Project Section

Robot Localization: Extended Kalman Filter in ROS 2

Objectives

- Implement an EKF in a Python package in ROS 2
- Run a localization simulation in Gazebo
- Understand the working principle of the EKF and the influence of parameters on the performance of the localization system

Simulation Environment

For this project, you must use the default world of the TurtleBot3 package that can be loaded using `gazebo.launch.py` as you did in Lab03.

Choose an initial pose, in which the robot is spawned when the simulation starts. Take note of this position because this will be the initial guess of your EKF node.
Hint: Use the launch arguments `x_pose`, `y_pose` and `yaw` to spawn the robot in the correct position.

The **white columns** in the TurtleBot3 worlds are the **landmarks** considered for this environment.

Use the `turtlebot3_teleop teleop_keyboard` node to control the robot motion with your keyboard.

Implement the EKF inside a ROS 2 package

These are suggested steps, you are free to choose another way of implementing the EKF with a ROS 2 interface.

1. Create the package

Create a ROS 2 Python package to contain your code.

2. Create the Python files containing the equations

Take as reference the Jupyter Notebook you used in Lab04, then:

- Define in a Python file inside your package, e.g. `ekf.py`, the class RobotEKF implemented in the notebook. The class must include (at least) the fundamental predict and update methods.
- Define in a Python file inside your package, e.g. `motion_models.py`, the equations of the motion models that are also defined in the notebook, i.e. Velocity m.m. and Odometry m.m.

- define in a Python file inside your package, e.g. `measurement_model.py`, the equations used for the range and bearing measurement model. Include also the function `z_landmark` that will be used to generate the simulated measurements and the `residual` function.

N.B: The Python file must be put inside the folder with the `__init__.py`. It is not necessary to define a main() function in these three Python files, they are used just to import the functions and classes in your ROS 2 Node.
The final file structure of your package, after these steps, should resemble this one:

```
.
├── localization_project
│   ├── __init__.py
│   ├── measurement_model.py
│   ├── motion_models.py
│   └── ekf.py
├── package.xml
├── resource
├── setup.cfg
├── setup.py
└── test
```

Create a ROS 2 Node to run the filter

Create a ROS 2 node to run your filter. Import all the necessary functions from the Python files you wrote in the previous step like a Python module, an example is reported below.

```
from localization_project.ekf import RobotEKF
```

The required node must run the EKF at a fixed rate, performing the prediction step and the update step with all the visible landmarks. At each iteration of the filter, the node must publish the result, including the covariance, on a ROS 2 topic called `/ekf`.
To perform the update step of the filter, you must simulate the measurements using the `z_landmark` function, using the ground truth of the robot and the list of landmark positions.

The node must have the following interfaces:

- **Subscriptions**:
  - `/ground_truth`
  - `/diff_drive_controller/odom`
- **Publishers**:
  - `/ekf`, message type `nav_msgs/msg/Odometry`
- **Parameters**:
  - `ekf_period_s`: the period of the EKF
  - `initial_pose`, float[ ]: initial guess for the EKF
  - `initial_covariance`, float[ ]: covariance of the initial guess

- `landmarks`, float[ ]: sequence of XY coordinates of the landmarks in the world's reference frame, e.g. [x1, y1, x2, y2, …]
- `std_rot1`, `std_transl`, `std_rot2`: noise parameters for Odometry m.m.
- `std_lin_vel`, `std_ang_vel`: noise parameters for Velocity m.m.
- `std_rng`, `std_brg`: noise parameters for landmarks measurement model
- `max_range`, `fov_deg`: parameters for landmarks measurement simulator (`z_landmark` function)

Experiments

Set the measurement parameters to the following values:

- `std_rng`: 0.3
- `std_brg`: 1°
- `max_range`: 8m
- `fov_deg`: 45°

Using the turtlebot3 teleop_key node, guide the robot around the world while registering and plotting the available odom topics (the standard diff_drive odom topic, the filtered odom topic and the ground truth robot pose).

1. Analyze both numerically and graphically the results you get from the localization system. You can use the same metrics required for Lab05.
2. Try to perform a tuning process of the noise parameters of your filter to optimize its performance, assuming that the measurement parameters are fixed.
3. Provide significant comments related to the performance of your system.