
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-521 — Mobile Robotics

# Trabajo Práctico

EKF and PF for Localization of a Mobile Robot

## 1. Introduction

The objective of the practical work is to understand how the Extended Kalman Filter (EKF) and the Particle Filter (PF) work for localizing a mobile robot and to develop an implementation of each.

Useful reading materials: lecture slides, Chapters 3, 4, 5, 7, and 8 of the Probabilistic Robotics book, by Thrun, Burgard, and Fox.

## 2. Submission

- A Git repository containing the developed code, a Docker image, and a `README.md` file with compilation and execution instructions must be provided.
- A report must be submitted in Lyx or LaTeX explaining the work performed and analyzing the results obtained.

# 4. Programming Exercises

Implement an Extended Kalman Filter (EKF) and a Particle Filter (PF) to localize a robot using *landmarks*. We will use the odometry-based motion model derived previously. We assume landmarks are present in the robot's environment. The robot receives the angles (*bearing*) to the landmarks and the landmark IDs as observations: (orientation, landmark ID).

We assume a noise model for the odometry motion model with parameter $\alpha$ (see Probabilisitc Robotics Book Table 5.6) and a separate noise model for the angle observations with parameter $\beta$ (see Probabilisitc Robotics Book Section 6.6). The landmark ID in the observations is noise-free. See the provided code for implementation details.

At each time step, the robot starts from the current state and moves according to the control input. The robot then receives an observation of the world. This information is used to localize the robot over the entire time sequence with an EKF and PF.

## 4.1. Code description

The provided code is written in Python and depends on the `NumPy` and `Matplotlib` libraries.

- `localization.py` — main entry point for running experiments.
- `soccer_field.py` — Implements the dynamic and observation model functions, as well as the noise models for both. Implement Jacobians here!
- `utils.py` — Contains several plotting functions, as well as a useful function for normalizing an angle between $[-\pi, \pi]$.
- `policy.py` — Contains a simple policy that you can safely ignore.

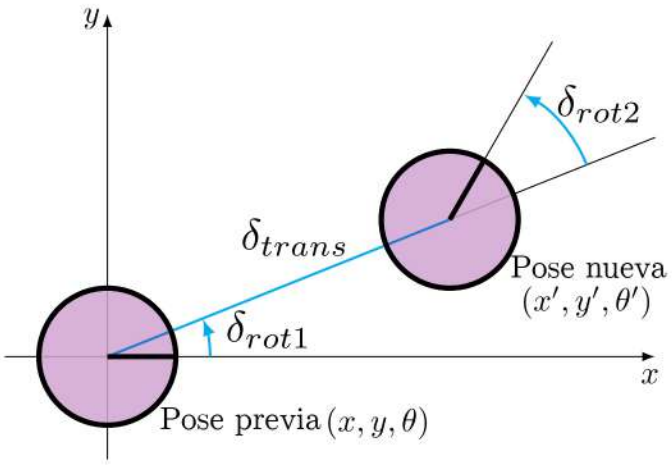


Figura 1: Motion model

- `ekf.py` — Implement the Extended Kalman Filter here!
- `pf.py` — Implement the Particle Filter here!

## 4.2. Command Interface

To visualize the robot in the soccer field environment, run

```
python localization.py --plot none
```

The  line plots the robot's position (ground-truth), which is the result of noisy actions. The  line plots the robot's position assuming the actions are not noisy (desired trajectory, but not the actual one). After implementing a filter, the robot's position estimated by the filter will be plotted in .

```
python localization.py --plot ekf
python localization.py --plot pf
```

To see the available command-line flags, run

```
python localization.py -h
```

## 4.3. Data Format

- state: $[x, y, \theta]$
- control: $[\delta_{rot1}, \delta_{trans}, \delta_{rot2}]$
- observation: $[\theta_{bearing}]$

## 4.4. Notes

- Call `utils.minimized_angle` whenever an angle or angle difference might exceed $[-\pi, \pi]$.
- Use the low-variance systematic sampler seen in class. It gives a smoother particle distribution and also requires a single random number per resampling step.
- Disable visualization to speed up execution.

**Ejercicio 3.** Implementing EKF

Implement the Extended Kalman Filter algorithm in `ekf.py`. You will need to complete `ExtendedKalmanFilter.update` and the $G$, $V$, and $H$ methods. The results of a successful EKF implementation should be comparable to the following results.

```
python localization.py ekf --seed 0
...
Mean position error: 8.9983675360847
Mean Mahalanobis error: 4.416418248584298
ANEES: 1.472139416194766
```

**a)** Plot the actual robot path and the path estimated by the filter with the default parameters (provided).

**b)** Plot the mean position error as the factors $\alpha$ and $\beta$ vary over $r = [1/64, 1/16, 1/4, 4, 16, 64]$[1] and analyze the results obtained.

Run 10 trials per value of $r$. An execution might look something like this:

```
python localization.py ekf --data-factor 4 --filter-factor 4
```

**c)** Plot the mean position error and ANEES (*average normalized estimation error squared*) as the filter factors $\alpha$, $\beta$ vary over $r$ (as above) while the data is generated with the default values. Analyze the results.

**Ejercicio 4.** Implementing PF

Implement the Particle Filter algorithm in `pf.py`. You must complete `ParticleFilter.update` and `ParticleFilter.resample`.

```
python localization.py pf --seed 0
...
Mean position error: 8.567264372950905
Mean Mahalanobis error: 14.742252771106532
ANEES: 4.914084257035511
```

**a)** Plot the actual robot path and the path estimated by the filter with the default parameters.

**b)** Plot the mean position error as the factors $\alpha$, $\beta$ vary over $r$ and discuss the results.

**c)** Plot the mean position error and ANEES as the filter factors $\alpha$, $\beta$ vary over $r$ while the data is generated with the default values.

**d)** Plot the mean position error and ANEES as the factors $\alpha$, $\beta$ vary over $r$ and the number of particles varies over $[20, 50, 500]$.

[1] Since the factors multiply with variances, this is between 1/8 and 8 times the default noise values.