B31YS Kalman Filter Assignment

For this assignment you will be required to code a Kalman filter for a ROSBOT 2. The following document provides with a tutorial on how to setup your environment.

After you are done you are required to submit:

i. A figure showing the values of the GPS and the corrected odometry using the Kalman filter.

ii. A screenshot of RViz showing the odometry estimation running

iii. The code you created.

Setting up your workspace

Clone the assignment repository to your workspace:

```
cd ~/catkin_ws/src git clone https://github.com/IgnacioCarlucho/B31YS_kalman_assi
gnment_ROS1
```

Install dependencies and build the workspace:

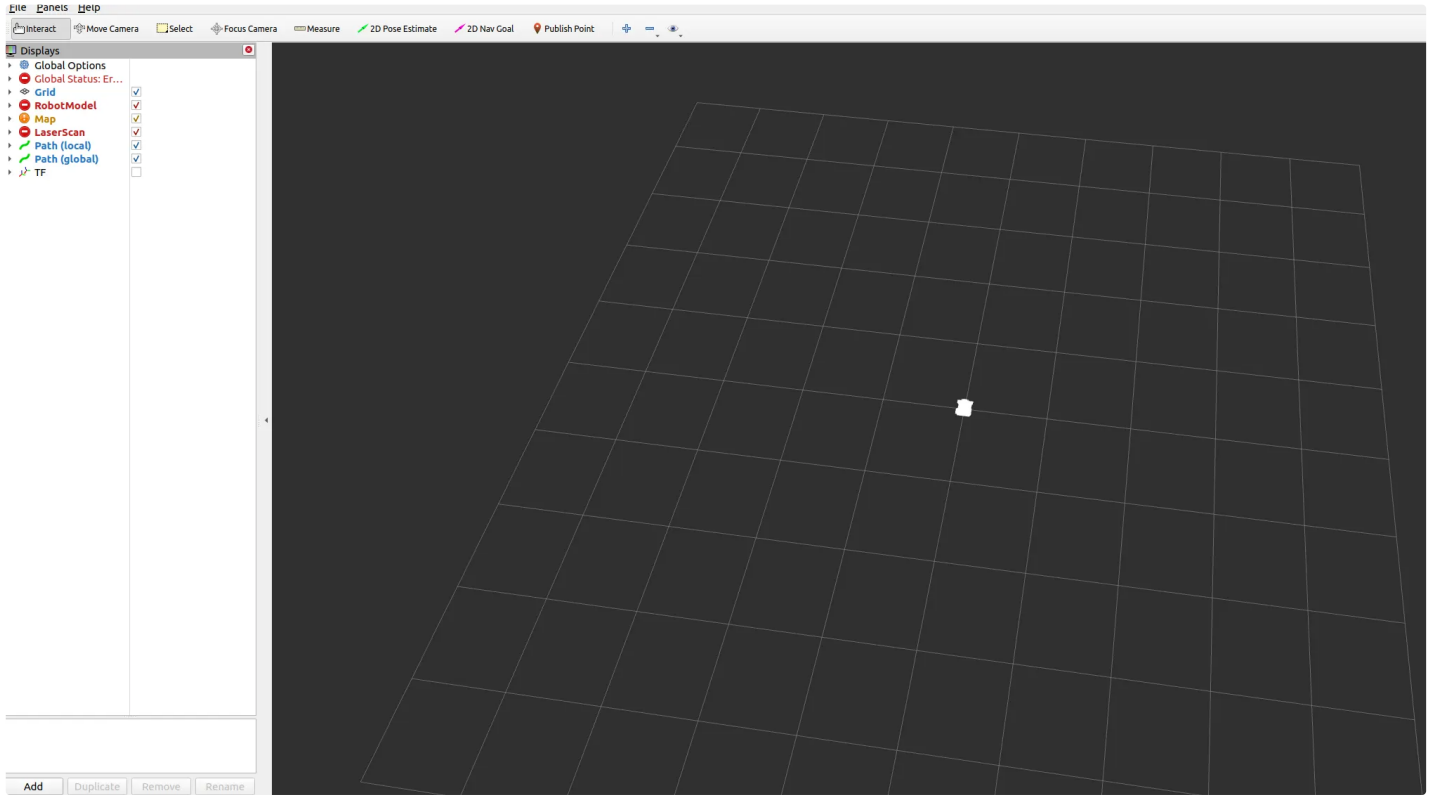
```
cd ~/catkin_ws catkin_make
```

Launching the robot:

Start the robot and sensors with:

```
roslaunch rosbot_bringup fake_gps_random.launch
```

The robot has a GPS that it is published in the gps topic, and a noisy odometry that it's published in odom1. Unlike the previous assignment, in this example we do not use a map. After launching you should see something like this:



The robot and all it's sensors are using the /odom frame for publishing. In RViz, change the global fixed frame to `/odom` to see the robot and sensors correctly.

Robot sensor overview

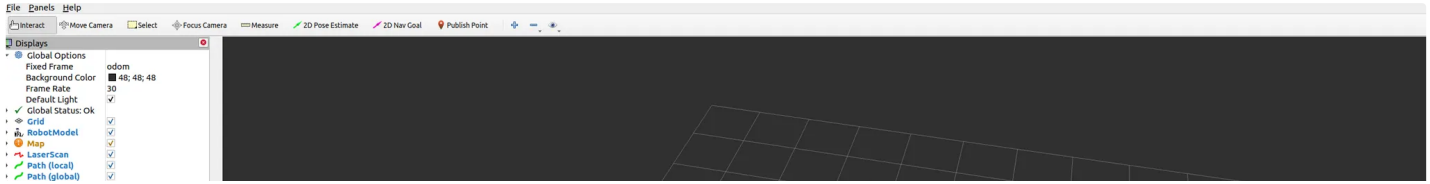
In this assignments the topics that you need to use are:

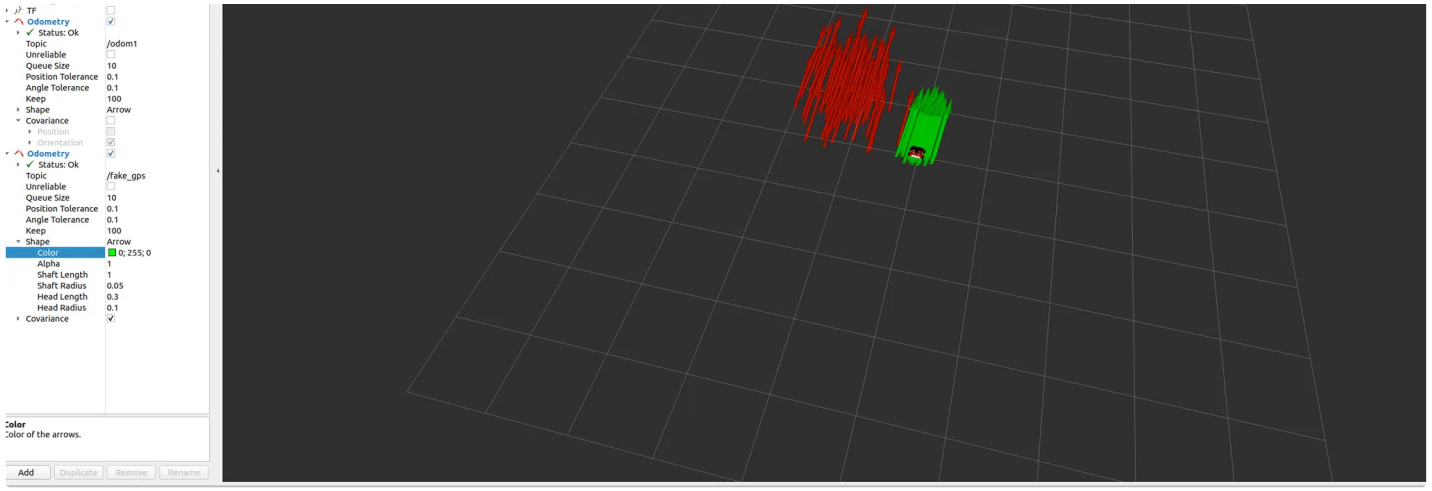
- GPS: This simulates a GPS sensor that gives a position estimate with low frequency. The position estimations of this sensor are meant to be accurate, but slow. It is important to note that this is not a typical GPS. This GPS is modified to give it's position in meters in the same frame as the odom. This simplifies the problem a lot, as GPS sensors usually give measurements in latitute and longitude, and a conversion is necessary to make this information useful locally. In this tutorial we are not so concerned with this, as primary goal is to understand how the Kalman filter works. So we have simplified it's operation.
- Command velocities: cmd_vel are the commands sent to the robot
- Odom1: This is a noisy odometry sensor. It is a "fake odometry" that we have modified to have a large std error, plus a drifting. This is is mean to simulate a cheap or old sensor that is unreliable. This is typically the case when you are using a low cost compass or IMUs. This sensor has high frequency.

There are two more topics that are available:

- Odom: This is a more reliable odometry estimation and you can use it as a ground truth.
- Imu: This is another sensor that is available and can be used to improve the Kalman filter estimation if desired.

Once the robot is running in Rviz you can add odometry to the visualizers in Rviz. So for example adding visualizers to Rviz of both odometries will show something like this:





You can play around with the visualization options until you get something that it is comfortable to you.

Kalman filter estimation

We will estimate the robot state as:

- State: [x, y, theta], with
  - x, y: position in meters
  - theta: orientation in radians

Kalman Filter Steps

1. **Prediction** – Use control input from `cmd_vel` (linear `v` and angular `ω`) to predict the next state.
2. **Correction** – Use GPS measurements (`x, y`) to correct the predicted state.

Assignment Instructions

We provide a template package. Run it with:

```
roslaunch kalman_filter_assignment kalman_execution.launch student_name:=YourName
```

The launch files calls two things:

```
<launch> <!-- 1. Launch Kalman Filter Template Node --> <node
pkg="kalman_filter_assignment" type="kalman_filter_template.py"
name="kalman_filter" output="screen"> <param name="student_name" value="$(arg
student_name)" /> </node> <!-- 2. Launch Trajectory node! --> <node
pkg="kalman_filter_assignment" type="student_trajectory.py"
name="student_trajectory" output="screen"> <param name="student_name"
value="$(arg student_name)" /> </node> <!-- Argument for student name --> <arg
name="student_name" default="student" /> </launch>
```

1. The first thing we call is the the script “kalman_filter_template”, in which you will have to code your own Kalman filter. Within that script, the most important function for you is this one:

```
def update_kalman(self, event): """ This is the main Kalman filter loop. In this
function you should do a prediction plus a correction step. """ # --- Prediction
--- x_pred = np.zeros((3,1)) # what goes here??? self.x = x_pred self.P = P_pred
# --- Correction --- self.x = 0 # what should go here? self.publish_estimate()
```

2. The second thing the launch file calls is a trajectory node that makes the robot move in a specific way. This important because you want the robot to be moving to understand it's position! This is a special node that takes as input your name and based on that commands the robot to move making in a specific way. infinite sequence

Assignment: Kalman filter estimation

BEng Student Tasks:

- Complete the update_kalman to:
  - Update using differential drive kinematics
  - Correct state with GPS
- Tune the process noise Q and measurement noise R

MSc Students Tasks:

- Complete the update_kalman to:
  - Update using differential drive kinematics
  - Correct state with GPS and odom1
- Tune the process noise Q and measurement noise R

Submission requirements

Run the code using:

```
roslaunch kalman_filter_assignment kalman_execution.launch student_name:=YourName
```

1. A PDF containing:
    a. A 2D plot showing X vs Y trajectories for GPS, Kalman, and ground truth
        i. Or: A 2D plot showing X-axis: Time (seconds); Y-axis: Position (meters); And three lines: GPS measurements (sparse), Kalman estimate (smooth), Ground truth (reference).
    b. A screenshot of RViz showing the odometry estimation running
2. The code you created.

Notes

- Prediction should run at **every loop**, based on `cmd_vel` .
- Correction is applied whenever a GPS measurement is available. But if you are also using odom1 you can update on every odom update.
- Keep the code simple and use only `numpy` functions (`np.array` , `np.dot` , etc.).