# SLAM System based on V-REP and ROS

Su Dan, Jehiel D. Santos, Pranay Sood

The Hong Kong University of Science and Technology
Department of Electronic and Computer Engineering

## I. Introduction

Simultaneous localization and mapping, simply known as SLAM, is the process of creating a map of an unknown environment by using the information from the sensors while keeping track of the location of the robot at the same time. The problem of knowing the location requires knowledge of the map to infer about the location of the robot with respect to its surroundings. On the other hand, the problem of building and updating the map requires knowledge of its location with respect to its surroundings. Hence, SLAM is considered as a chicken-and-egg problem since the two output depend on each other simultaneously. This approach finds its purpose in various application in navigation, building maps, augmented reality, and other essential applications in robotics.

## II. Main Objectives

The main task for this project is to build a SLAM system based on Virtual Robot Experimentation Platform (V-REP) and Robotic Operating System (ROS). The environment in V-REP is already provided as shown in Figure 1. The robot is equipped with a 2D laser scanner and a vision sensor as an input to be processed by the system.

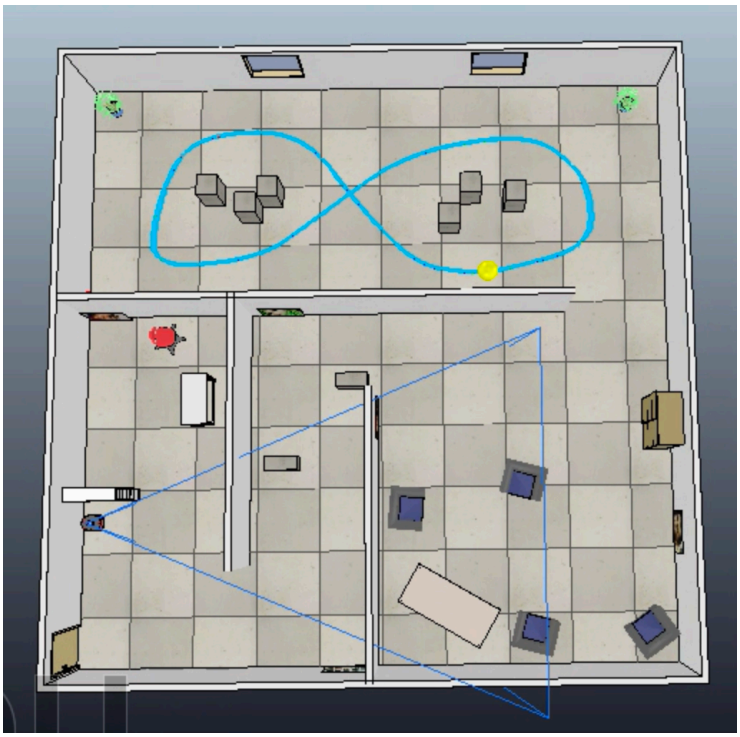
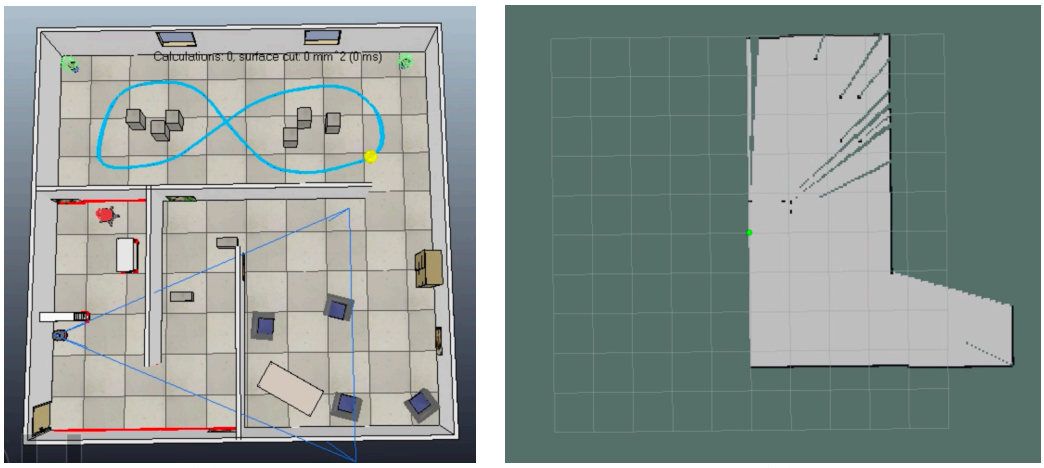


Figure 1. Simulation Environment in V-REP

The main objective is divided into five (5) different subtasks which are (a) to build a 2D grid map of the surroundings by using the data from the laser scan by using ROS Visualization (`RViz`), (b) move the robot in the simulation environment by using the keyboard, (c) recognize the images present in the simulation environment and indicate their location by adding a marker in the `RViz` environment, (d) follow the yellow ball once detected, and (e) write a launch file to run the programs as indicated at the same time.

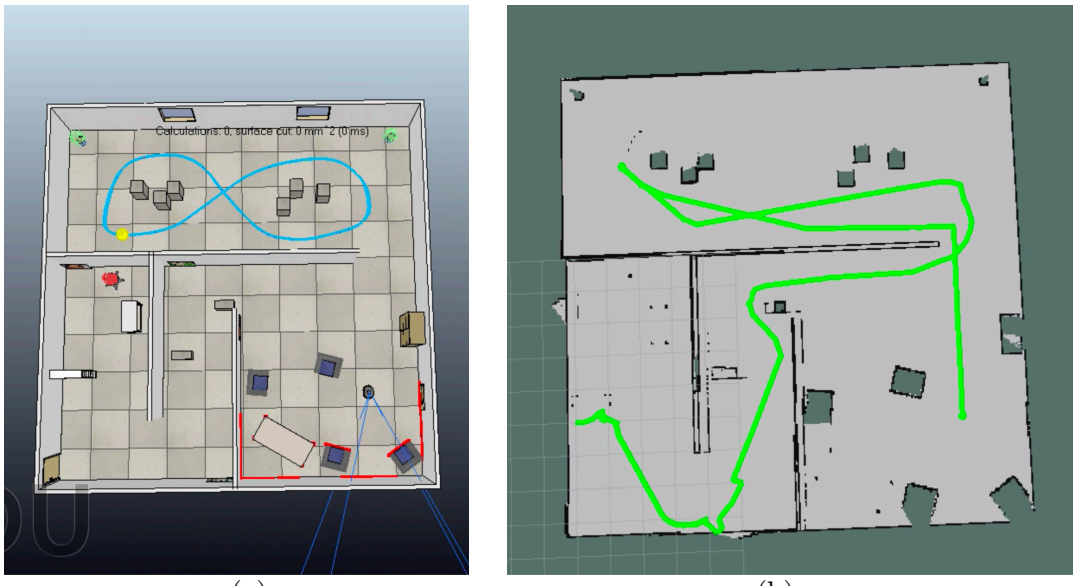
A. First Task: Build a 2D Grid Map of the Surroundings

The first task is to build a 2D grid map of the surroundings. Initially, the map created in `RViz` is only a portion of the entire map as shown in Figure 2. As soon as the keyboard is used to control the robot, the map is being updated simultaneously with the location of the robot. This is made possible by using the existing package called `hector_slam` [1]. A green path marker is placed to show the movement of the robot. The complete version of the map is shown in Figure 3.



(a) (b)

Figure 2. 2D grid map at initialization



(a) (b)

Figure 3. 2D grid map after moving around the simulation environment

B. Second Task: Move the Robot

This task is done in connection with the previous task in which the robot is to be controlled with the following keys as shown in Figure 4, in which the linear and angular speed can also be varied. This is accomplished by using the packages `teleop_twist_keyboard` and `teleop_tools` which are available in repositories [2, 3]. This package publishes its output on the `/vrep/cmd_vel` topic for the direction and velocity of the robot.

```
Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit
```

Figure 4. Control setting to move the robot from keyboard

C. Third Task: Recognize and Localize the Images

The next task is to localize and recognize the images. To recognize the images, the Fast Library for Approximate Nearest Neighbors (FLANN) based matcher is used in which the features of one image will be compared to the features of the other images [4]. Since the number of images to be recognized is limited, i.e. five images, this procedure can still be done efficiently. In Figure 5, the complete system architecture is illustrated in which the scale invariant feature transform (SIFT) descriptors are to be extracted and matched with the input frames from the training images by employing FLANN [5, 6].

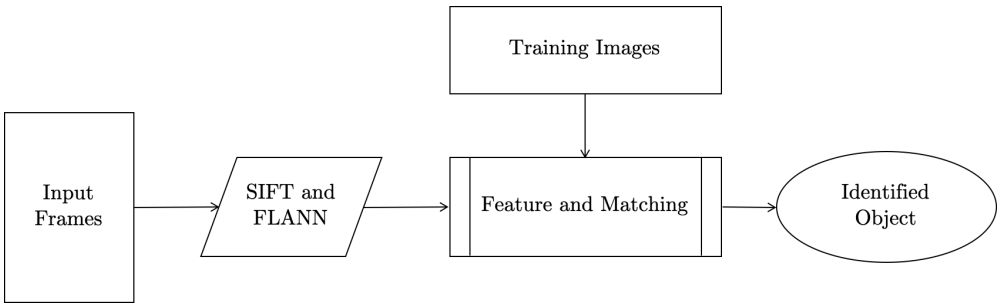



Figure 5. System architecture for image recognition [5]

The Euclidean distance is obtained as seen in Figure 6 between the features of the images and classified using k-Nearest Neighbors (KNN). Once the image is identified, the information is to be published in `/visualization_marker` to localize the image as shown in Figure 7.

(a) (b)

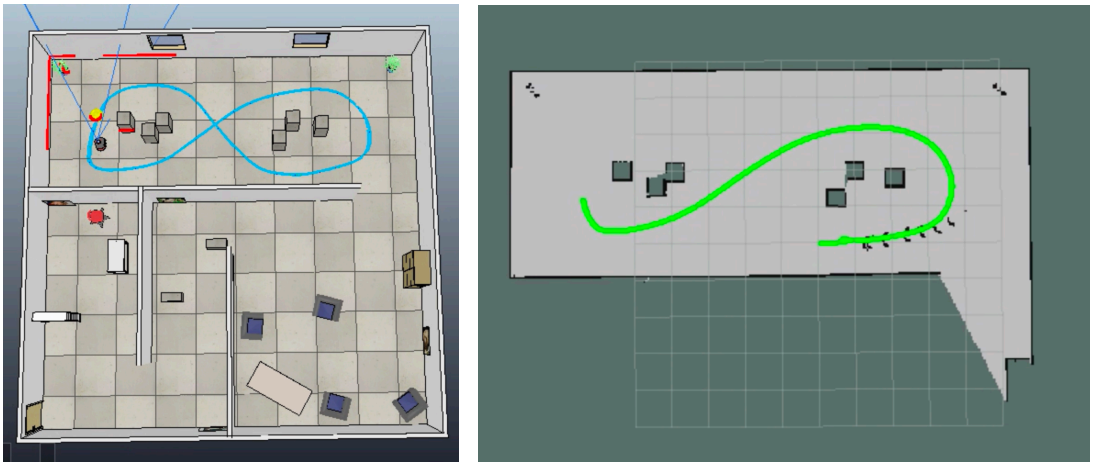
Figure 6. Image recognition by extracting SIFT descriptors


Figure 7. Image recognition and localization in `RViz`

D. Fourth Task: Follow the Ball

The task to follow the ball is simplified due to the color setting of the ball which is `RGB:#FFFF00`. An illustration is shown in Figure 8(a) in which the robot is following the ball. The track is being published in `RViz` as shown in Figure 8(b). The ball is detected by color segmentation in which once the color yellow is detected in the input frame, the robot will recognize the presence of the ball. Then, masking technique will be used to remove the background of the region of interest (in our case, the ball) as shown in Figure 9. Finally, once the ball is detected, a simple proportional-integral-derivative (PID) controller in python is used to follow the ball [7]. The complete track is shown in Figure 10.

Specifically, masking is done by converting the frame from BGR to HSV color space and creating a mask which comprises the object in yellow color. Then a bitwise operation is done such that the yellow colored object is highlighted and stored. The contour of the circle is identified in the image and the center and radius of the ball is calculated. The error with the radius and the center of the ball will determine the vertical and horizontal distance of the robot from the ball. This error will be the input to PID to control the movement of the robot.



(a) (b)

Figure 8. Robot following the yellow ball: (a) simulation environment (b) track in `RViz`

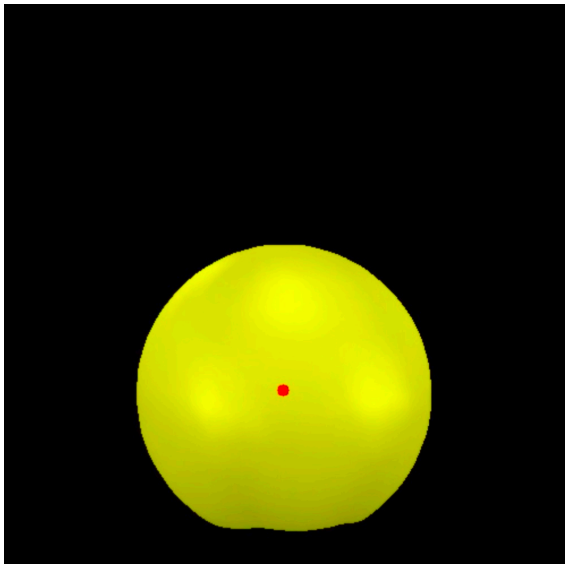


Figure 9. Masking technique

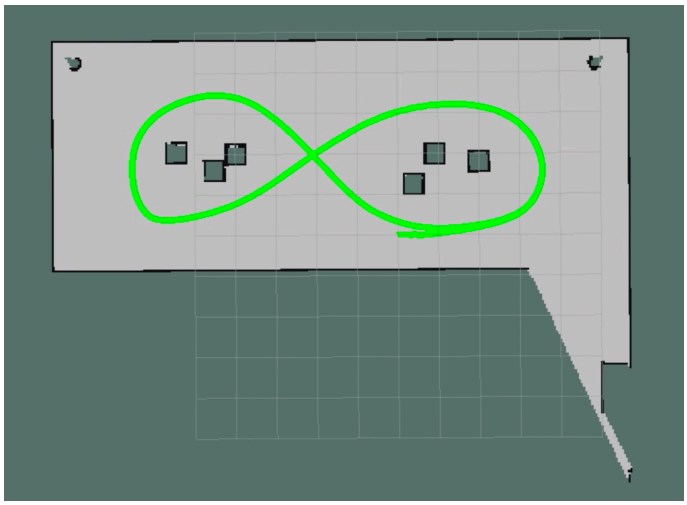


Figure 10. Complete track

E. Fifth Task: Compilation of all the Tasks

The nodes and topics of the entire program is shown in Figure 11. This shows the dependencies of each nodes by the predefined topics. This is displayed using the `rqt_graph` function present in ROS. The launch file is labeled as `final.launch` and can be executed by using `roslaunch final.launch` in the launch folder of catkin.

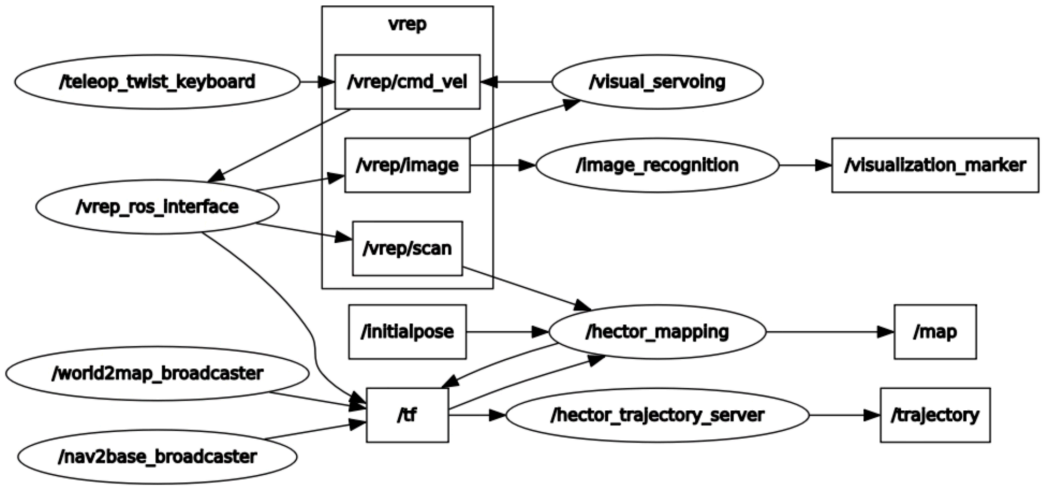



Figure 11. The nodes and topics in ROS

The boxes contain the topics and the ellipses contain the nodes. To start with, the `/vrep_ros_interface` provides the interface between the VREP and ROS and publishes information on `/vrep/image` and `/vrep/scan` which are the data for the images and the laser scan for the map respectively. It is also subscribed to the `/vrep/cmd_vel` for the movements of the robot, e.g. direction and velocity. The `/image_recognition` node subscribes from the `/vrep/image` to obtain the images from VREP-ROS and publishes a marker to `/visualization_marker` whenever an image is being recognized. The node `/visual_servoing` is the one responsible for following the yellow ball. It subscribes from the `/vrep/image` for the images and publishes to `/vrep/cmd_vel` to set the velocity and follow the ball by using PID controller. The node `/hector_mapping` is the one responsible for the creation of the map by subscribing from the laser scan data and publishing the output to `/map`. The rest of the nodes are part of the `hector_slam` package which are necessary to build the map.

## III. Conclusion

In conclusion, using these above-mentioned packages and the knowledge of ROS and V-REP, it has made it possible for us to perform all the tasks and objectives that were required. As a result, it was possible to perform navigation of the robot, image recognition and localization, construction of a 2-D grid map and finally, allow the robot to follow the movement the ball. Despite being a challenging task for us, with preparation from the ROS tutorials and various online references, it was possible for us to accomplish the objectives as required by the task. In addition, this project allowed us to garner invaluable knowledge in Ubuntu (Linux OS), ROS, V-REP and Python programming language.

## IV. References and Repositories

[1] "Hector Slam," https://github.com/tu-darmstadt-ros-pkg/hector_slam. Accessed: 2019-05-11

[2] "Teleop Twist Keyboard," https://github.com/ros-teleop/teleop_twist_keyboard. Accessed: 2019-05-11

[3] "Teleop Tools," https://github.com/ros-teleop/teleop_tools. Accessed.: 2019-05-11

[4] "FLANN based Matcher," https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html. Accessed: 2019-05-11

[5] R. Megalingam, G. Sriteja, A. Kashyap, A. S., V. Gedala, S. Badhyopadhyay, "Performance Evaluation of SIFT & FLANN and HAAR Cascade Image Processing Algorithms for Object Identification in Robotic Applications," International Journal of Pure and Applied Mathematics, vol. 118, no. 18, 2018.

[6] V. A, D. Hebbar, V. Shekhar, K. Murthy, S. Natarajan, "Two Novel Detector-Descriptor Based Approaches for Face Recognition using SIFT and SURF," 4th International Conference on Eco-friendly Computing and Communication Systems, ICECCS 2015, 2015.

[7] "Simple PID," https://pypi.org/project/simple-pid/. Accessed: 2019-05-11