan^{-1} \frac{h}{d} - \frac{\pi}{2}$$

$$d = x - L_3 \cos \theta_3, \quad h = y_0 - L_3 \sin \theta_3$$

2-Dimension equation 사용

총 4개의 joint 중 실제로 제어하는 joint의 개수는 3개로 2개의 joint로 position에 도달하도록 inverse kinematic을 구현하였다.

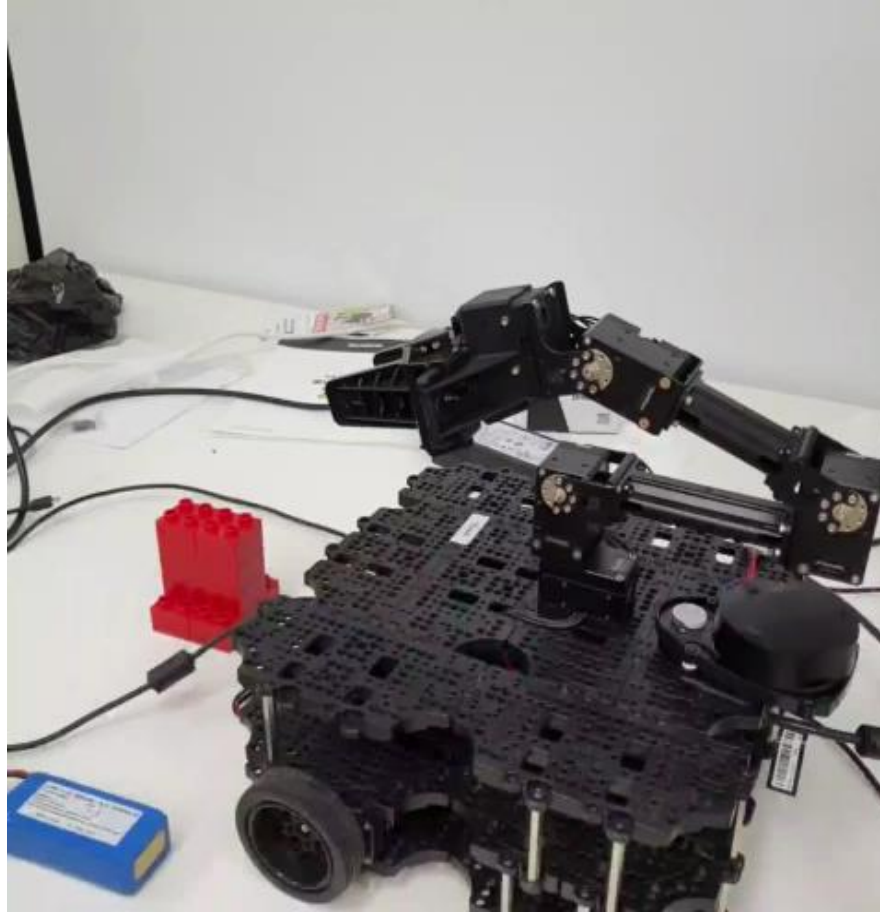
Using 2-Dimension equation

Only three of four joint is actually controlled. Two joints for position and the other is for picking angle.

So, we made inverse kinematic about two joint for position.

Results – Task 4

Arm Manipulation 영상



The image is a collage of six terminal screenshots arranged in a 3x2 grid, showing ROS (Robot Operating System) commands and outputs for different robotic applications. Each screenshot has a red border and a title bar.

- Top Left:** Terminal window titled "roscore http://192.168.0.119:11311/". It shows the command `is <1GB.` and the output of `roscomm version 1.12.14`. The title "roscore" is overlaid in the center.
- Top Right:** Terminal window titled "lee@lee-Inspiron-7386: ~ 49x28". It shows a series of `rostopic pub` commands for a topic named `operation`, publishing `std_msgs/Float32MultiArray` messages with data like `[250,250]`. The title "Manipulator-subscriber operation" is overlaid in the center.
- Middle Left:** Terminal window titled "lee@lee-Inspiron-7386: ~ 52x27". It shows the command `roslaunch open_manipulator_1 om_with_tb3.launch` and the output of `process[om_with_tb3/robot_state_publisher-1]: started with pid [14589]`. The title "Camera Navigation-publisher" is overlaid in the center.
- Middle Right:** Terminal window titled "lee@lee-Inspiron-7386: ~ 101x27". It shows the command `roslaunch open_manipulator_1 om_with_tb3.launch` and the output of `process[om_with_tb3/robot_state_publisher-1]: started with pid [14589]`. The title "Manipulator-publisher" is overlaid in the center.
- Bottom Left:** Terminal window titled "lee@lee-Inspiron-7386: ~ 49x27". It shows the command `roslaunch open_manipulator_1 om_with_tb3.launch` and the output of `process[om_with_tb3/robot_state_publisher-1]: started with pid [14589]`. The title "Rviz-moveit" is overlaid in the center.
- Bottom Right:** Terminal window titled "lee@lee-Inspiron-7386: ~ 101x28". It shows the command `roslaunch open_manipulator_1 om_with_tb3.launch` and the output of `process[om_with_tb3/robot_state_publisher-1]: started with pid [14589]`. The title "Test for Arm-manipulation" is overlaid in the center.

Appendix

Software

- Thank you -

