Lab04

Localization: Extended Kalman Filter in Python

Objectives

- Implement the equation of the EKF in a Python script
- Run a localization simulation with the implemented script
- Understand the influence of parameters of the EKF

Installation

Install Jupyter Notebook and some Python modules

From Ubuntu, open a terminal and run the following commands to install Jupyter Notebook and some dependencies.

```
$ sudo apt update
$ sudo apt install python3-pip
$ pip3 install notebook numpy sympy matplotlib ipympl
```

Install extensions to use Jupyter Notebook from VS Code

Open VS Code and from the Extension Tab install the extension “ms-toolsai.jupyter”.

Download the material

Download the file `lab04_material.zip` from Portale della Didattica.

Exercise

Task 1

Open `ekf_students.ipynb` and follow the instructions in the notebook that guide you through the implementation of an EKF using Python.

In the first section of the notebook, you can find some entry-level examples about using matrix and numerical operations in Python (NumPy, SymPy,...)

Then, you will find the concrete exercise of this lab, starting from the definition of the motion model for robot localization.

For a working program, you should complete the notebook where indicated, in particular, you have to write:

- The motion model matrix gux **= ...**
- The Jacobians *G* and *V* of the motion model
- The function *h* of the measurement model, and its Jacobian *H*

- The predict method of the EKF
- The update (correction) method of the EKF

Once you complete the notebook and the final output matches the one reported in the *expected_output.png* go on with the following task.

Report: comment on the main lines that define your EKF (predict, update, motion and measurement models).

Task 2

Using the provided landmarks and simulation time, run the simulation using different combinations of uncertainty for the measurements.

Report: Try at least the following values and comment on the results (numerically and graphically).

- **std_range**: 0.1, 0.3, 0.5 [m]
- **std_bearing**: 0.5, 1.0, 5.0 [deg]

Task 3

Using the provided landmarks and simulation time, run the simulation using different combinations of uncertainty for the motion model.

Report: Try the following values and comment on the results (numerically and graphically).

- **std_lin_vel**: 0.05, 0.1, 0.5 [m/s]
- **std_ang_vel**: 0.1, 1.0, 5.0 [deg/s]

Task 4

Using the provided landmarks, simulation time and uncertainties (**std_lin_vel**=0.1 m/s, **std_ang_vel**=1.0 deg/s, **std_range**=0.3 m, **std_bearing**=1.0°) change the frequency at which the EKF is computed.

Report: Try at least the following values and comment on the results, (numerically and graphically).

- **ekf_dt**: 0.2, 0.5, 1.0 s

Task 5

Change the landmarks with the configurations reported below and run the simulation with the standard parameters, as described in Task 4, and **ekf_dt=**1.0.

Report: Comment on the performance of the filter for each landmark configuration.

- **Dense landmarks** (default): [[5, 12], [10, 5], [15, 15], [10, 14], [6,6], [12,9]]

- **Sparse landmarks**: [[5, 12], [15, 15], [12,9]]
- **Sparse landmarks with gap**: [[5, 12], [15, 18], [6,6]]

Task 6

Change the motion model used by the EKF in an Odometry Model following the instructions in the notebook.

- Implement the needed equations using SymPy and convert them to functions that can be evaluated with numeric values
- Create a function that given the current and previous odometry measurements returns the inputs for the Odometry motion model $\left[\delta_{rot1}\ \delta_{trasl}\ \delta_{rot2}\right]$
- Use the new model with the `RobotEKF` class present in your notebook
- Choose suitable parameters for the process noise covariance matrix.
- Run the filter with the standard parameters as in Task 1 and check the correspondence with the expected output.

Report: comment on the main lines that define your odometry motion model. Describe how you defined the process noise covariance matrix. Include the output plot.

Task 7

Perform again *Task 4* with a non-constant velocity (e.g. changing the input every N steps of the simulation), running both the filter with the velocity motion model and the filter with the odometry motion model.

- Which one performs better? Why?
- How does the performance of the filter change, changing the EKF frequency?

Report: answer the previous questions motivating your response with some graphical and numerical examples.