ELEC6910R - Robotic Perception and Learning Course Project

WINATA Genta Indra (20378324), LIN Zhaojiang (20440751)

1 Project Overview

Our project video is on YouTube https://youtu.be/n-XK24o42Oo

1. Build 2D grid map by using laserscan data and show it via rviz
2. Control the mobile robot in the simulation environment with keyboard (drive it to move)
3. Image Recognition and Localization. There are five images of different people in the environment and we need to do the following:
   - judge whether the target images occurred in current vision data
   - if yes, estimate the location of target images
   - add markers to the map in rivz which stands for the target images position
4. Visual Servoing. There is a slowly moving yellow (rgb:255,255,0) ball in the environment and we need to write program (rosnode, in c/c++ or python) to control the mobile robot to follow the ball
5. The room is divided into several areas, let the robot judge which area it locates
6. Write a launch file to roslaunch all of above programs at once
7. **Additional work:** Automatic self-exploration script

2 Environment Setup

Our project requires several libraries and packages as follows:

- ROS kinetic, see http://wiki.ros.org/kinetic/Installation/Ubuntu
- V-REP 3.5.0
- OpenCV
- Hector slam library

In this project, we use **Ubuntu 16.04 LTS** operating system. These are the instructions to prepare the workspace environment:

```
# install ROS kinetic from http://wiki.ros.org/kinetic/Installation/Ubuntu
# install V-REP 3.5.0 from http://www.coppeliarobotics.com/downloads.html
```

Prepare the workspace

```
# To prepare the workspace
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Additional libraries

```
# install hector-slam
sudo apt install ros-kinetic-hector-slam
```

3 ROS graph

Figure 1 shows the ROS graph generated from `rqt_graph` tool.

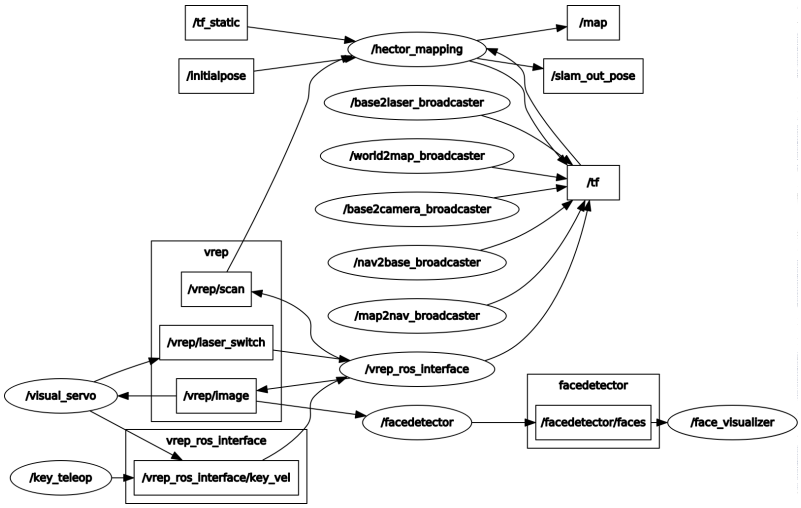

Figure 1: ROS graph

4 Task 1: Build 2D grid map

To build 2D grid map, we use hector slam package and **rviz**. Hector slam library includes the hector mapping, a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform or both). The library receive 2D laser data and output a grid map. We use Rviz to visualize the map. Figure 2 shows the example in Rviz, and Figure 3 shows the expected finished mapping.

To launch the node, we use the following script:

```
<node pkg="rviz" type="rviz" name="rviz"
args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>
```

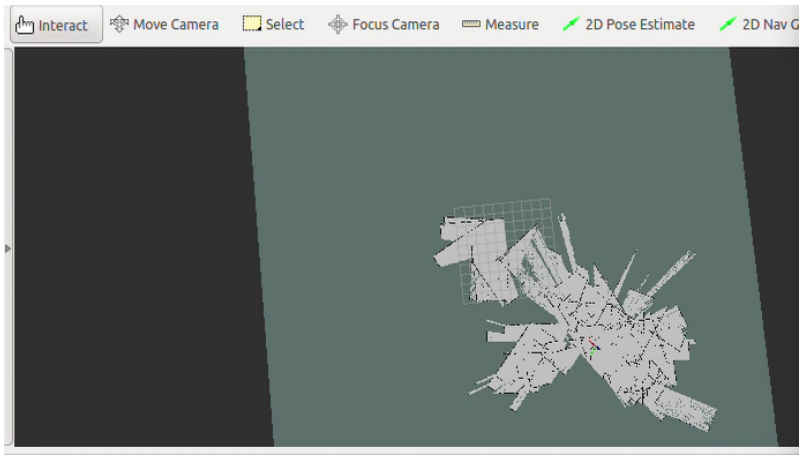

Figure 2: Mapping Sample using Rviz and SLAM

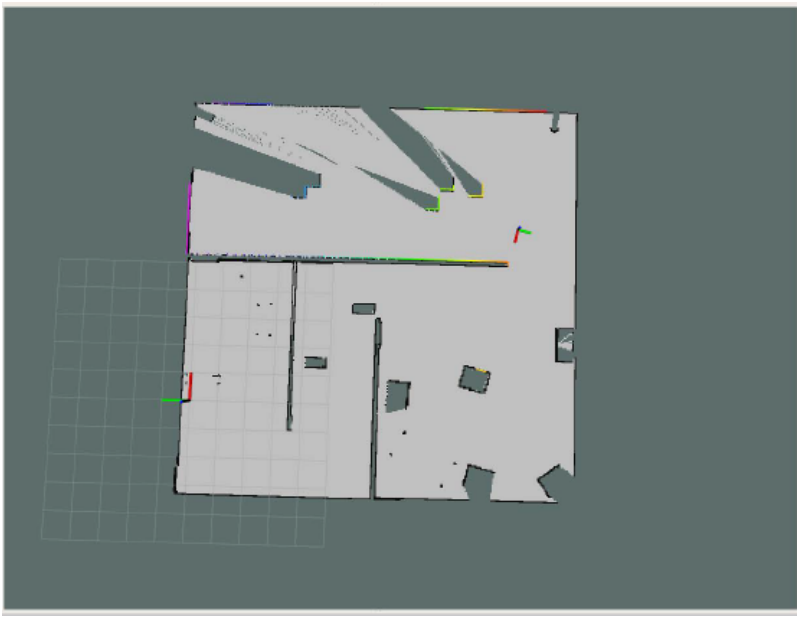

Figure 3: Completed Mapping using Rviz and SLAM

5 Task 2: Control Robot by Keyboard

We extend the `key_teleop` source code to control the robot by publishing the velocity command (`/vrep_ros_interface/key_vel`) to control the robot in V-REP simulation environment. We use keyboard arrows as our cursors, and alter the forward and backward speed. The script is implemented using Python under **project** node. The command to run the script:

```
rosrun project key_teleop.py
```

Figure 4 shows the keyboard interface.

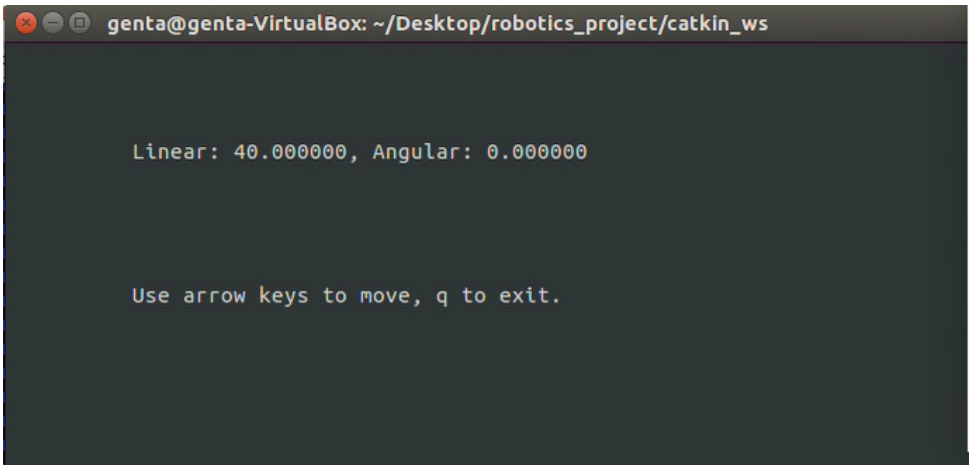

Figure 4: Keyboard Control Interface

6 Task 3: Image Recognition and Localization

For image recognition part, we use OpenCV library.

- During preprocessing stage, we use `self.detectorCascade.detectMultiScale function` to extract features from train images.

- During training stage, we use the features to train a model for face recognition (`cv2.face.createLBPHFaceRecognizer()`).
- During test stage, the camera sensor captures image. `self.detectorCascade.detectMultiScale function` detects the object in the image. Then, the face recognizer module will output the prediction in the new window.

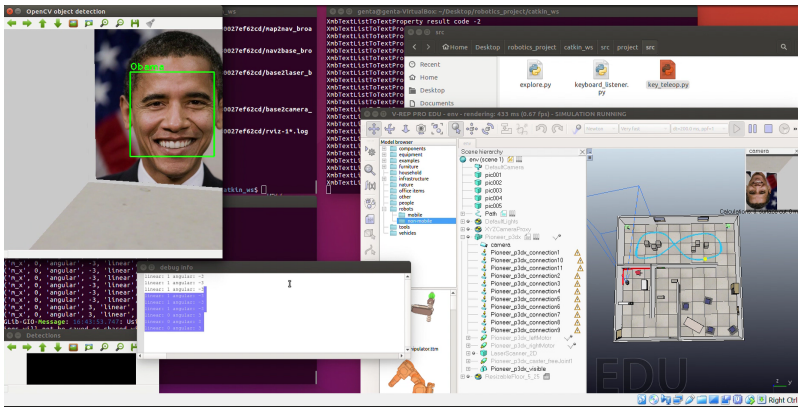

Figure 5: Face detection

7 Task 4: Visual Servoing

In this task, the machine need to track and follow the yellow ball. We set `#FFFF00` color for the ball. We propose following steps for the ball tracking:

1. If the ball is not within the distance of the visual sensor, the robot will move and explore the map.
2. Else, we calculate the ratio r of the area of the visible ball with the area of the entire window.

$$r = (h \times w)/(512 \times 512)$$

where h is the height of the visible ball, and w is the width of the visible ball relative to the window. Then, we decide the velocity and angular according to r.

```
angular = float(m_x - 256)*(0.7)/256
if r < 0.05:
    linear = 0.6
    if angular > 0:
        angular = 0.1
    else:
        angular = -0.1
elif r > 0.1:
    linear = -2
```

The reasons of putting these numbers are three-fold: (1) When the ball is inside the screen and relatively close to the robot (approx. 0.1), we will keep larger distance between the robot and ball to make sure we don't lose the ball. (2) When the ball is bit far from the robot (approx. 0.05), the robot need to get closer to the ball. (3) To optimize the tracking, we adjust the robot's orientation to the center of the ball. Figure 6 shows the screenshot of visual servoing. A red rectangle is drawn to locate the yellow ball.

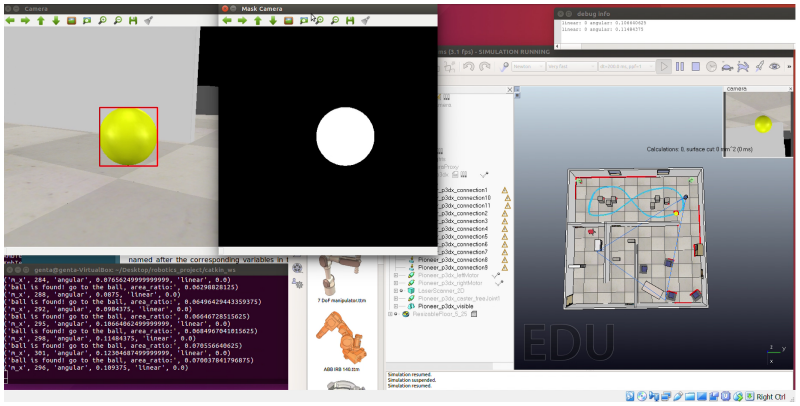

Figure 6: Visual Servoing

8 Task 5: Robot Localization

In this task, we try to identify the room information (`A, B, C, D`) where the robot locates. The room information can be easily identified if the location is known. The location of the robot is calculated by comparing the observation from the sensors with the constructed map. After the location is obtained, we run a simple set of rules, to classify the room information given the location of the robot. Figure 7 shows the flow chart diagram and Figure 8 illustrates the divided areas.

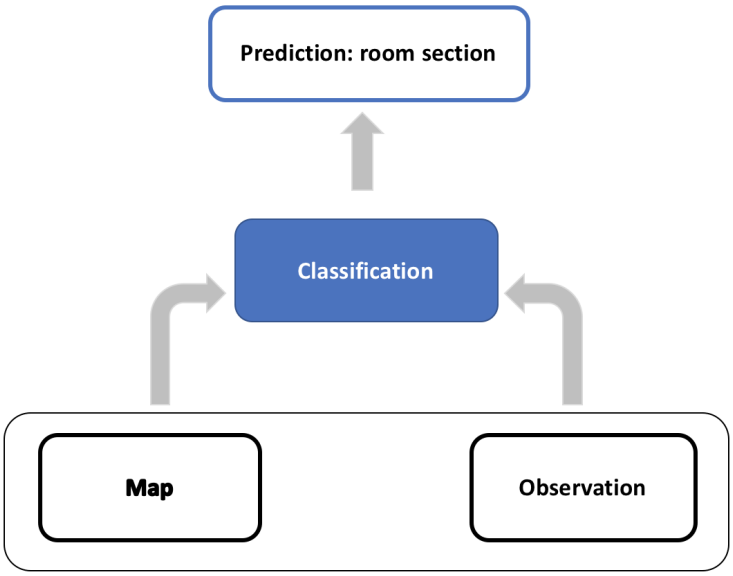

Figure 7: localization diagram

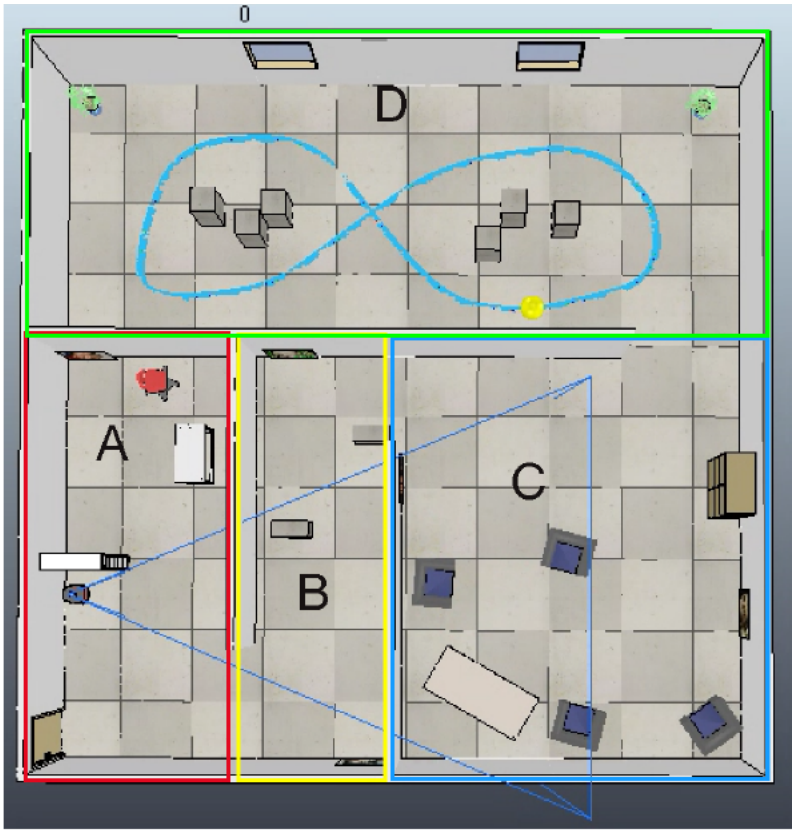

Figure 8: Room information

9 Task 6: Roslaunch Script

We create a launch file to execute all scripts at once. We include our script in this document.

```
<?xml version="1.0"?>
<launch>
<include file="$(find opencv_detector)/launch/facedetector.launch">
  <arg name="image_topic" value="/vrep/image" />
</include>
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find hector_slam_launch)/rviz_cfg/mapping_de
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_link"/>
  <arg name="odom_frame" default="nav"/>
  <arg name="laser_frame" default="laser_link"/>
  <arg name="camera_frame" default="camera_link"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="vrep/scan"/>
  <arg name="map_size" default="1024"/>
  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />
    <param name="odom_frame" value="$(arg laser_frame)" />
    <param name="odom_frame" value="$(arg camera_frame)" />
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>
    <param name="map_resolution" value="0.050"/>
```

```
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />
    <param name="advertise_map_service" value="true"/>
    <param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>
    <param name="tf_map_scanmatch_transform_frame_name" value="$(arg tf_map_scanmatch_transform_fr
  </node>
  <node pkg="tf" type="static_transform_publisher" name="world2map_broadcaster" args="0 0 0 0 0 0
  <node pkg="tf" type="static_transform_publisher" name="map2nav_broadcaster" args="0 0 0 0 0 0 ma
  <node pkg="tf" type="static_transform_publisher" name="nav2base_broadcaster" args="0 0 0 0 0 0 n
  <node pkg="tf" type="static_transform_publisher" name="base2laser_broadcaster" args="0 0 0 0 0 0
  <node pkg="tf" type="static_transform_publisher" name="base2camera_broadcaster" args="0 0 0 0 0
  <node pkg="visual_servo" type="visual_servo.py" name="visual_servo" output="screen"/>
  <node pkg="detection_msgs" type="patch_visualizer" name="face_visualizer" output="screen">
    <remap from="camera" to="/vrep/image" />
    <remap from="detections" to="/facedetector/faces" />
    <param name="cleanup_delay" value="0" />
  </node>
</launch>
```

10 Additional work

In addition, we created an automatic self-exploration script. Our robot will explore the map by either going forward or backward with some probability.

```
#!/usr/bin/env python
import rospy
import random
from std_msgs.msg import String, Bool, Float32
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist
import numpy as np

if __name__ == '__main__':
    rospy.init_node('visual_servo')
    while(True):
    if random.random() < 0.7:
        angular = 3
            linear = 0
        else:
        angular = -3
            linear = 1
    twist = Twist()

    twist.linear.x = linear
        twist.angular.z = angular
    self._pub_cmd.publish(twist)
    rospy.spin()
```