Coding exercise 2 for CS 498 GC

- ☐ This homework consists of a written and coding part. Both must be submitted individually at Gradescope.
- ☐ The coding part must be developed in python with ROS2. When submitting this section, only upload the python file *coding_ex2.py* (see attached template).
- ☐ The written part must be submitted in PDF format. When submitting this file, Gradescope will ask you to select the pages where each question is. Please do it precisely.

Learning outcomes

- ☐ Understand Kalman filters and EKF
- ☐ Implement Extended Kalman filter software in Python and ROS
- ☐ Present report and analyze results for robustness
- ☐ Present code and get evaluated on style and readability in code to improve your coding skills as a roboticist.

Description

In coding exercise 1, we found that the trajectory estimated by wheel odometry significantly differs from the ground-truth trajectory, as it accumulates drift over time. In this coding exercise, we are going to implement a more robust algorithm, by fusing GPS and inertial measurements with an Extended Kalman Filter (EKF). While GPS provides position measurements with a certain accuracy, it does not provide information about the orientation of the robot. By integrating the inertial measurements, we can even estimate subtle changes in position and orientation. We are going to use the high-rate inertial measurements in the prediction step and the low-rate GPS measurements in the correction step.

GPS-INS sensor fusion is a key and critical part of mobile robot localization, especially outdoors. We have studied in class in detail the operational principles of GPS and INS, we have also derived the mathematical formulations for Bayesian sensor fusion problems and its extension to this problem through the Extended Kalman Filter. The goal in this lab is to bring all of this to practice by learning to create high-quality real-time implementable software for GPS-INS fusion.

We will use the same rosbag file as in coding exercise 1, where the robot makes a double loop around a house. Figure 1 shows the ground-truth path.

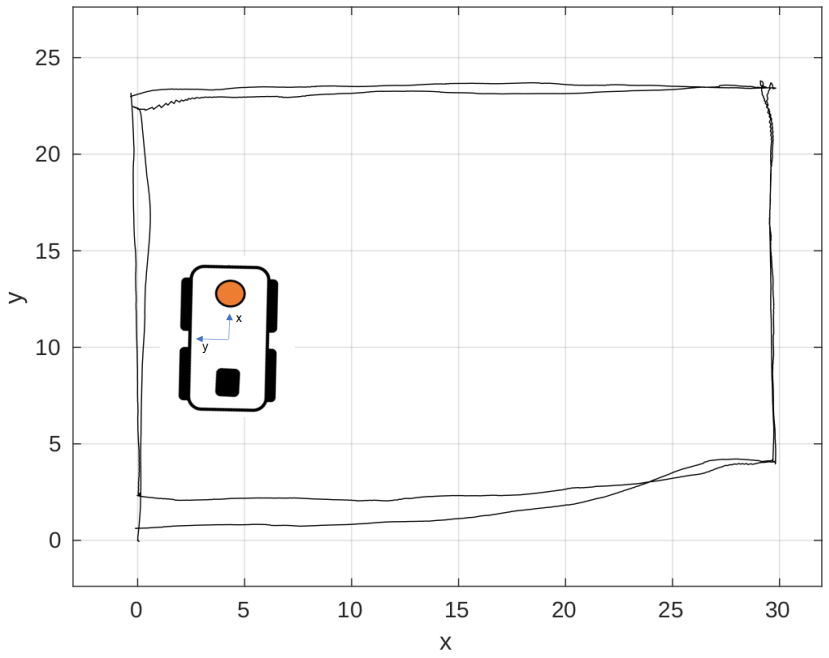


Figure 1. Ground-truth trajectory

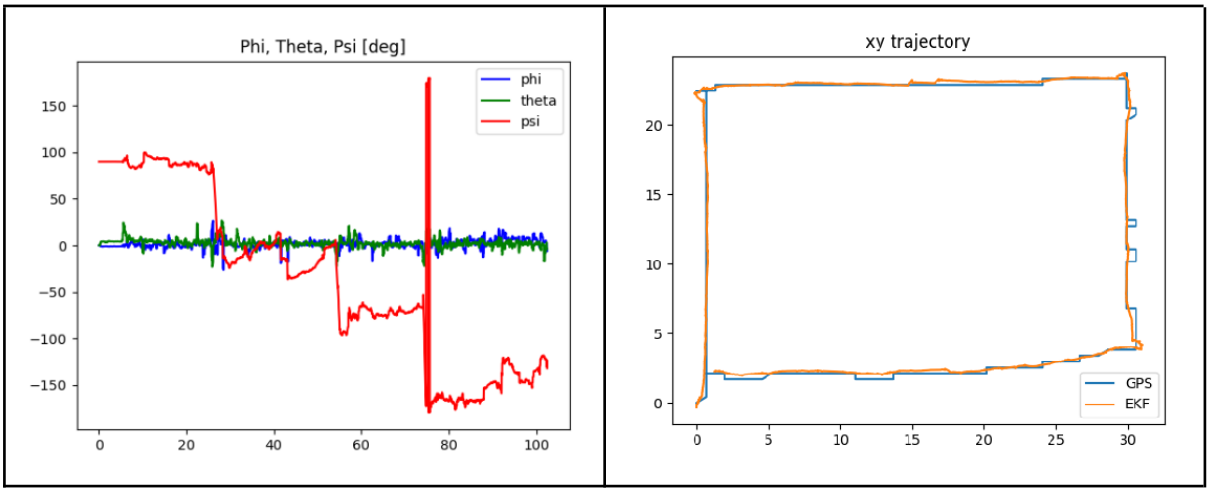
1. Play the rosbag file *solar_house_v3* and identify the publishing rate of the IMU and GPS measurements (add results to your report). Check if the sample time (self.dt) in the main template is correct.
2. Complete the callback functions *callback_imu()*, *callback_gps_speed_east()*, and *callback_gps_north()*. Make sure you assign these measurements to the vector *measure*[] in the order below.

measure [0]	Angular velocity around x axis
measure [1]	Angular velocity around y axis
measure [2]	Angular velocity around z axis
measure [3]	Linear acceleration in x axis
measure [4]	Linear acceleration in y axis
measure [5]	Linear acceleration in z axis
measure [6]	GPS position x
measure [7]	GPS position y
measure [8]	Position z
measure [9]	GPS velocity x
measure [10]	GPS velocity y
measure [11]	Velocity z

3. Complete the function *ekf_function*(). Follow the instructions given in the template and the equations of the section GPS-INS of the lecture.
4. Plot the estimated xy trajectory, euler angles, gyroscope bias, accelerometer bias, and covariance of position ((add results to your report).
5. Create an odometry publisher that publishes the estimated position and orientation of the robot. Visualize the trajectory on rviz2 (add a screenshot to your report).
6. Evaluate the performance of the Kalman filter for different values of Q and R matrices, in particular, try the following different cases and ***qualitatively*** describe what happens to the estimated trajectory with respect to the GPS trajectory. Is the filter fairly robust to Q and R? (add xy plots and description to your report).

   *R=0.01*Roriginal*
   *R=100*Roriginal*
   *Q=0.01*Qoriginal*
   *Q=100*Qoriginal*
7. Set the R_GPS to zero (I.e. assume the GPS is mounted at the CG), what happens to the position estimates and the attitude estimates? Explain why it happens in your report.

Sample outputs



Appendix

Tutorial 1. Building a ROS package

This tutorial is intended for ubuntu 22.04 and ROS2 Humble. For further information go to the official tutorials, where you can find other alternatives.

ROS Humble. https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html

ROS Foxi. https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html

- Create a ROS workspace if you have not created one yet

```
>> source /opt/ros/humble/setup.bash
>> mkdir -p ros2_ws/src
```

- Create a package in your ROS workspace named *mobile_robotics*.

```
>> cd ros2_ws/src
>> ros2 pkg create --build-type ament_python mobile_robotics
```

- Put the python templates in the directory …ros2_ws/src/mobile_robotics/mobile_robotics/
- Edit the *package.xml* file adding the dependencies rclpy, std_msgs, geometry_msgs, nav_msgs, tf2_ros and sensor_msgs

```
<license>TODO: License declaration</license>
  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>geometry_msgs</exec_depend>
  <exec_depend>nav_msgs</exec_depend>
  <exec_depend>tf2_ros</exec_depend>
  <exec_depend>sensor_msgs</exec_depend>
```

- Add a new console script in the file *setup.py* of your package as follows

```
'console_scripts': [
        'part1 = mobile_robotics.coding_ex1_part1:main',
        'cod_ex2 = mobile_robotics.coding_ex2:main',
    ],
```

- Build your package

```
>> sudo apt install python3-colcon-common-extensions //if you have not installed it yet
>> cd ros2_ws
>> colcon build --symlink-install --packages-select mobile_robotics
//with symlink-install, you only have to build the package once!
```

- Source your ROS workspace and local workspace in a new terminal and run your node

```
>> cd ros2_ws
>> source /opt/ros/humble/setup.bash
>> source install/setup.bash
>> ros2 run mobile_robotics cod_ex2
```

- You can play the rosbag file in other terminal

```
>> ros2 bag play solar_house_v3
```