ROS: Robot Localization

Course assignment: Intelligent Agents and Robotic Systems
University of Piraeus, Demokritos

Papadopoulos, Vasileios
vassilispapadop@gmail.com

March 7, 2021

1 Introduction

Robot localization is a process to examine the exact position within its environment; Navigation is a crucial activity for a mobile robot for safety, operation and other reasons. A mobile robot needs to know not only its *absolute* position but also its *relative* position in respect, for example to a human who might interact with it. Navigation process requires 4 building blocks; *Perception*, *Localization*, *Congition* and *Motion Control*.

2 Methods

The *robot_localization_demo.launch* file consists of total 6 nodes.

- turtlesim node: displays real turtle's measurements and estimated position
- turtle1_position_system_node: adds noise to the position of the turtle at a given frequency
- turtle1_odomentry_node: adds noise to the movements of the turtle at a given frequency
- robot_localization_ekf_node_odom: Extended Kalman Filter to odometry frame
- robot_localization_ekf_node_map: Extended Kalman Filter to map frame
- transformation_visualiazation_node: Visualization of estimated position in map frame

In order to point out the differences between robot_localization_ekf_node_odom and robot_localization_ekf_node_map we need to briefly describe *Kalman Filter* and *extended Kalman Filter*. Kalman Filter, is an iterative mathematical process that uses set of equations and consecutive data inputs to estimate values such as: position and velocity of an object when the measured values contain unpredicted or random error, uncertainty or variation. The Kalman Filter does not *wait* for a bunch of input data to make an estimate(for example, average) instead, it makes a quick prediction/estimate from few data points by *undestanding* the variation or the uncertainty in them. Generally, the input data are not the true value but something around the true value, Kalman Filter is a process of estimating the true value *real time*. The extended version of the process, models the added noise as a non-linear function.

Both EKF nodes are operating at the same frequency 10Hz which is the rate of producing a state estimate. Also, they have the same sensor_timeout. The differences lie onto where the filter is making an estimate. In the first case, it estimates the odom frame while in second the map frame.

To perform our experiments we made some preliminary changes to the original demo. At first step, we launched the robot_localization _robot package and recorded a *rosbag* of approximately for 35 seconds passing the parameter *-a* to capture all available topics. This recording is used for the carried experiements, which will be present in later sections. Then, we cloned the robot_localization _demo.launch file into mylaunch.launch and removed the node with name *teleop*, type *turtle_teleop_key* from package *turtlesim* to prevent keyboard teleoperation. Within the newly created launch file, we added a new node from the pack-

age *rosbag*, type *play* and passed the path of the previous rosbag. Now, everytime we launch the *mylaunch.launch* file the simulation immediatelly starts playing the rosbag file. We ran 6 different cases. This step is referred as *baseline* in which we simply replay the rosbag mentioned above. The other five cases are presented in the Table 1 below.

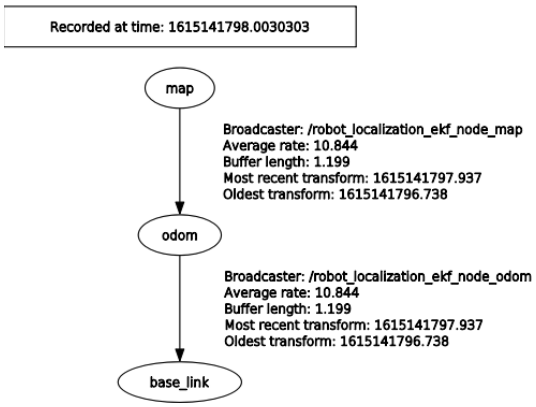

Figure 1: rqt tf tree

3 Results

In this section we present the results of previous test cases. The simulation consists of 4 turtles. The turtle with *gray* line is the real turtelsim robot. The position sensor which measures the absolute position and orientation is drawn with *blue* line while the turtle which measures the linear and angular velocity is shown in *red* line. Finally, the *green* line represents the estimate of the pose in map frame.

In Figure 2, the baseline case we could track the *red* line and how it starts to drift apart. Red line estimates pose in the map frame by odom frame. The accumulated error becomes clearly visible.

In Figure 3, in contrast to baseline case we have increased the noise in position sensor. Particularly, the standard deviation in X and Y have increased from 0.2 units to 0.5 units per measurement. We observe a pattern seems to follow, loosely speeking the actual position of the turtle though, it jumps randomly quite a bit.

In Figure 4, we doubled the linear velocity error from 0.05 to 0.1.

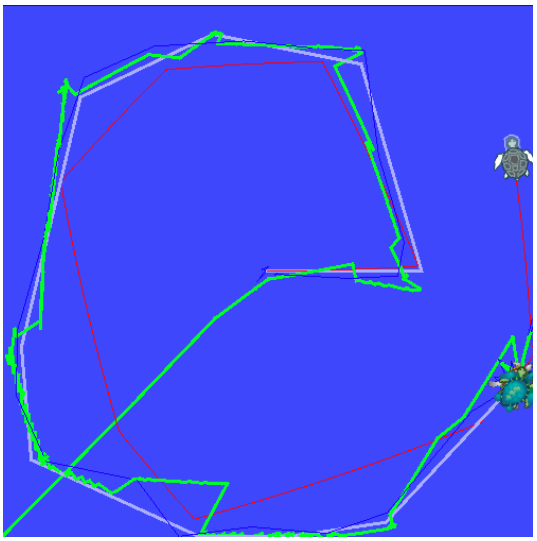

Figure 2: Baseline path

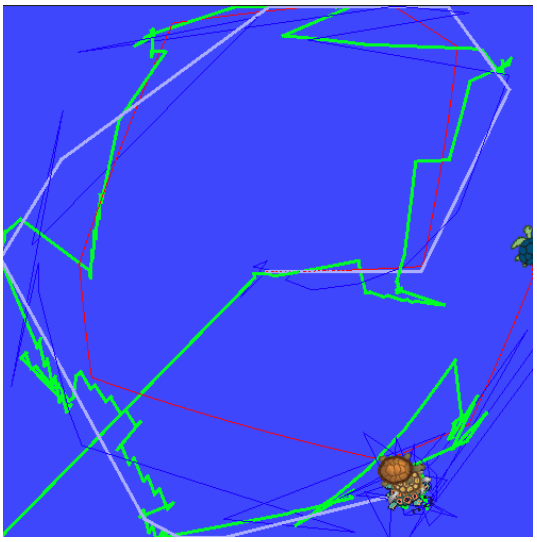

Figure 3: Path after position sensor noise

In Figures 5 and 6 we tested the behaviour of the system when changing sensors sampling rate and the frequency the 2 EKF nodes make an estimate. In Figure 5 the red line shows much greater drift from the odometry sensor with half the sampling rate 10Hz.

	Parameters	Node
1) Baseline	default parameters	-
2) Position Sensor Noise	-f 1. -x 0.5 -y 0.5 -t 0.2 -v	turtle1_positioning_system_node
3) Velocity Sensor Noise	-f 20. -x 0.1 -X 0. -t 0. -T 0.02 -v	turtle1_odometry_node
4) Sensors Sampling Rate	-f 1.8 -x 0.2 -y 0.2 -t 0.2 -v	turtle1_positioning_system_node
4) Sensors Sampling Rate	-f 10. -x 0.05 -X 0. -t 0. -T 0.02 -v	turtle1_odometry_node
5) EKF Frequency	20Hz	robot_localization_ekf_node_odom
5) EKF Frequency	5Hz	robot_localization_ekf_node_map

Table 1: Test cases

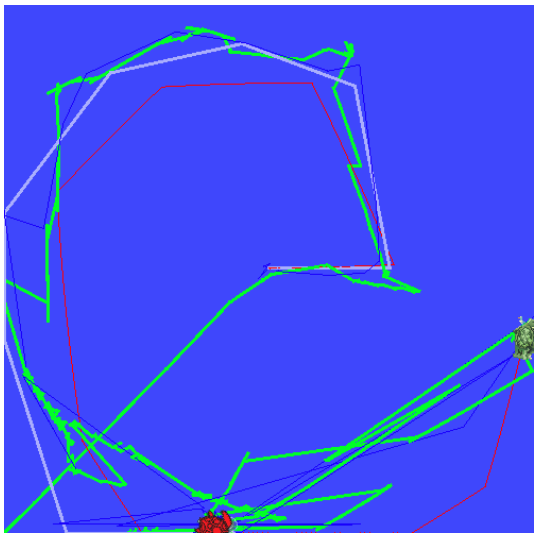

Figure 4: Path after velocity sensor noise

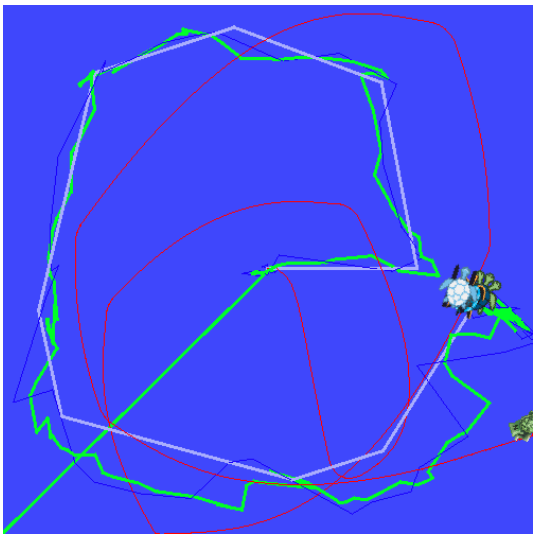

Figure 5: Path after sampling rate change

4 5.a

map-frame←odom-frame←base-link. Both *map-frame* and *odom-frame* are world fixed frames. Though, odometer will drift and accumulate error. To fix this we need to publish a map to odom transformation which will essentially fix the pose of the robot in the map frame.

5 5.b

References

[1] Thomas Moore and Daniel Stouch. *A Generalized Extended Kalman Filter Implementation for the Robot Operating System*. Sensor Processing and Networking Division Charles River Analytics, Inc. Cambridge, Massachusetts, USA.

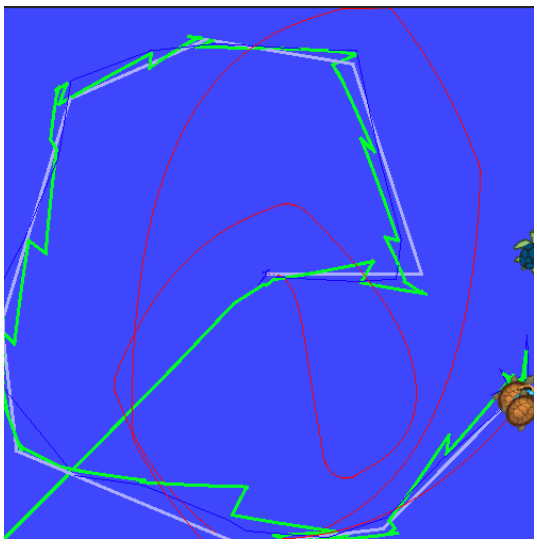

Figure 6: Path after EFK frequency change