Lab04

Localization with EKF

Objectives

- Develop a ROS 2 package to run EKF on the robot.
- Localize your robot using landmark measurements.
- Fuse measurements from different sensors (odometry, IMU) to improve localization accuracy.

Requirements

- Concepts and examples explained during lectures (motion models, sensor models, EKF)
- Package `turtlebot3_simulations`
- Package `turtelbot3_perception`

Install turtlebot3_perception package

1. Download the **`turtlebot3_perception`** package from GitHub

```
git clone https://github.com/SESASR-Course/turtlebot3_perception.git
```

2. Follow the instruction in the Installation section of the README.md

Exercise

Main objective: realize a ROS 2 package for EKF localization. The package will be validated in Gazebo simulation and then deployed on the robot.
The lab exercise is divided into 4 Tasks. Tasks 0, 1 and 2 can be prepared before the experimental session in the lab using the Gazebo simulation.

Suggestion: if you are running out of time in the lab, skip Task 2 and try your solution (Task 1) on the real robot (Task 3).

Task 0 [1 point]

Implement probabilistic models with dedicated Python functions.

- **Probabilistic velocity-based motion model (sampling)**:
    - It should receive the current state x, the command u, and the noise numerical coefficients (or std deviation) as arguments.
    - It should return a new pose x' as a numpy array.
    - Make 500 samples of the next pose from an initial state $[x_0, y_0, \theta_0]$ with a fixed given command $[v, \omega]$ using at least two different sets of noise parameters (one to highlight uncertainty on the angular motion, one on the linear motion)
    - Plot the obtained sample poses
    - Compute the Jacobians of the non-linear motion model G, V with respect to both the state and the command variable
- **Probabilistic measurement model - landmark model (sampling)**:
    - It should receive in input the state x, the measurement z, and the noise numerical coefficients (or std deviation) as argument;
    - It should return a numpy array with range and bearing of the detected landmark
    - Make 1000 samples of the pose from an initial initial state $[x_0, y_0, \theta_0]$ and a given measurement $z = [r, \phi]$
    - Plot the resulting distribution of sampled poses
    - Compute the Jacobian H of the model over the state x

Task 1 [3 points]

Starting from the EKF scripts that were presented during lectures, implement inside a ROS 2 node an EKF for tracking the robot position given landmarks measurements.

State of the filter: the state of the filter is the one reported below, where x, y, θ represent the position of the robot in the global reference frame.

$$\mu = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Prediction: perform the prediction step at a **fixed rate** of 20 Hz using the **velocity motion model**. Use as **command** the most recent v and ω taken from the `/odom` **topic**. Data is used in this way because the velocity reported by the robot is closer to the real velocity than the command sent on `/cmd_vel` topic. **Create a timer in your node to perform the prediction operation of EKF.**

Update: measurements are provided as **range and bearing** of **landmarks** in the field of view of the robot. Measures are published on **topic** `/landmarks` (or `/camera/landmarks` on the real robot) inside a message of type `landmark_msgs/msg/LandmarkArray`.

Table 1. Landmarks coordinates

id	11	12	13	21	22	23	31	32	33
x	-1.1	-1.1	-1.1	0.0	0.0	0.0	1.1	1.1	1.1
y	-1.1	0.0	1.1	-1.1	0.0	1.1	-1.1	0.0	1.1
z	0.5	1.5	0.5	1.0	0.75	0.3	1.5	1.0	0.0

Perform the update operation of the EKF inside the subscription callback for each landmark listed in the message.
Once all the updates are done, **publish** the estimated state in a message of type `nav_msgs/msg/Odometry` on topic `/ekf`. Make sure to **fill** the `header.stamp` field with the current time taken from `self.get_clock().now().to_msg()`.

Landmarks coordinates are provided in Table 1 and in a yaml file inside the `turtlebot3_perception` package (`turtlebot3_perception/config/landmarks.yaml`).

Task 2 [1 point]

Extend the state of the filter to also include linear and angular velocity. **The new state is the one reported below.**

$$\mu = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix}$$

Prediction: create **new functions to predict** the state and to compute **its Jacobians G** and **V**. It is reported here the function $g(u, x)$ and its Jacobian with respect to the new state $G(u,\, x)$. You can notice that the part of the Jacobian in the red box is the Jacobian matrix you were using in the first version of the filter.

$$\begin{bmatrix} x' \\ y' \\ \theta' \\ v' \\ \omega' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix} + \begin{bmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t \\ 0 \\ 0 \end{bmatrix} + \mathcal{N}(0, R_t)$$

$$\begin{bmatrix} 1 & 0 & -\frac{v\cos(\theta)}{w} + \frac{v\cos(dtw+\theta)}{w} & -\frac{\sin(\theta)}{w} + \frac{\sin(dtw+\theta)}{w} & \frac{dtv\cos(dtw+\theta)}{w} + \frac{v\sin(\theta)}{w^2} - \frac{v\sin(dtw+\theta)}{w^2} \\ 0 & 1 & -\frac{v\sin(\theta)}{w} + \frac{v\sin(dtw+\theta)}{w} & \frac{\cos(\theta)}{w} - \frac{\cos(dtw+\theta)}{w} & \frac{dtv\sin(dtw+\theta)}{w} - \frac{v\cos(\theta)}{w^2} + \frac{v\cos(dtw+\theta)}{w^2} \\ 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Update: beside landmarks detection model `ht_landmark`, **consider wheel encoder** data and **IMU data** to correct the v, ω components of the state.
Use the wheel encoder data (in `/odom`) to update both v and ω with a function `ht_odom`, and the IMU to only update ω with a function `ht_imu`.
Measurement functions in matrix form are reported below, compute the associate Jacobian `Ht_odom`, `Ht_imu` with respect to the new state, necessary for the correction step of the filter. Also, update the Jacobian of the landmark measurement function `Ht_landmark` to consider the new state. Based on the expected confidence in the measurements, define the uncertainty parameters `std_odom` and `std_imu` to define the covariance matrix ***Q*** and to sample from the new measurement models.

$$h_{odom}(\bar{\mu}_t) = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \mathcal{N}(0, Q_{odom}) \ , \quad h_{imu}(\bar{\mu}_t) = \begin{bmatrix} \omega_t \end{bmatrix} + \mathcal{N}(0, Q_{imu})$$

Task 3 [1 point]

Run the ROS 2 package for EKF localization on the real robot. You can choose between the implementation used in Task 1 and the one developed for Task 2.

Report requirements

- Provide a concise description and comments on the main structure of your ROS 2 program (do not copy your code in the report, just highlights the main elements and the workflow: nodes, publishers/subscribers to relevant topics and parameters)

- **[Task 0]: Run your Python function with probabilistic models in dedicated Python scripts**. Then:
    - Report the plot (x, y) of the obtained samples poses with the velocity motion model in the two case of noise parametrization, also plotting the current pose of the robot
    - Report the plot (x, y) of the obtained samples poses with the landmark measurement model
    - Comment the results obtained with different uncertainty values
    - Report the expression of the Jacobian matrix obtained for the motion model (G, V) and the measurement model (H), and the resulting linearized expression of both models. (Examples on how to compute literal expression with `sympy` provided in `python-crash-intro` repository Module4)

- **[Task 1]**: **Run your filter in the simulation environment.** Record the `/ground_truth`, `/odom` and `/ekf` topics. Then:
  - **Make plots** comparing the position and orientation reported in the three topics
    - Make a plot for each state (x, y, θ) vs. time and compare the different topics. You shall put the state reported by all the three topics in the same plot. (Examples provided in python-crash-intro repository Module3)
    - Plot the trajectory on the 2D plane using the robot's pose data from the three topics.
    - Comment the results
  - **Compute the following metrics** with respect to the ground truth: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). Use the tools provided in **04_Metrics from rosbag.zip** that you can download from Portale della Didattica.

- **[Task 2]**: perform the same experiments and obtain the same plots and performance results of Task 1 for the extended EKF state with sensor fusion described in Task 2. Make some significant comparison comment on the different results obtained.

- **[Task 3]**: **Run your filter on the real robot.** You can choose between the implementation of the EKF you developed for Task 1 or Task 2. Record the `/odom` and `/ekf` topics. Then:
  - Make plots comparing the position and orientation reported in the two topics. For `/odom` you can subtract the first pose to all the other poses to align to the origin. Then:
    - Make a plot for each state component over the time length of the experiments, and compare the different topics.
    - Plot the (x, y) trajectory using the data collected by the two topics.
    - Comment on the results.

How to test your algorithms

Simulation Environment

The following operations shall be executed on your PC.

A simulation is provided to ease the development and testing of your code. To launch the simulation run the following command in a workspace with both `turtlebot3_simulations` and `turtelbot3_perception` packages. (Mac users should run the `ignition` version of the command in the same way adopted for Lab02)

```
ros2 launch turtlebot3_gazebo lab04.launch.py
```

The simulation environment is the one shown in Figure 1, with the **white columns** acting as **landmarks**. The orange labels are the landmarks IDs.

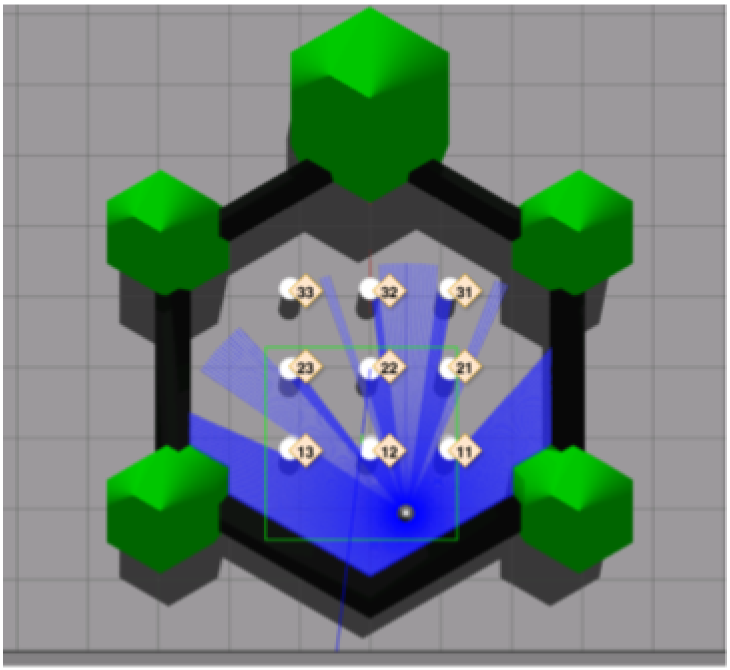

Figure 1. Simulation environment with labelled landmarks

Real Robot

The following operation shall be performed on the robot to start landmark detection.

1. In two different terminals start the camera driver and the landmark detector using the following commands

```
# Terminal 1
ros2 launch turtlebot3_perception camera.launch.py

# Terminal 2
ros2 launch turtlebot3_perception apriltag.launch.py
```

2. If landmarks are present, they will be published on topic `/camera/landmarks` at approximately 6 Hz.