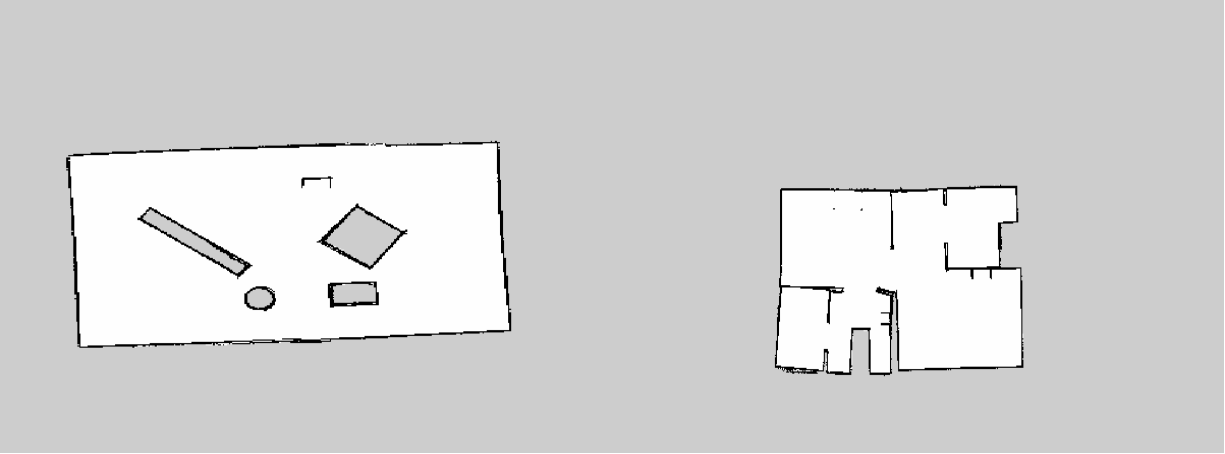
# ***GROUP 23 PROJECT 3B***

*TEAM MEMBERS : ZAKK GIACOMETTI (REPORTER)*

*VIVEK VERMA (TECH LEAD)*

**Task 1 :** The maps created in task 1(World index 0 and 1) are pasted below :



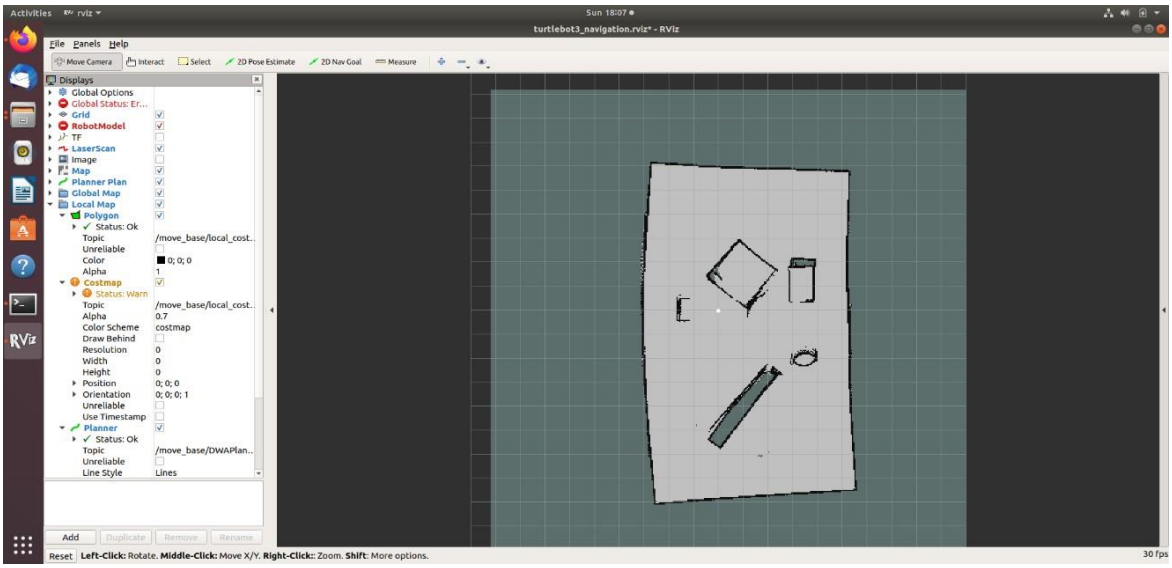
**Task 2 :** After its launch, the turtlebot3_navigation.launch file calls internally *turtlebot3_remote.launch*, *amcl.launch*, *move_base.launch(which has a move_forward_only argument set to "false")* and *turtlebot_navigation.rviz.*

The following nodes become active:

1.)*amcl* – It is a localization system for a robot moving in 2D. It takes in laser-based map, laser scans, transformed messages and outputs pose estimates.
*2.)map_server* – Map server provides the map_server node to ROS node, which offers map data. It also provides the map_saver, which allows dynamically generated maps to be saved to file.
*3.)move_base* – It results in the robot attempting to achieve a goal pose with its base to within a tolerance.
*4.)robot_state_publisher* – It allows to publish the state of a robot to tf. Tf is a package that keeps track of multiple coordinate frames over time.
*5.) rviz* – 3D visualization tool for ROS.

A *navigation stack* basically takes in information from odometry and sensor streams and outputs velocity commands to send to the mobile base.

Below is the example of a map loaded in RVIZ :




**Task 3 Node:**

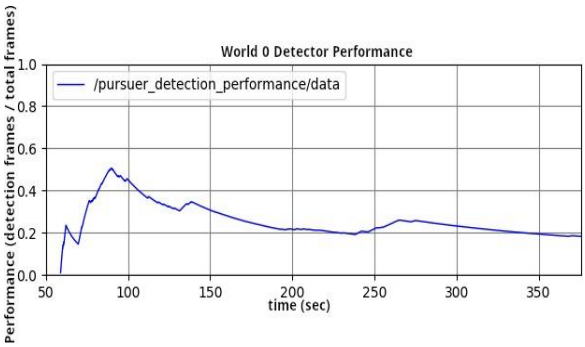
To track the evader, we equipped the turtlebot with a combination depth and rgb image sensor. The parameters of the camera are similar to those of real depth cameras available commercially (such as the Stereo Labs ZED) with resolution downscaled to 640x480.

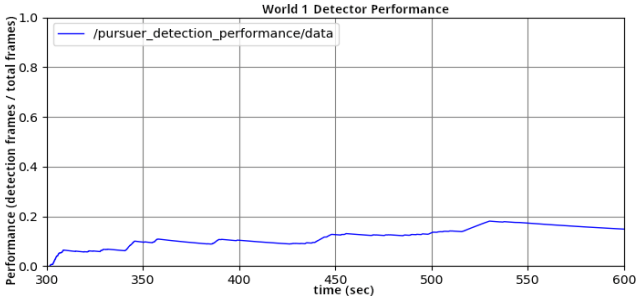
The pursuer_detector.py node in the pursuit_evasion package handles detection of the evader and extraction of high-level information of its information in the camera view. Specifically, it outputs a region of interest and approximate distance to the evader. Please see the included README.md for detailed topic types and information. The node can be launched with the correct parameters for tb3_0 with the included launch file as roslaunch pursuit_evasion detect_pursuer.launch PursuerDetector. Our detector utilized a Histogram of Oriented Gradients (HOG) approach, using the built-in HOGDescriptor class from OpenCV. Additionally, the pre-trained person model provided by OpenCV (as HOGDescriptor.getDefaultPeopleDetector()) was utilized. A winStride of (8,8), padding of (8,8), and scale of 1.05 were set as the parameters of the HOG detector.

After extraction of possible bounding boxes via HOG, we apply Non-Maximum Suppression (NMS) to eliminate most overlapping detections. NMS in this case is done by geometric sorting but sorting by weights was also experimented with. Finally, a heuristic thresholding is done on the remaining detections based on detection weight and bounding box height.

A metric of detection performance is calculated from the ratio of image frames with a detection to the total number of image frames received. This is also published on the ROS network.

The tracking performance for both the world is pasted below:




**Task 4 Node :** In this node we've setup a *Multigoals* class which has publisher *pub* setup which publishes goalMessages which is a type of *PoseStamped* message to the subscriber *sub* whose node is MoveActionResult. Topic for the subscriber is "*/tb3_0/move_base/result*" and topic for the publisher is "*/tb3_0/move_base_simple/goal*".
There are two more subscriber's setup: 1.) Subscribes to the */pursuer_distance*. This value is taken from the depth image, which outputs the depth of the detected region in meters. 2.)Subscribes to the */pursuer_detections*. The output is x,y values of bottom left of the rectangular edge and also the width and height(w,h) of the boxes which are added to the x,y coordinate to get a rectangle.

We make use of the *transformListener()* in this project, in which the incoming goal is transformed from the robots to the maps frame. The goals which we set for pursing the evader is in the *recalculate_goal* method. What it basically does it is it get the depth value optimizes it(we've used original value-0.5 for optimized results) and sets it as a *x goal* for the robot. (The x-coordinate of the bounding box + width/2 – 640/2 )*8/640 (We optimize the values and do a scaling from pixel to real time map frame) forms the *y goal* of the goalmessage which we set.