AA 274: Principles of Robotic Autonomy
Problem Set 3

Name: Ashar Alam
SUID: 06265091 ashar1

March 8, 2019

Problem 1

(i) No need for writeup

(ii) No need for writeup

(iii) No need for writeup

(iv) No need for writeup

(v) No need for writeup

(vi) No need for writeup

(vii) No need for writeup

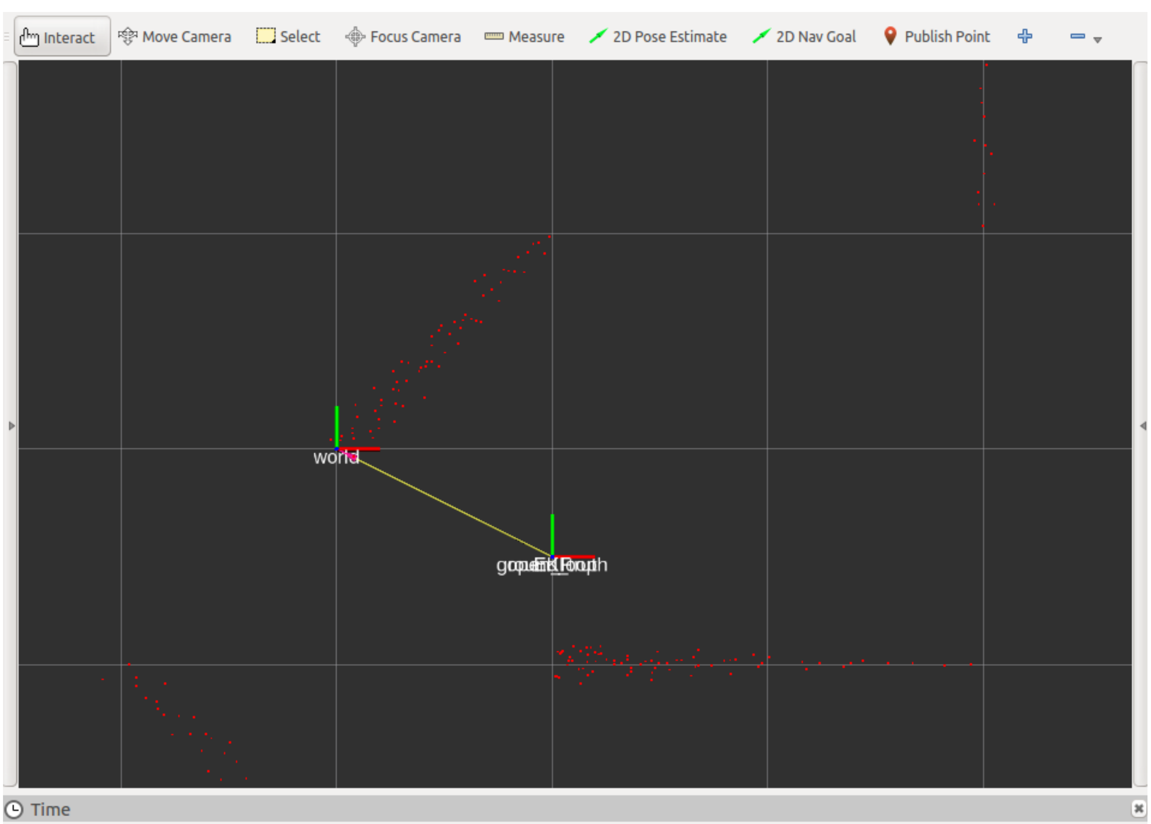
(viii) 

Figure 1: Initial state

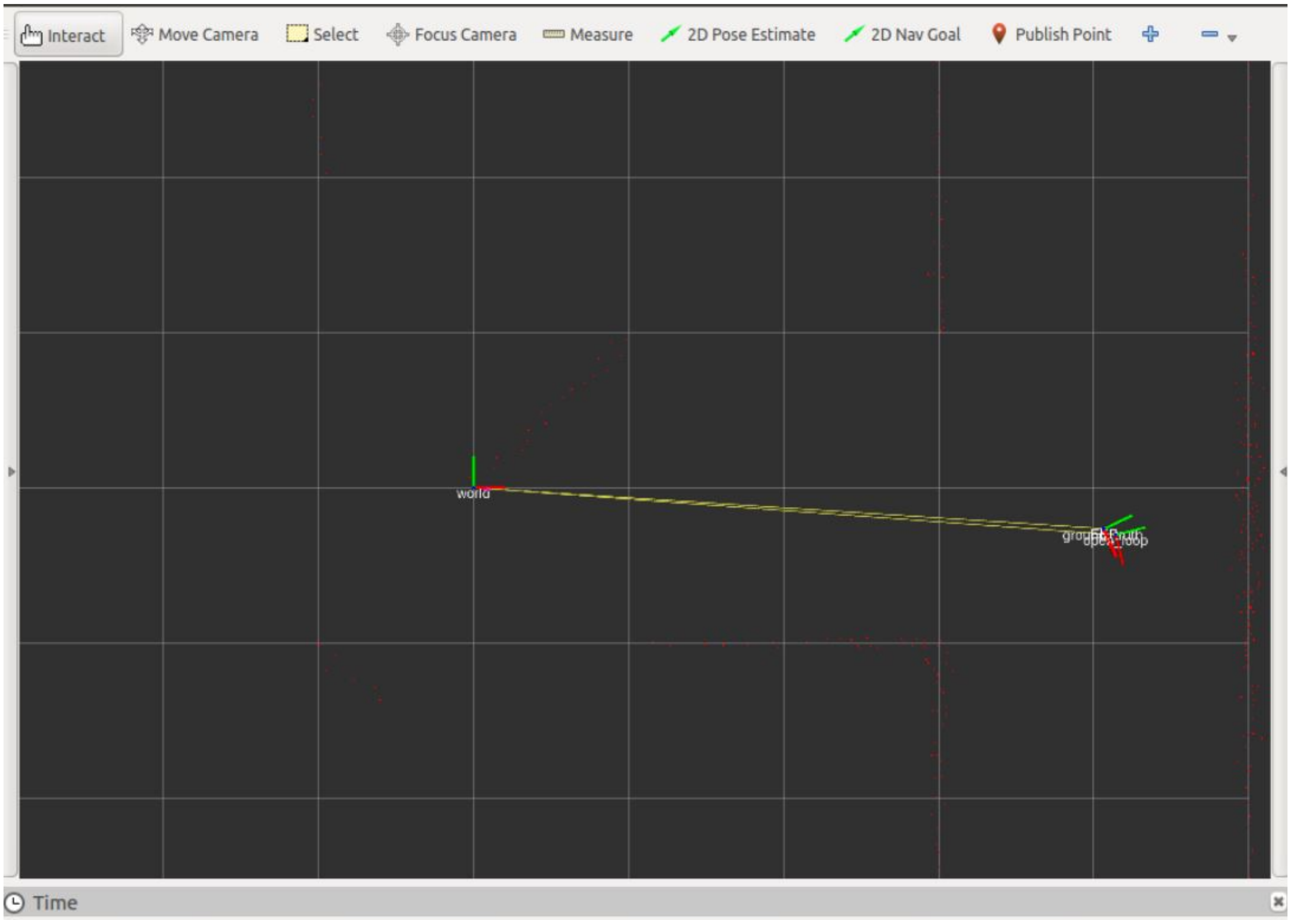


Figure 2: Far from initial state

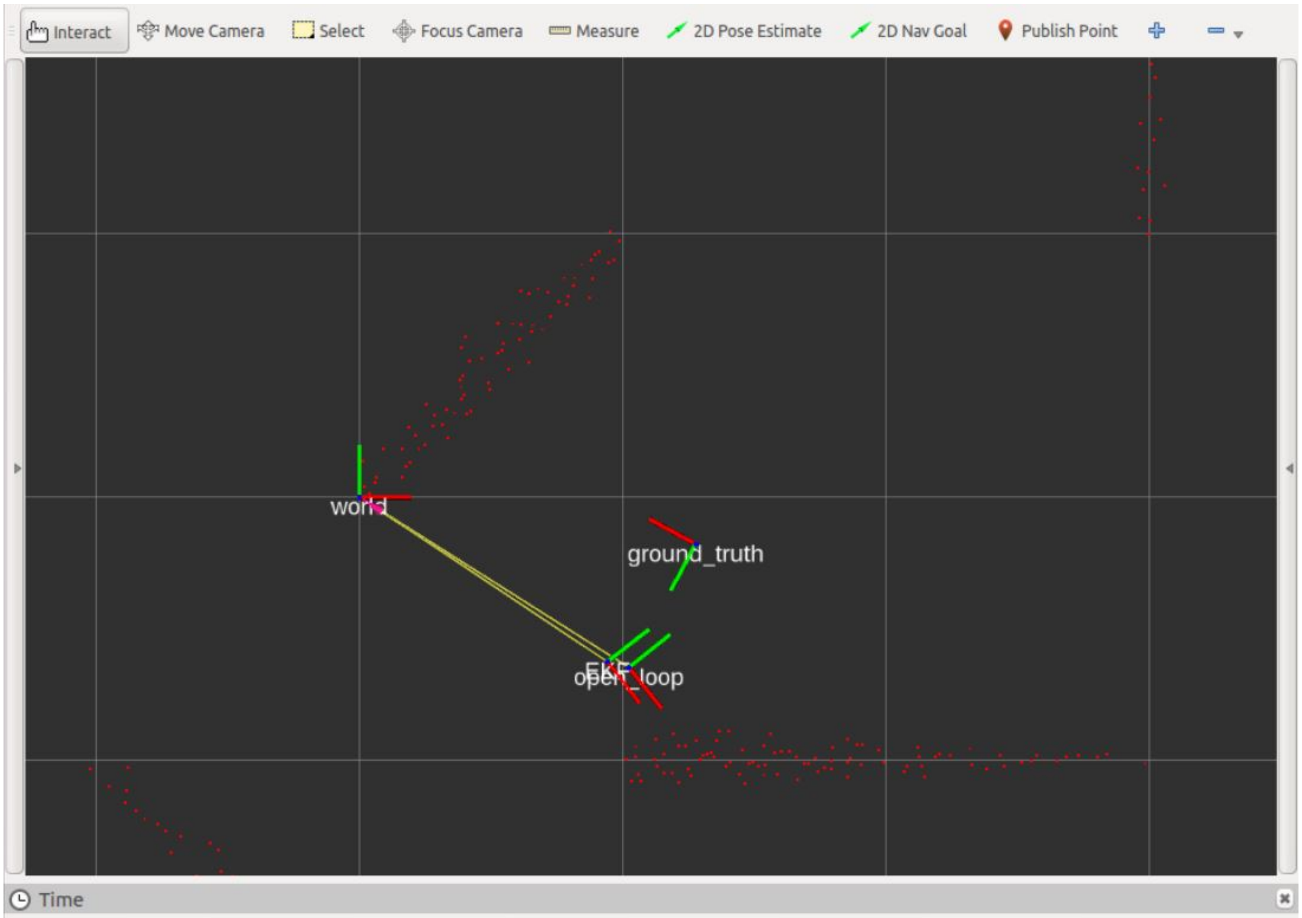


Figure 3: Estimate diverges

Open loop and EKF state estimate changes when the robot encounters rapid turns in its path. This corresponds to variation of trajectory of the turtlebot from linear one (in the initial state) to a non-linear one (in the diverging state). When this happens, it introduces errors in velocity calculation and state tracking because of introduction of noise in encoders or slipping/drifting of the robot wheels during frequent turns or delay in camera calibration during turns.

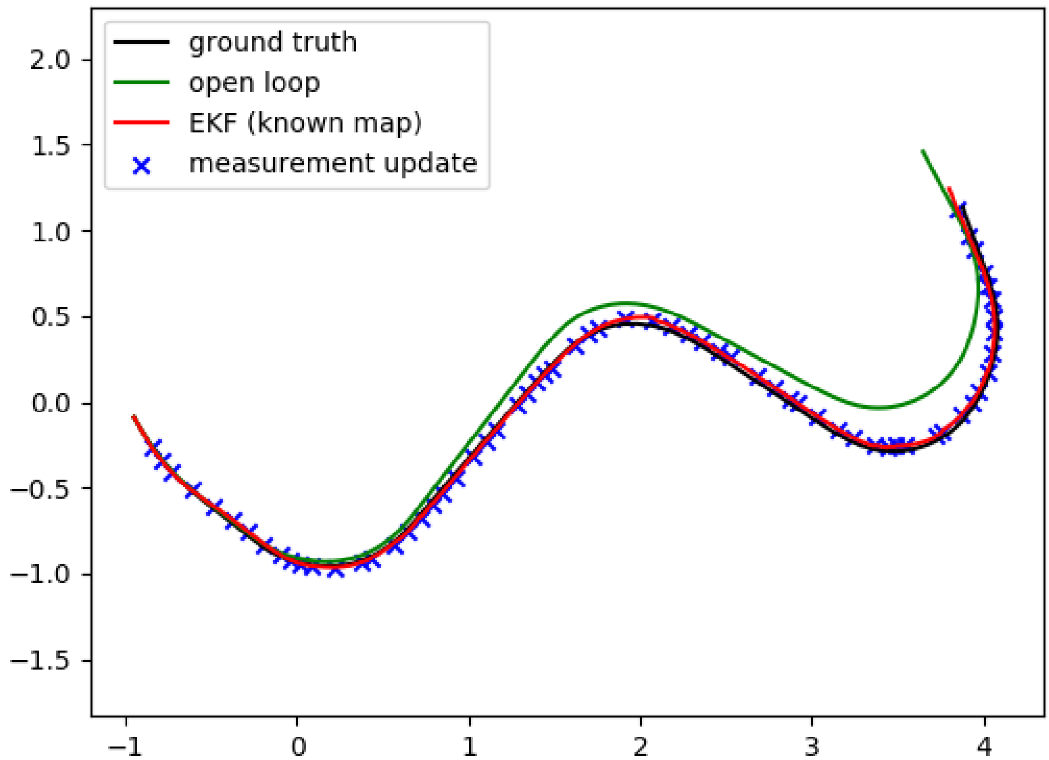


Figure 4: EKF Localization validation results

Problem 2

(i) No need for writeup

(ii) No need for writeup

(iii)

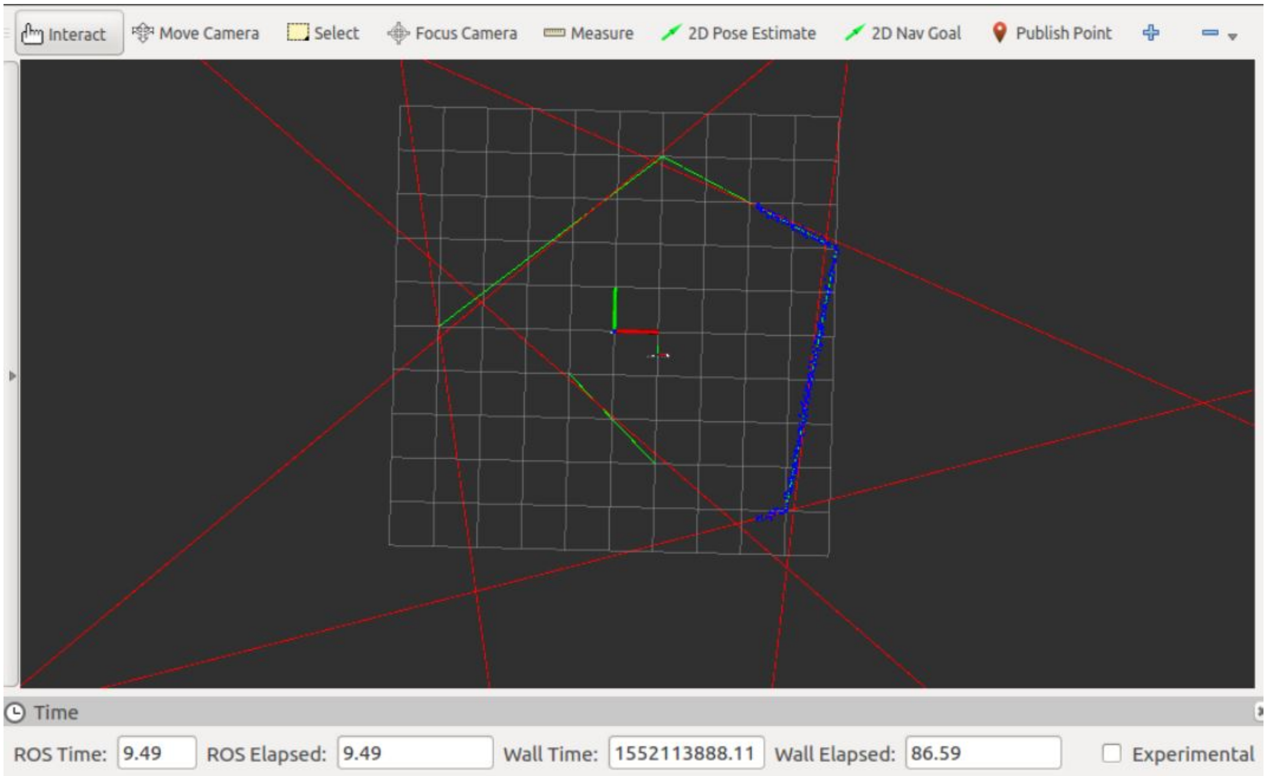


Figure 5: Initial state

Figure 6: Far from Initial state

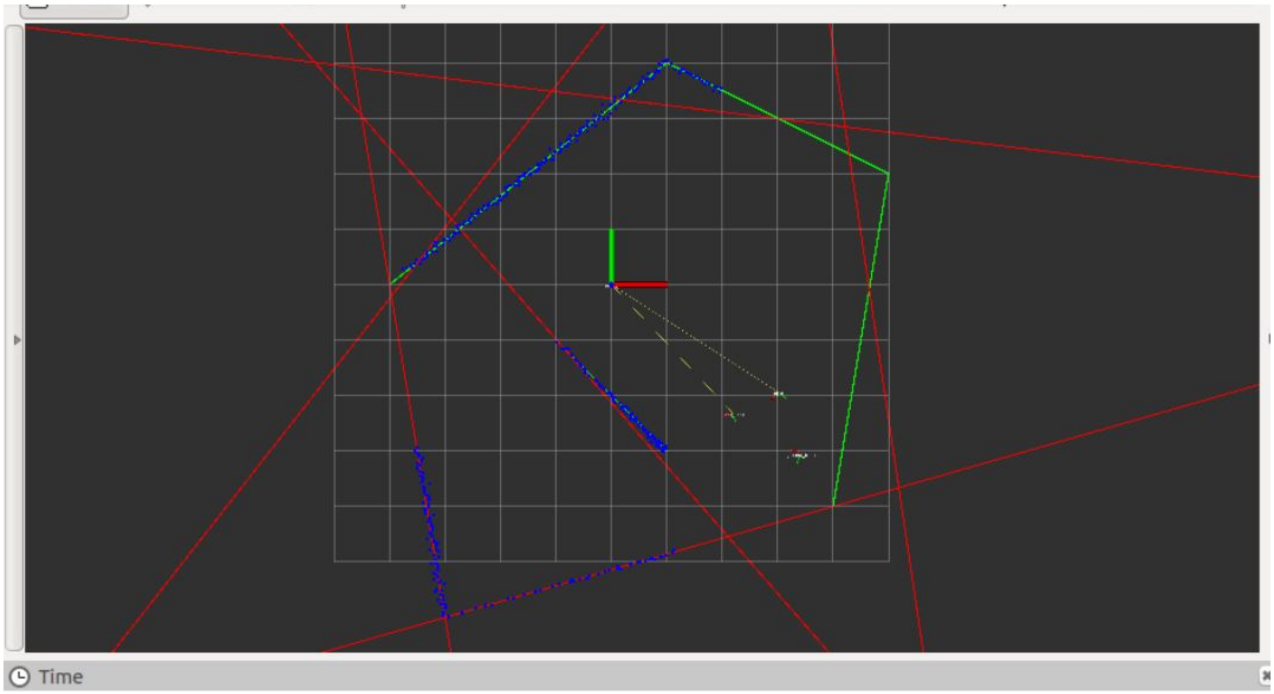


Figure 7: Estimates converged

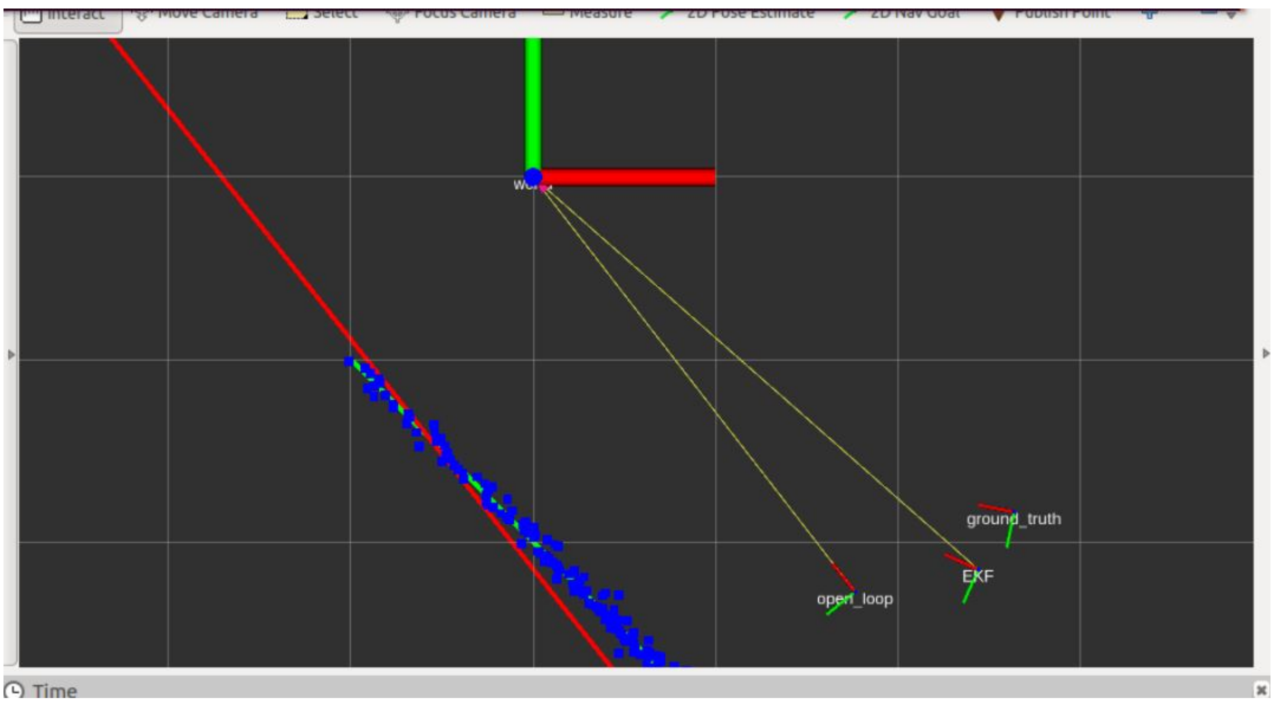


Figure 8: Estimates diverged

EKF and ground truth estimates diverge when the robot encounters and tries to run into a wall. This is because EKF estimates do not account for the motions when the robot runs into physical barriers like walls, fences, etc.

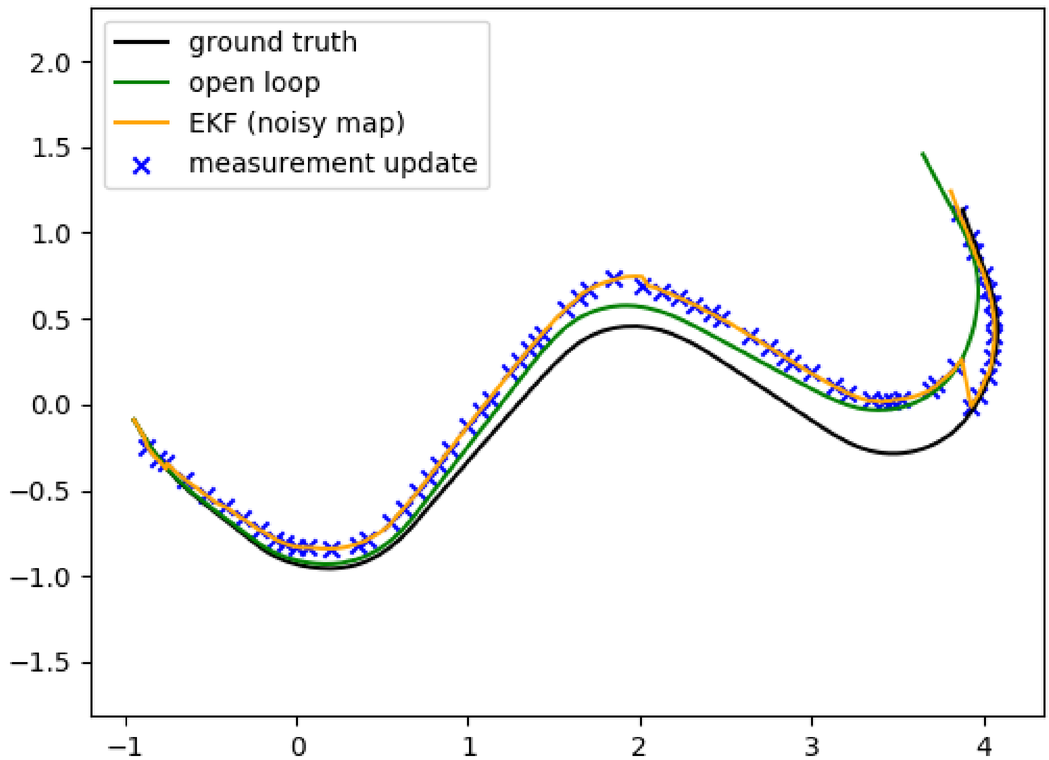


Figure 9: EKF SLAM validation results

Problem 3

(i) The use_gazebo parameter is used to determine when to subscribe to the Model_States message. When this is set to false; the current node no longer subscribes to the Model_States message which holds turtlebot's pose. In this case, turtlebot's position is updated using tf.
rosparam is command-line tool for getting and setting ROS Parameters on the Parameter Server. To set or check the value of rosparam in the terminal, we can use the rosparam_ set or rosparam_get command, respectively. In hw2_control_demo.launch we used rosparam as $< rosparam\, param = "sim" > false < /rosparam >$ which can be broken down as the following syntax: $< rosparam\, param = "parameter\, name" >< true/false >< /rosparam >$ to set the rosparams.
Yes, we can use tf_lookup to get the robot states and accommodate in the supervisor.py script when Gazebo is not used.

(ii) **First section (lines 2-6)**: These lines are used to set arguments (parameters) for Gazebo.
Second section (lines 9-15): These lines are used to set ROS parameters for several parameters namely sim, map,use_tf, rviz.
Third section (lines 17-23): These lines are used to generate empty world environment for the Gazebo by launching the empty_world.launch file and speciy the arguments(parameters) for this .launch file.
Fourth section (lines 25-28): These lines specify robot_description paramaters and spawns our urdf turtlebot in Gazebo.
Fifth section (lines 29-31): These lines setup the node named "robot_state_publisher" and sets its

arguments(parameters).
Sixth section (lines 33-65): These lines setup a node named slam_mapping from the gmapping package and sets its many arguments (parameters).
Seventh section (lines 67-69): These lines launch various nodes from the asl_ turtlebot package namely, turtlebot_detector, turtlebot_pose_controller, gazebo_plot.
Last section (lines 71): This line launches rviz node while specifying its other parameters.

When we intentionally drive into an obstacle, we observe that the frames of map and odom separate which were earlier coincident.

(iii) We wanted to determine the direction of the robot to know the front-facing face of the robot. So, we added arrow markers.
We can add visualization markers by adding a marker node in a launch file using code like:
$< nodepkg = "asl\_turtlebot" \; type = "marker" \; name = "MARK" >$ Then, we can publish this mode to see the markers.