CMP9767M ROBOT PROGRAMMING ASSESSMENT 2

To detect and track the grape bunches in the provided training arenas which is simulated in the Gazebo vineyard by driving the robot autonomously in the ROS Software.

SADANAND VITHAL GIRADDI
Student ID - 25895786

Table of Contents

To detect and track the grape bunches in the provided training arenas which is simulated in the Gazebo vineyard by driving the robot autonomously in the ROS Software.

1. ABSTRACT

The main work is to implement the software system for an agricultural robot provided in the training arenas which is simulated in Gazebo vineyard to estimate the crop yield of a field. The robot navigates using the topological navigation method autonomously avoiding any obstacles through the waypoints by detecting with robot camera sensors such as RGBD camera, laser scanners and count all grape bunches present in the frames of the vineyard map using OpenCV python script.

2. INTRODUCTION

The ROS is a framework for writing the robot software in which it has a collection of tools and libraries which aims to simplify the complex tasks on the robotic platforms. The different training arenas will be provided in the form of gazebo simulations and the actual environment will be very similar to the training arenas which includes the same type of plants and general layout, but slightly the number position of the plants is different to test our solution. And this was a relatively a challenging task to drive the robot autonomously to detect and track the grape bunches in the provided vineyard map [1].

2.1 HISTORY

It is originated by graduate student at Stanford Artificial Intelligence Lab (2007). It is taken up and developed by Willow Garage, as it was a research lab used in developing the hardware and open-source software for robotics application and extended the concepts further more for research and done well testing implementations. Furthermore, it is supported by the Open-Source Robotics Foundation (OSRF) (2013) and gradually became widely used in the robotics research community. The ROS system now consists of more than lakh of users worldwide working in the domains for building the automation systems [1].

3. NAVIGATION IN ROS.

The ROS navigation stack is the set of collection of the software packages that helps the robot to move from the starting location to the goal point safely. Navigation stack setup uses two costmaps in order to store the information about the obstacles. The one costmap is used as global planning for creating the long-term plans over the entire environment and the other costmap is used for local planning and obstacle avoidance. And in local planner uses the local information which is obtained from the sensors to plan its path. And the Fig 1 is taken from the classroom ROS programming slides [2].

`move_base`

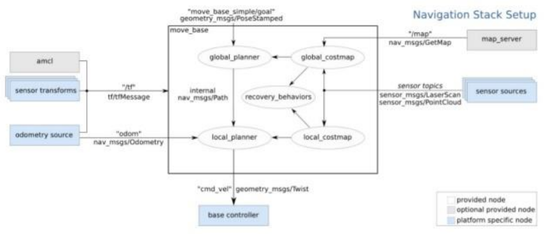


Fig 1: Navigation stack setup

ROS node move_base package provides the implementation of an action that given the goal which attempts to reach its with its mobile base. The move_base node links the global and local planner for its global navigation task.

3.1 SLAM – MAP BUILDING AND NAVIGATION.

The SLAM is the technique in which the robot generates the map of an unknown environment (Mapping) and simultaneously keep track of its position within the map (Localization). The term SLAM stands for Simultaneous Localization and Mapping. IT consists of different parts such as state estimation, landmark extraction. It is widely used in the robotics and it is the central to a range of indoor and outdoor applications. It is also applicable for both 2D and 3D motion [3]. In this assessment the SLAM uses the topological navigation method instead of metric navigation method and launched thorvald gazebo simulation- vineyard _small map.

3.1.1 TOPOLOGCAL NAVIGATION.

The topological navigation has two types environment, one is the topological representation based upon the movements and the other is the topological representation based on the geometrical maps [4]. In this assessment the navigation is based upon the movements. To count the number of grape bunches in the vineyard map the robot has been navigated through the directions of the waypoints created between the nodes of each frames in the map. And this map is loaded in the rviz using the launch file topo_nav.launch

3.1.2 POSES,POSITIONS,ORIENTATIONS.

The agricultural robot will have a bunch of subsystems, such as the cameras, laser scanners attached to the robot to allow it to navigate through the world. The front camera, left camera and the right cameras were attached to the robot in such a way to find the items to detect and track it them while navigating autonomously. In a position, there are three numbers (x, y, z) which indicate the extent to which the coordinates translate to each axis and in an orientation, there are three numbers (roll, pitch, yaw) to indicate the degree of rotation about each axis [1]. waypoints are created by knowing the pose of the robot in the gazebo world and adjusting it with exact positions by creating the waypoints for the robot to navigate autonomously in the specified path (created waypoints).

To make the pose estimate better in Gazebo vineyard simulation, we should drive the robot around a bit in front of the plants which are held in the particular frames for getting better pose coordinates for creating the waypoints for robot navigation before doing any autonomous navigation.

3.1.3 WAYPOINTS CREATING

The robot is navigated using the topological navigation by creating the waypoints in amcl map and with the help of rqt_image_view the exact waypoints were created by seeing the frames in which the grape bunches were stored in the vineyard_small map.

Waypoints can be added using the ROS service call and first node will be created in the yaml file by giving appropriate x and y coordinates.

In rviz, the waypoints are added using the ROS service call and the first node should be created in yaml map file and later map file should be added to the folder. This can be done by using the specific code:

rosrun topological_utils load_yaml_map.py $(rospack find uol_cmp9767m_tutorial) /maps/new_test.yaml

I have created the new_test.yaml file and created the waypoints in that amcl file map.

The first waypoint co-ordinates:

```
rosservice call
/topological_map_manager/add_topological_n
ode "
name: WayPoint1
pose:
  position:
    x: -11.0
    y: -4.5
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0
add_close_nodes: false"
```

The above code is used to create the waypoints in the map [5].

The below code is used for creating the waypoints in each node [5].

```
rosservice call
/topological_map_manager/add_edges_betwee
n_nodes "
origin: WayPoint0
destination: WayPoint1
action: move_base
edge_id: WayPoint0_WayPoint1"
```

And where the action move_base is used to move the robot from one destination to the

other. The uni-directional edge between the nodes has been added and even bi-directional edges can also be added for the robot to come back or navigate to the other waypoint using bi-directional point.
The Fig 2: shows about the waypoints that has been created in the amcl new_test.yaml file and loaded in the rviz using topo_nav.rviz config file.

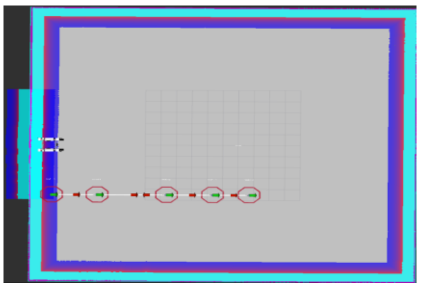


Fig 2: Waypoints created and loaded in the rviz.

4. AUTONOMOUS NAVIGATION AND OBSTACLE AVOIDANCE.

To navigate the robot autonomously the python script is written and the python language is used widely in the world for development of their projects and the ROS reads the python scripts and it is compatible. After modifying the map by creating the new waypoints to navigate the robot in the specified direction, the action client is created that send the goals to the robot topological navigation action.
The navigation is done in two ways: the one is navigating the robot in rviz by Click on the "2D Nav Goal" button, and then click and drag in the rviz window to give the robot a target position, known as goal pose the robot should drive to the goal pose on own without hitting anything on the way.
And the second way is writing the python script by sending the goals from one position to the other autonomously.
By running the below code in the terminal:

```
rosrun uol_cmp9767m_tutorial set_topo_nav_goal.py
```

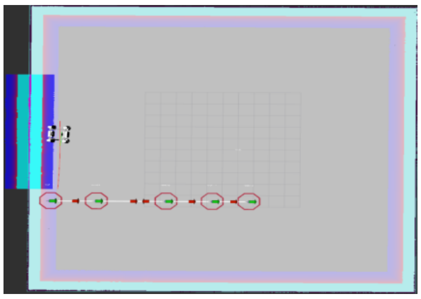
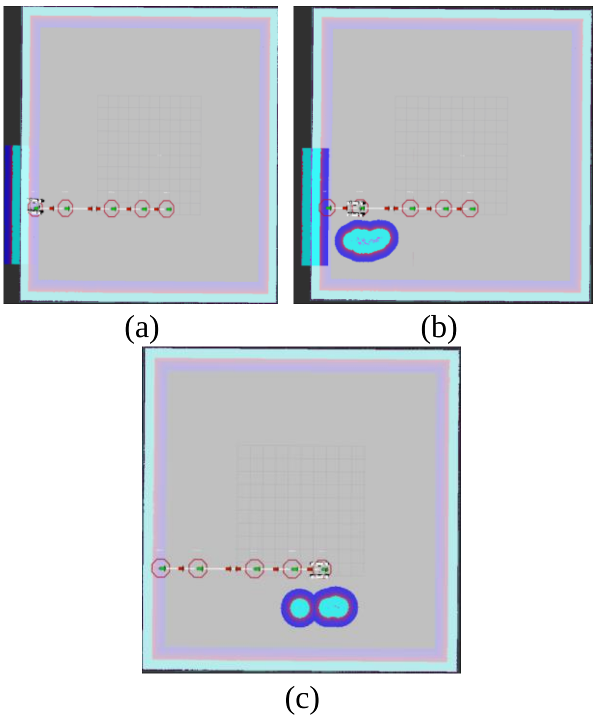


Fig 3: The robot started moving autonomously navigating through each waypoint.

The robot started navigating autonomously in the specified path and detecting the obstacle. The move_base performs the task to avoid the obstacles and this is done by using the data from the laser scanner sensor. The blue inflation layer shows that the robot is near to the obstacle.


(a) (b) (c)
Fig 4: (a)The robot navigated autonomously to the first waypoint.
(b)(c) The robot started moving autonomously navigating through each waypoint by detecting the obstacles.

5. DETECTION AND TRACKING USING OPENCV.

The ROS uses OpenCV for complex tasks so that they can be simplified [6]. In this work, Using OpenCV the grape bunches were detected in the frames by choosing the colour of the bound i.e., Grape bunches (purple colour) and by adjusting the lower bound and the upper bound colours for detecting the exact size and mask of the grape bunches. The image of each frame i.e., BGR is converted to the HSV (Hue, saturation, value) for creating the mask and kernel close is used to eliminate the background and keeping the foreground of the image for morphological transformation. With the help of mask, the contours were found and drawn the rectangle bound shape contour and added the text in tracking the exact number for each grape bunch in the frame. And the robot is navigated autonomously through each waypoint and making the robot to stand at least 10 seconds at each waypoint for detecting and tracking the exact number of grape bunches in the frames.

The bound values for detecting the masks are lower bound values (100, 18, 46) and upper bound values (107, 255, 255).

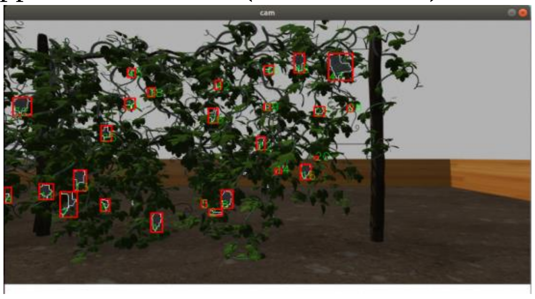
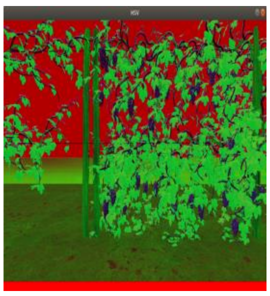
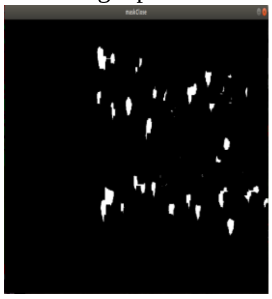


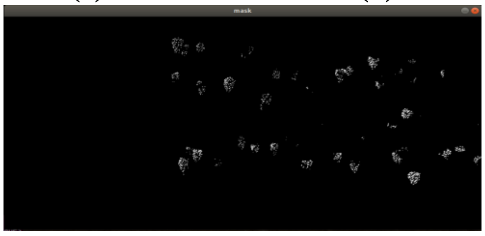
Fig 5: Detection of grape bunches and tracking it with the exact numbers using OpenCV.



(a)



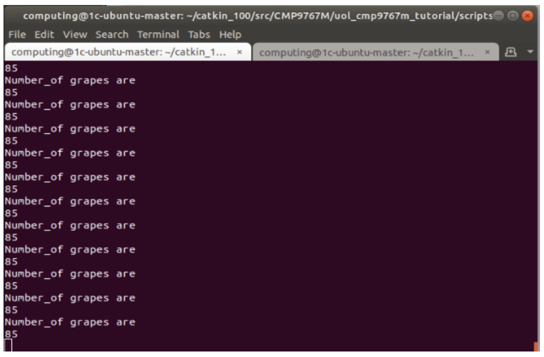
(b)



(c)

Fig 6: (a) HSV Image of the grape bunches.
(b) mask close of grape bunches.
(c) mask of the grape bunches.

6. RESULT



```
computing@1c-ubuntu-master: ~/catkin_100/src/CMP9767M/uol_cmp9767m_tutorial/scripts
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
Number_of grapes are
85
```

The robot is navigated autonomously from one waypoint to the other waypoint, and at each waypoint the robot stayed for 10 seconds for detecting the grape bunches and tracking it with the text number using the python script. The number of grapes is obtained in each frames of the vineyard_small map.

7. CONCLUSION

The robot moves autonomously through the created waypoints by detecting the grape bunches and tracked it with the text numbers in the given training maps. And it needs certain time to detect the grape bunches as it is going to stay for at least 10 seconds. Hence the number of grape bunches that the robot counts may varies from 1% to 5% from the actual number of the grapes. On comparing it with the actual environment, creating waypoints is the main challenging task to make the robot navigates autonomously in the actual environment.

8. REFERENCES

1. Programming robots with ROS: [book] [ebook] a practical introduction to the Robot Operating System *By* Quigley, Morgan
2. https://blackboard.lincoln.ac.uk/ultra/courses/_170758_1/cl/outline
3. Pajaziti, A., 2014. Slam–map building and navigation via ros. *International Journal of Intelligent Systems and Applications in Engineering*, *2*(4), pp.71-75.
4. Gomez, C., Hernandez, A.C., Crespo, J. and Barber, R., 2016. A topological navigation system for indoor environments based on perception events. *International Journal of Advanced Robotic Systems*, *14*(1), p.1729881416678134.
5. https://github.com/LCAS/CMP9767M/wiki/Workshop-8---Topological-Navigation
6. Bradski, G. and Kaehler, A., 2000. OpenCV. *Dr. Dobb's journal of software tools*, *3*, p.2