DESIGN ORIENTED PROJECT

ROBOT NAVIGATION USING REINFORCEMENT LEARNING

SUBMITTED TO -

DR J.L. RAHEJA

SUBMITTED BY -

RIGVITA SHARMA
2016A7PS067P
SHRUTI SHARMA
2016ABPS0833P

1. **Introduction**
    a. **Abstract**

Developing a proper navigation system requires an efficient object detection algorithm with tuning over a large dataset for training. However, this manual labelling of data could be sometimes expensive. Here we have worked on the acquisition of depth point cloud from a Kinect sensor and tried training a robot in a simulated environment (ROS-gazebo) using a dueling architecture based deep double-Q network (D3QN) for obstacle avoidance. The actual implementation and tuning on a simulated robot is performed using ROS and can be transferred to real robotic systems.

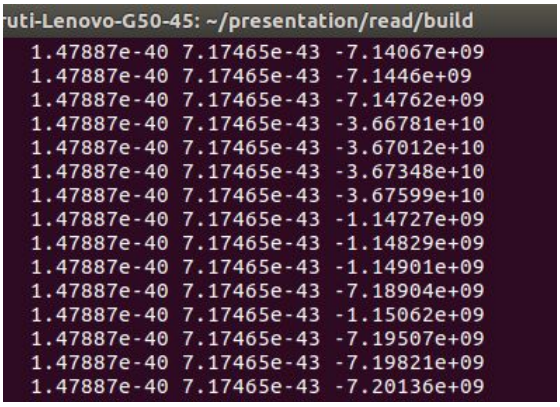
    b. **Problem Statement**
        i. Work on the conversion of RGB-D images into point cloud data.
        ii. Development of an object detection algorithm using ROS-PCL
        iii. Developing a dataset of depth PCD images using Kinect v2 for assisting in the implementation of a given algorithm on an actual robot.
        iv. Development of an obstacle avoidance algorithm using reinforcement learning

2. **Conversion into PCD**
    a. Dataset used -
    PUTKK – PUT Kinect 1 & Kinect 2 data set, Mobile Robots Lab
    b. Library used - Open 3D
    c. Code snippet **:**

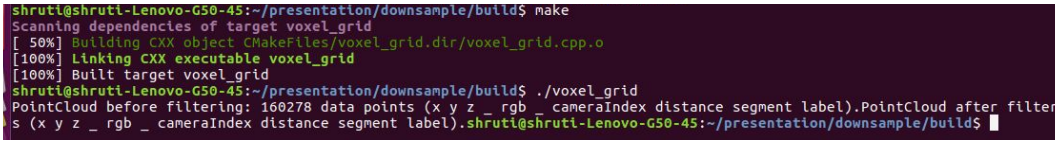
```
pcd = create_point_cloud_from_rgbd_image(rgbd_image,
PinholeCameraIntrinsic(PinholeCameraIntrinsicParameters.PrimeSenseDefault))
pcd.transform([[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, 0], [0, 0, 0, 1]])
write_point_cloud(str(file)+".pcd", pcd)
```

3. **Object Detection using ROS-PCL**

a. Dataset used -
    i. Cornell RGB-D dataset with 24 labelled office scene point clouds.
    ii. The files are in .pcd format with fields - x, y, z, RGB, camera Index, distance_from_camera, segment_number and label_number. The label string corresponding to the label_number can be obtained from the file scene_labelling/scene_processing/labels.txt (for example, 1 is 'wall', 2 is 'floor', etc.).
b. Computational Tools used -
    i. Robotic Operating Software - ROS
        1. Point cloud library (PCL) is used to handle the 3-D point cloud data
c. Reading the data -
    i. Every point cloud is stored using a pointer,
    ii. and the x,y,z coordinates are extracted.



d. Filtering
    i. Downsampling :
        1. It is used to reduce the number of data points.
        2. A 3d voxel grid is created in the space and all the points present are approximated as the centroid of these points.
        3. For eg executing this on scene 1, we get :

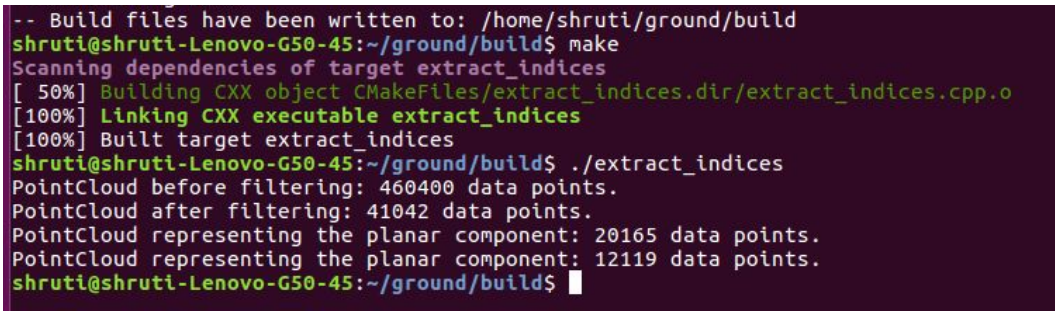


        4. Sparse outlier removal based on the computation of the distribution of point to neighbours distances in the input

dataset.

a. We compute the mean distance from a point to all its neighbours, for all points. We assume that the resulted distribution is Gaussian with a mean and a standard deviation. Then, the mean and standard deviation can be considered as outliers and trimmed from the dataset for all points whose mean distances are outside an interval defined by the global distances.

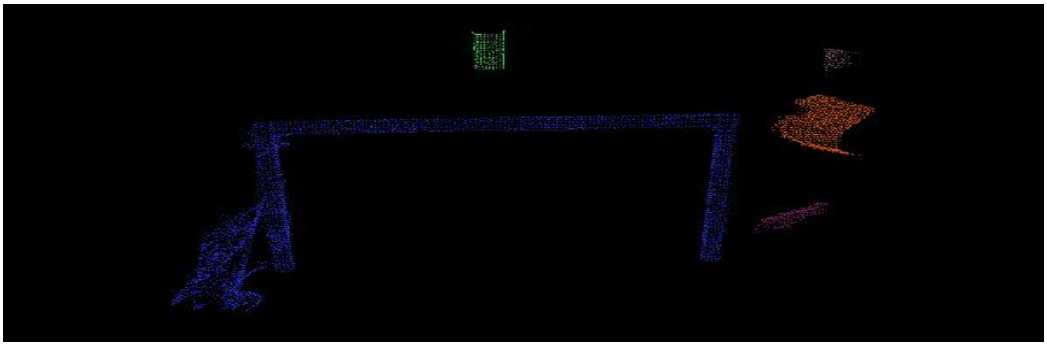
e. Segmentation
   i. Plane Model Segmentation
      1. Does simple plane segmentation of a set of points,
      2. that is found all the points within a point cloud that support a plane model
      3. Used for the identification of planes such as ground.



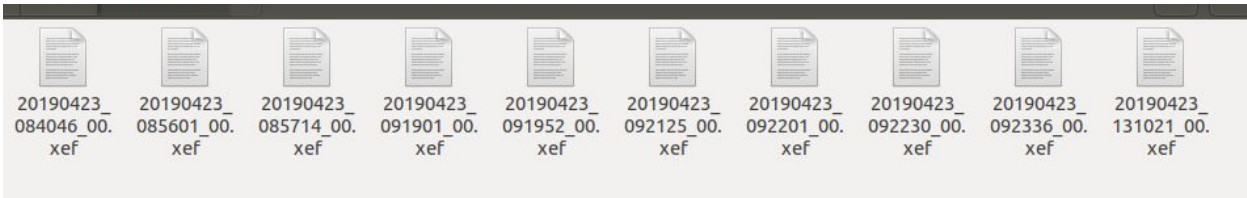
f. Object Detection
   i. Euclidean Clustering
      1. After segmentation of planes such as walls and ground, we need to identify the objects or obstacles.
      2. Space is divided into a 3D grid, using fixed width boxes, or more generally, an octree data structure - that is subdividing the space recursively into eight octants.
      3. We find and segment the individual object point clusters on the plane.
      4. Similar to the flood fill algorithm.

   ii. Results -

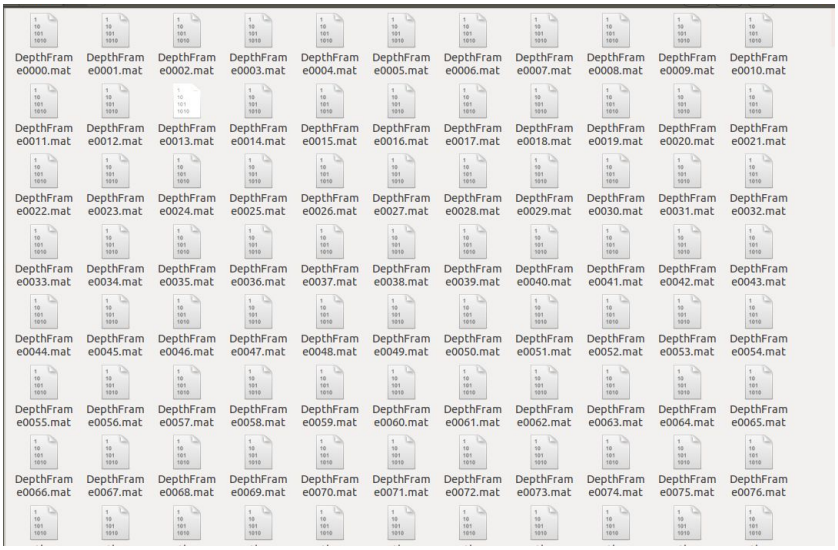
```
Scanning dependencies of target cluster_extraction
[ 50%] Building CXX object CMakeFiles/cluster_extraction.dir/cluster_extraction.cpp.o
[100%] Linking CXX executable cluster_extraction
[100%] Built target cluster_extraction
shruti@shruti-Lenovo-G50-45:~/distance/build$ ./cluster_extraction
PointCloud before filtering has: 460400 data points.
PointCloud after filtering has: 41042 data points.
PointCloud representing the planar component: 20527 data points.
PointCloud representing the planar component: 12433 data points.
PointCloud representing the Cluster: 4872 data points.
PointCloud representing the Cluster: 1386 data points.
PointCloud representing the Cluster: 320 data points.
PointCloud representing the Cluster: 290 data points.
PointCloud representing the Cluster: 121 data points.
shruti@shruti-Lenovo-G50-45:~/distance/build$ ./pcl_viewer scene1.pcd
bash: ./pcl_viewer: No such file or directory
shruti@shruti-Lenovo-G50-45:~/distance/build$ ./pcl_viewer cloud_cluster_0.pcd cloud_cluster_1.pcd cloud_cluster_2.pcd cloud_cluster_
d_cluster_4.pcd
bash: ./pcl_viewer: No such file or directory
shruti@shruti-Lenovo-G50-45:~/distance/build$
```



4. **Kinect and Point cloud Dataset -**
    a. An indoor environment consisting of an indoor hostel scene was created with a sudden falling object
    b. Kinect Studio was used to capture the coloured and 3-D point cloud video.
    c. The video in .xef format was converted into frames and stored as .mat files.
    d. The recorded videos using Kinect v2 in the .xef format



    e. The conversion into frames/ depth images -



5. Obstacle avoidance using deep reinforcement learning -

We use dueling network based deep double-Q network for obstacle avoidance.

a. What is D3QN?

Optimal Q-value function -

By choosing the optimal action each time where $Q^*(x_t, a_t) = max_\pi \mathbb{E}[R_t | x_t, a_t, \pi]$, we can have the optimal Q-value function

$$Q^*(x_t, a_t) = \mathbb{E}_{x_{t+1}}[r + \gamma \max_{a_{t+1}} Q^*(x_{t+1}, a_{t+1}) | x_t, a_t], \quad (2)$$

Here, we approximate the optimal Q-value function with deep neural networks, instead of computing the Q value function directly from a large state space.

b. Dueling Network - In traditional DQN, we estimate the Q-value of each action-state pair, given the current state using convolution layers. Then, only one stream of fully connected layers are constructed. However, in the dueling network, we have two streams of fully connected layers to compute the value and advantage functions separately. These streams are finally combined together for computing Q-values. This two-stream

dueling network structure is as -

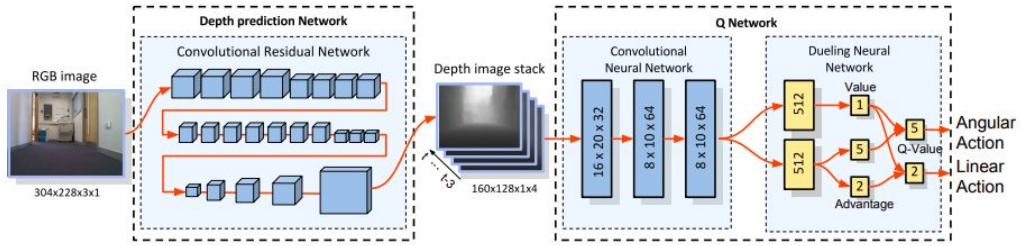


Fig. 1: Network architecture of monocular image based obstacle avoidance through deep reinforcement learning. A fully convolutional neural network is firstly constructed to predict depth from a raw RGB image. It is then followed by a deep Q network which consists of a convolutional network and a dueling network to predict the Q-value of angular actions and linear actions in parallel.

c. Methodology -

The online network updates weights by back-propagation at every training step. However, unlike the online network, here the weights of the target network are set for a short period and then copied from the online network. The result state xt+1 is modified from both the online as well as the target network to calculate the optimal value Q 0∗ at time t +1. Then, with the discount factor γ and current reward rt, we receive the target value y at t. Finally, the error is calculated by deducting the target value with the optimal value Q ∗ predicted by the online network, given current state x, and is then back propagated to update the weights.

With the two techniques, the D3QN is expected to be more data efficient to speed up learning.

TABLE I: Parameters of D3QN Model for Obstacle Avoidance

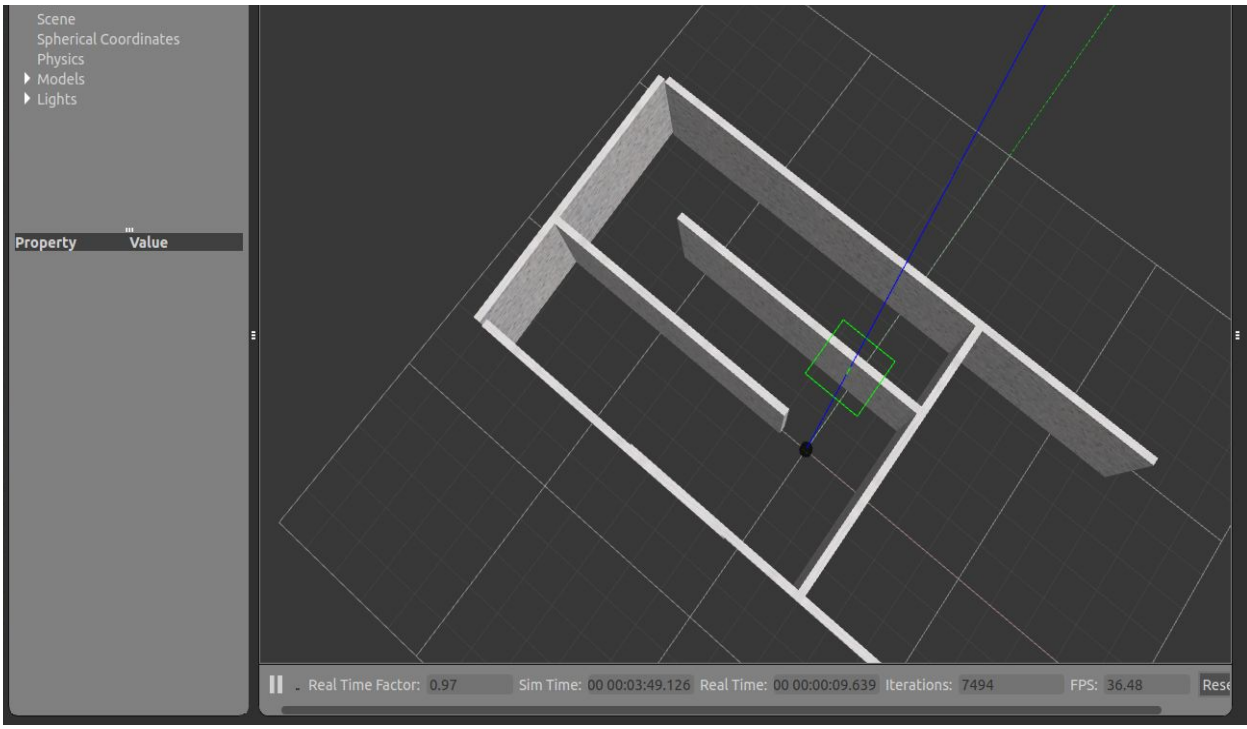
Name of layer	Size of filters or number of neurons	Stride
Conv 1	(10, 14, 32)	8
Conv 2	(4, 4, 64)	2
Conv 3	(3, 3, 64)	1
FC 1 for advantage	512	-
FC 1 for value	512	-
FC 2 for advantage of angular actions	5	-
FC 2 for advantage of linear actions	2	-
FC 2 for value	1	-

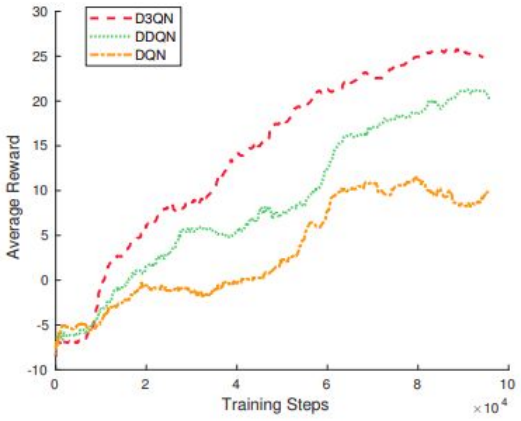
d. Model and training settings -

To train the network to produce a feasible control policy, robot actions are defined using linear and angular velocities in discretised form. $r = v * \cos(\omega) * \delta t$ where v and ω are local linear and angular velocity respectively and δt is the time for each training loop, is defined as the instantaneous reward function. Here, δt is set to 0.2 seconds. The reward function is designed for the robot to run as fast as possible and be penalized by merely rotating on the same locality.

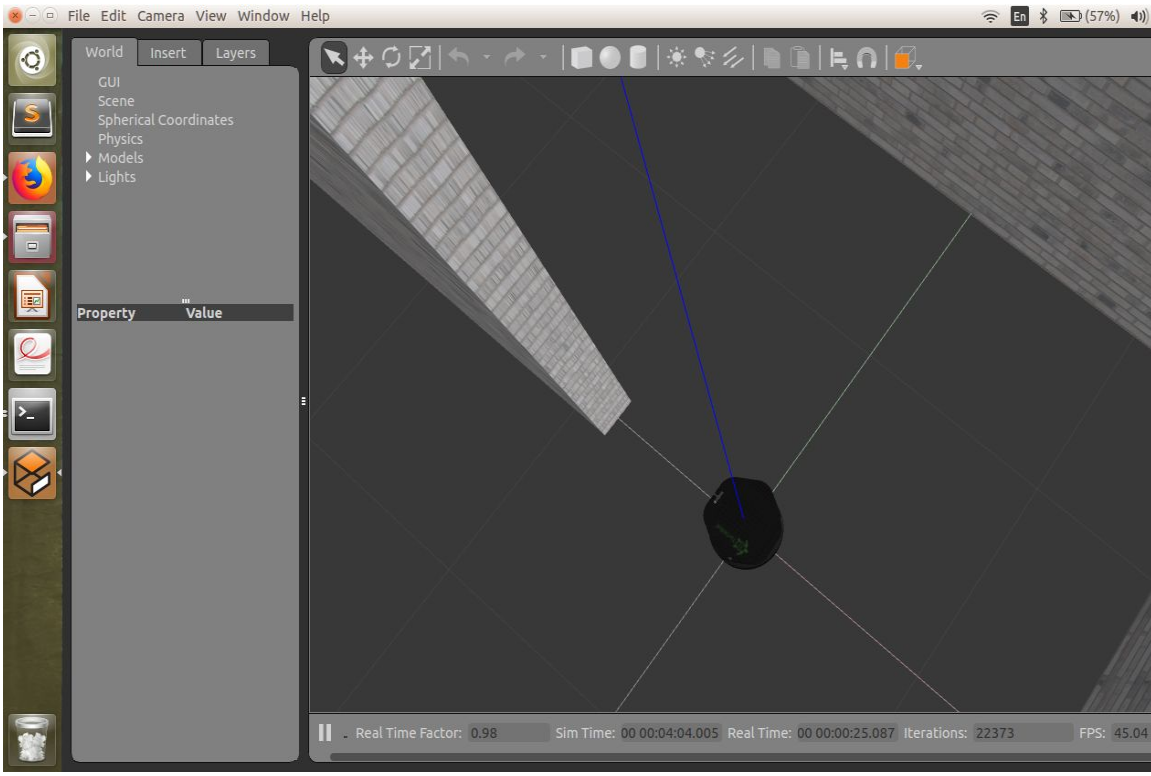
The sum of the instantaneous rewards at all steps in an episode is the total episode reward. If there is a collision, we add an extra penalty of −10. Otherwise, the episode will continue until the maximum number of steps (Our experiments - 500 steps) and terminates with no penalties designed.

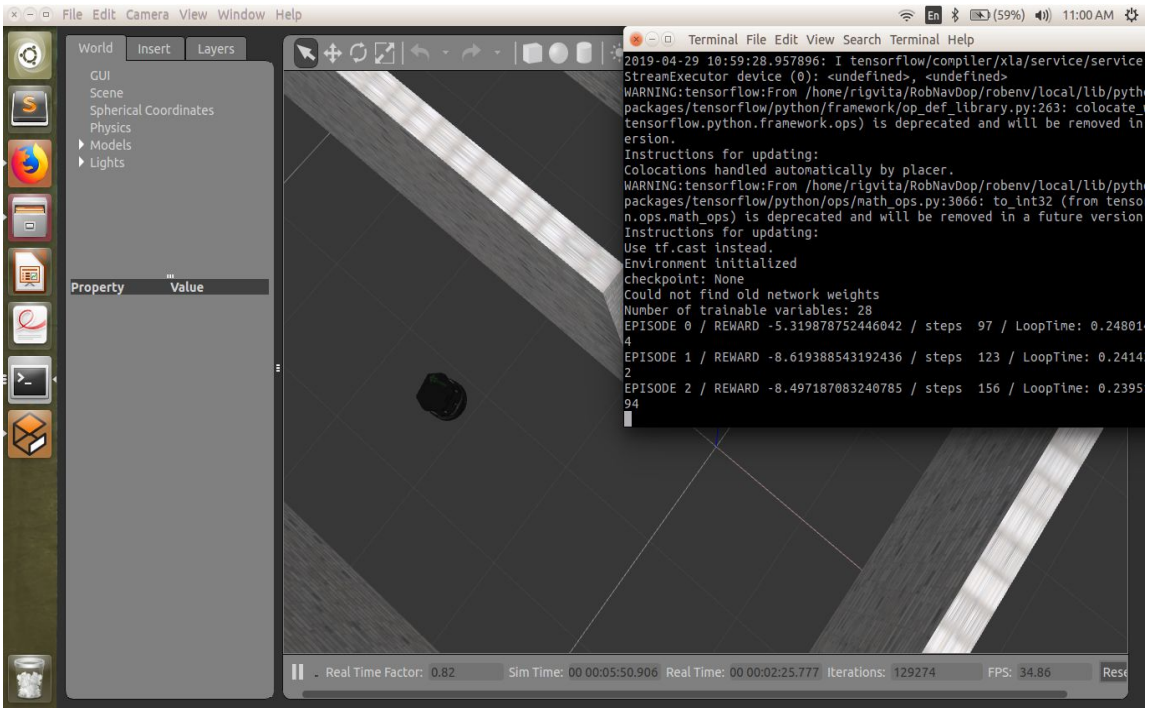
The environment -

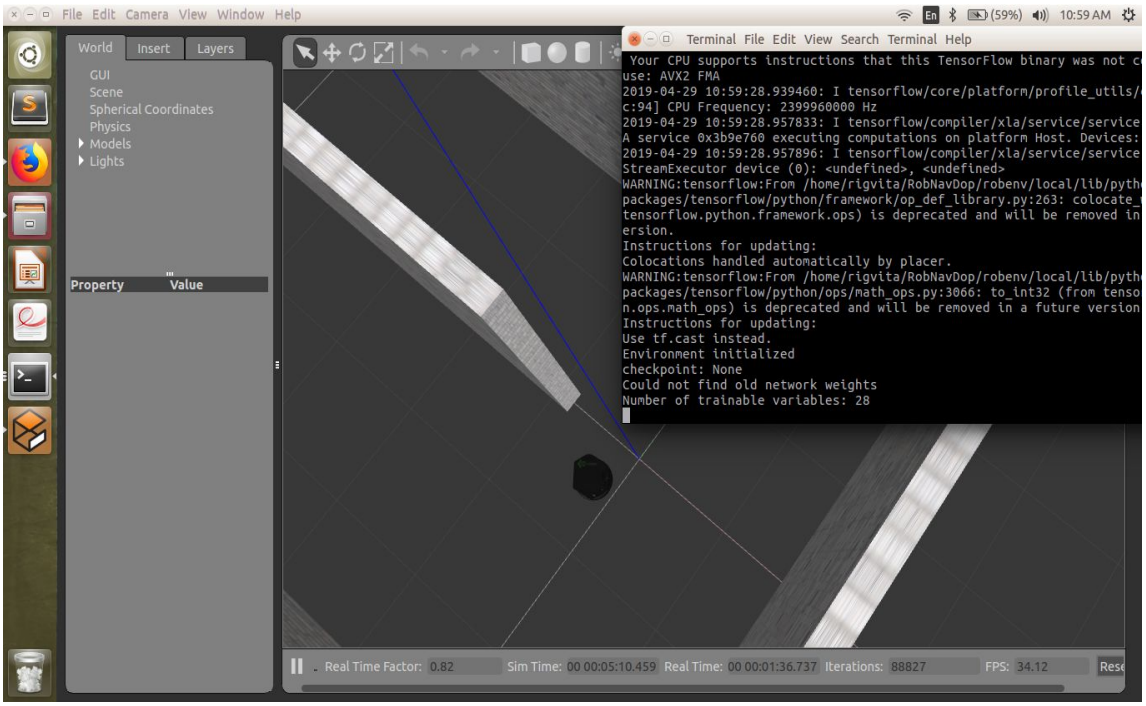




e. Experimental results - in a simulated environment







6. **Conclusion -**

We have used D3QN based algorithm for obstacle avoidance using RGBD image as input. The training is done in a simulated environment. We have worked in a simulated environment and the proposed algorithms can be easily applied to

real-world robots to test the efficiency of the simulated model.

7. References

a. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning, Linhai Xie, Sen Wang, Andrew Markham and Niki Trigoni
b. Kinect and RGBD Images: Challenges and Applications, Leandro Cruz, Djalma Lucio, Luiz Velhoz, IMPA - VISGRAF Lab Rio de Janeiro, Brazil
c. https://code.tutsplus.com/tutorials/histogram-equalization-in-python--cms-30202
d. Open3D: A Modern Library for 3D Data Processing Qian-Yi Zhou Jaesik Park Vladlen Koltun Intel Labs
e. https://ythej.wordpress.com/2017/06/05/setting-up-kinect-v2-for-ubuntu/
f. https://github.com/georgeerol/RoboticPerception
g. http://incompleteideas.net/book/bookdraft2017nov5.pdf
h. http://wiki.ros.org/pcl/Tutorials
i. https://arxiv.org/pdf/1804.10332.pdf
j. Automatic Robot Navigation Using Reinforcement Learning, Amirhosein Shantia