Machine Learning for Probabilistic Robotics with Webots

Joan Gerard[1]
Promotor: Prof. Gianluca Bontempi [1]

[1]Université Libre de Bruxelles

March 9, 2019

Table of Contents

Table of Contents

Robot Positioning Problem

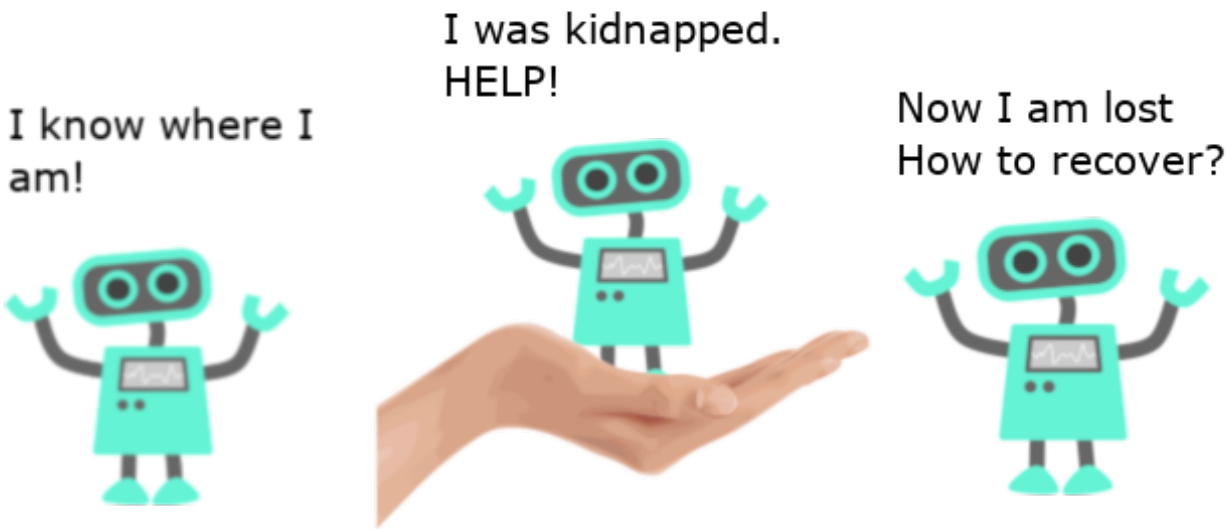
- A robot needs to deal with uncertainty most of the time.
- It needs to predict and preserve its current position and orientation within the environment.

Robot Positioning Problem

- Navigation is about controlling and operating the course of a robot and it is one of the most challenging skills that a mobile robot needs to master in order to successfully moving through the environment.
- Succeeding in navigation means to succeed likewise in:
  - Sense
  - Localization
  - Cognition
  - Control Action

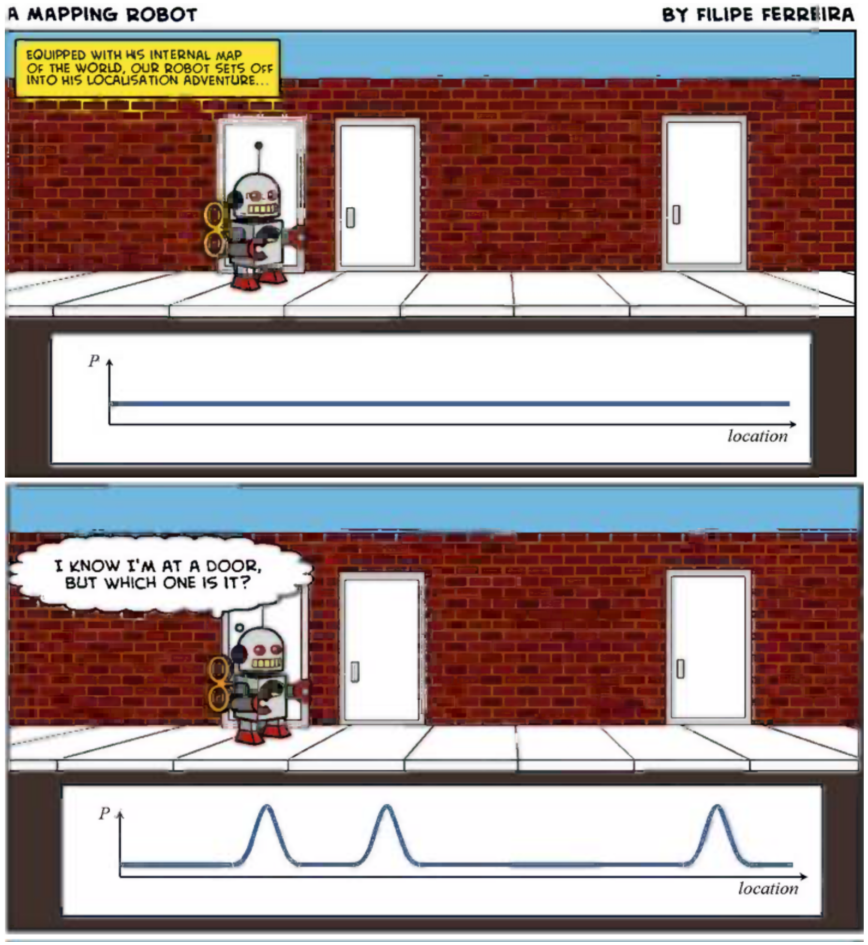
Robot Positioning Problem

- Local Robot Positioning (position tracking): The robot knows its initial position.
- Global Robot Positioning: The robot does not know its initial position and it is able to estimate it even in a global uncertainty. Ex. Kidnapped robot problem.

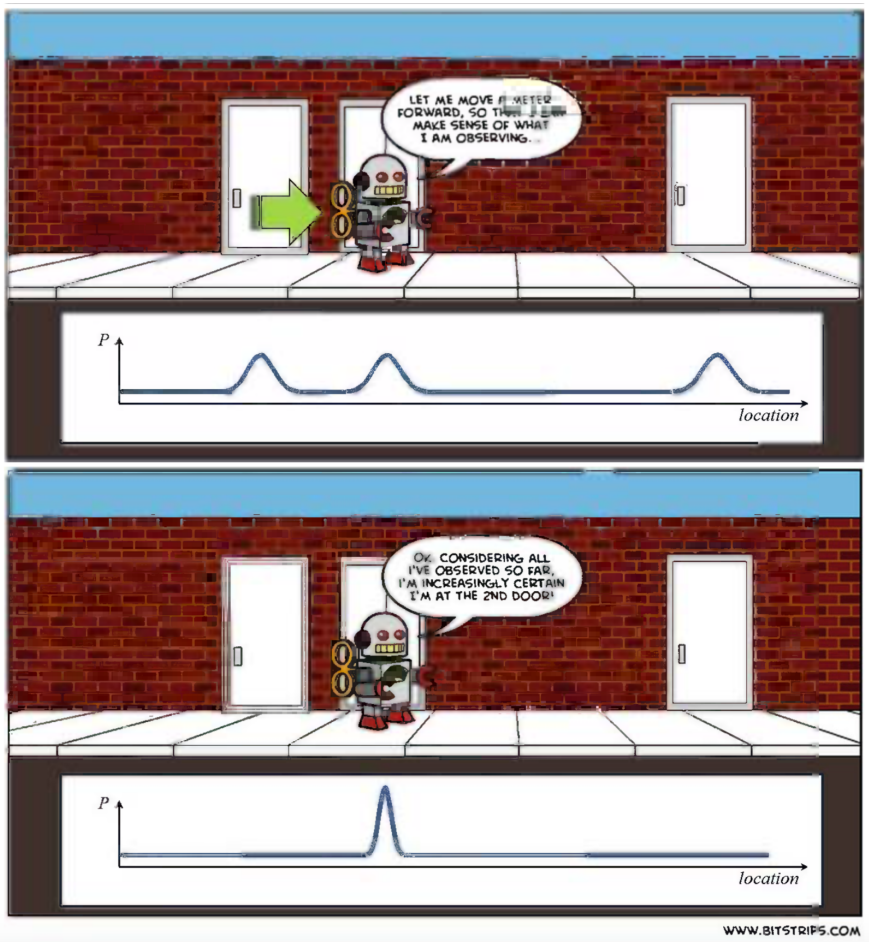


Bayes Filter

- Prediction



- Error Correction



Particles Filter

- Each particle has a state (x, y, θ) and a weight.
- The weight is an indicator of how close the particle' state is to the real state.

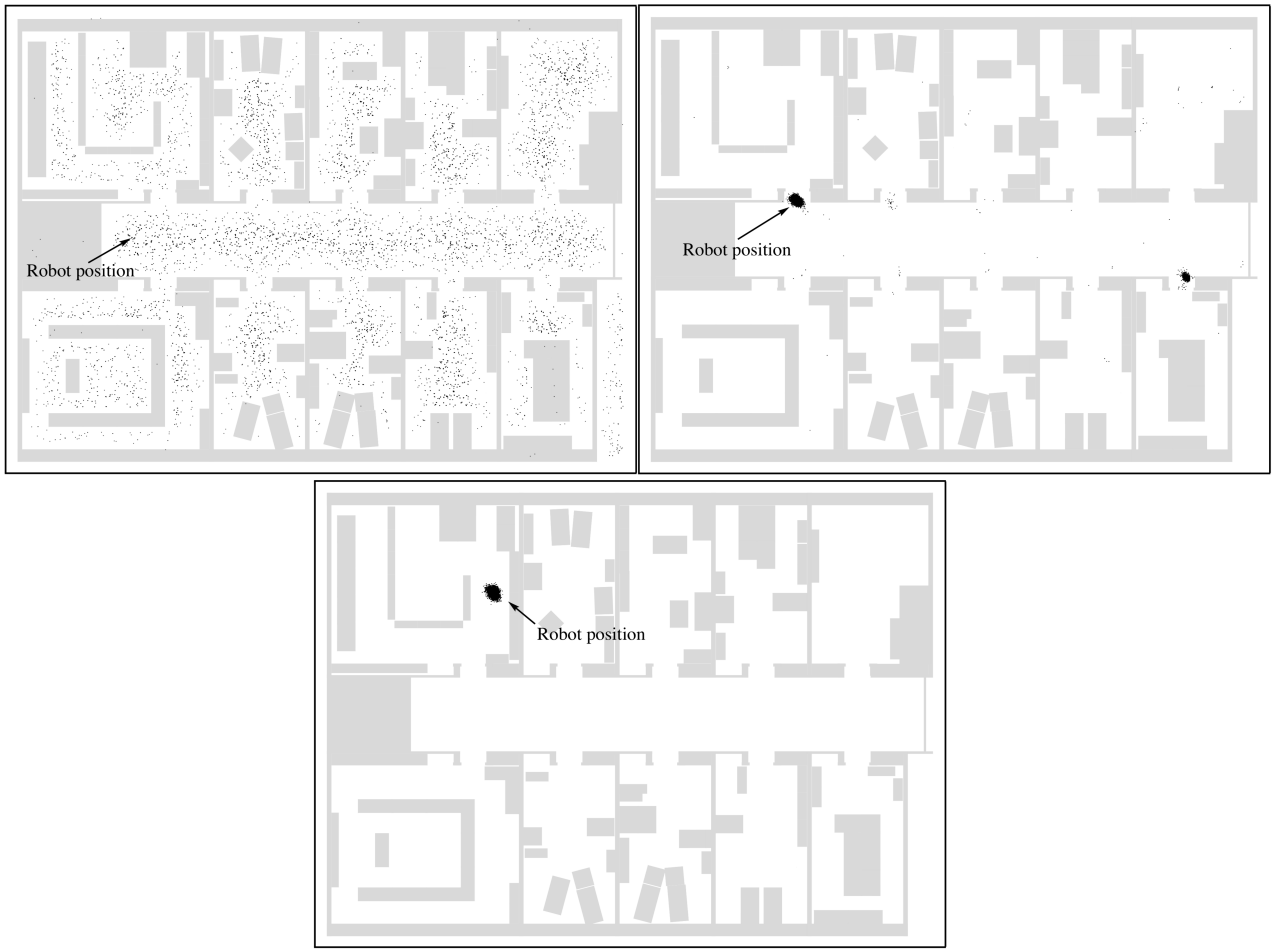
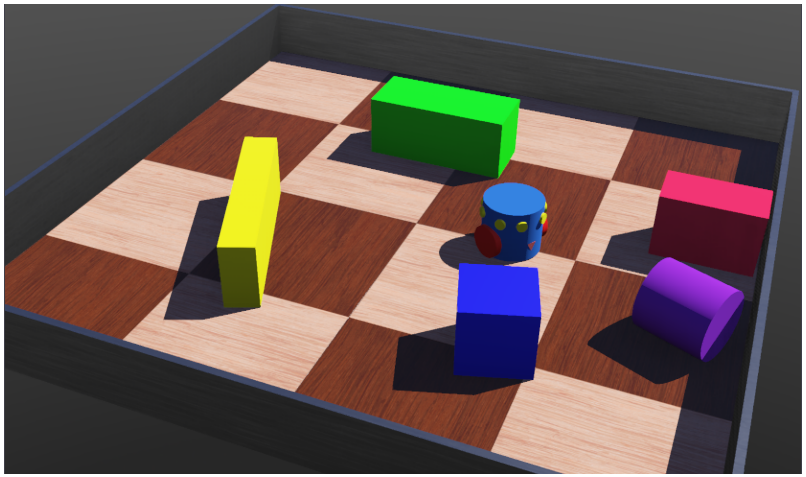


Table of Contents

What is Webots?

- Tool for simulating robots in virtual environments
- Open Source
- Python, C++, Java, etc.
- 44 robot models
- Robot model creation/customization
- Robust documentation



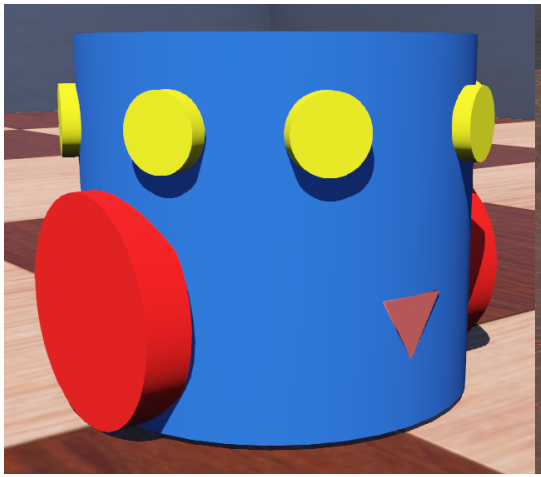


Table of Contents

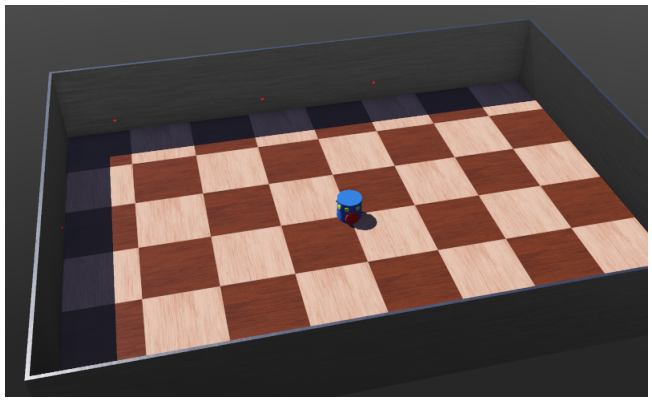
Objectives:

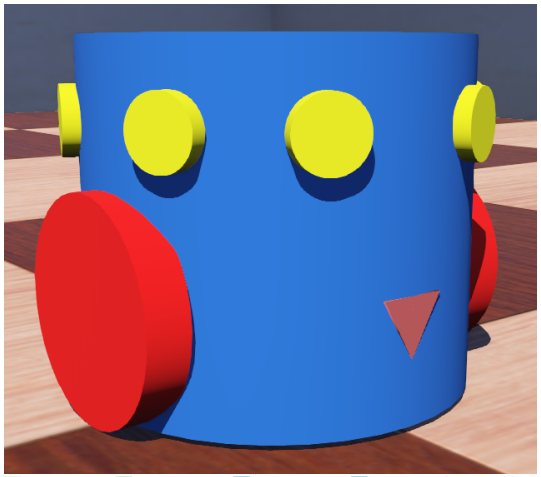
- The objective will be to exploit the simulation benefits of Webots to introduce Machine Learning techniques together with non-parametric filters (such as Particles Filter) for robot positioning estimation for in-door environments.
- Find a solution in which the learned knowledge can be transferred from one environment to another without the need to go through the training process using transfer learning techniques.

Table of Contents

Problem Formulation

- There is a robot with 8 laser sensors around it.
- The sensors measure the distance between the robot and the closest object.
- There is an arena of 2x1.5m with four walls.
- The robot has odometry data available.
- The robot is positioned in the middle of the arena and knows its initial position.
- The robot needs to track its position.





Solution Approach

- Capture data with the simulator.
  - True robot position (x, y, θ).
  - Sensor measurements $(s_1, s_2, ..., s_8)$.
- Train ML Models
  - Train 8 neural networks that receives a robot state as input and returns the estimated sensor measurements.
- Put the particles filter together with the ML Models to get the robot position estimation.
- Correct the position of the robot using the robot position estimation.

Capture data with the simulator

- Robot turns with a probability of 0.2; otherwise, it goes straight.
- Robot turns randomly to left or right maximum 30 degrees.
- Robot avoids colliding with the walls.
- Data is captured each 20 robot steps.
- The Data is captured for 10 minutes in fast simulation mode.

Train ML Models

- The data is normalized and shuffled.
- Many NN architectures were tested. The one that gave better results was:
  - Input Layer: 3 neurons
  - Intermediate Layer: Fully connected with 10 neurons
  - Intermediate Layer: Fully connected with 6 neurons
  - Intermediate Layer: Fully connected with 3 neurons
  - Output Layer: 1 neuron

Particles Filter + ML Model

input : deltaMove, sensorsData
output: particles

```
particles.state = particles.state + deltaMove
addRandomMove(particles.state)
foreach particle ∈ particles do
    particle.weight ← predictError(particle.state, sensorsData)
end
normalizeWeights(particles)
particles = resamplingBasedOnWeights(prob=particles.weight,
  replace=True)
return particles
```

Correction Step

- Take the weighted average state of all the particles.
- This value becomes the estimated state of the robot.
- Apply the correction step at each robot step.

Experiments

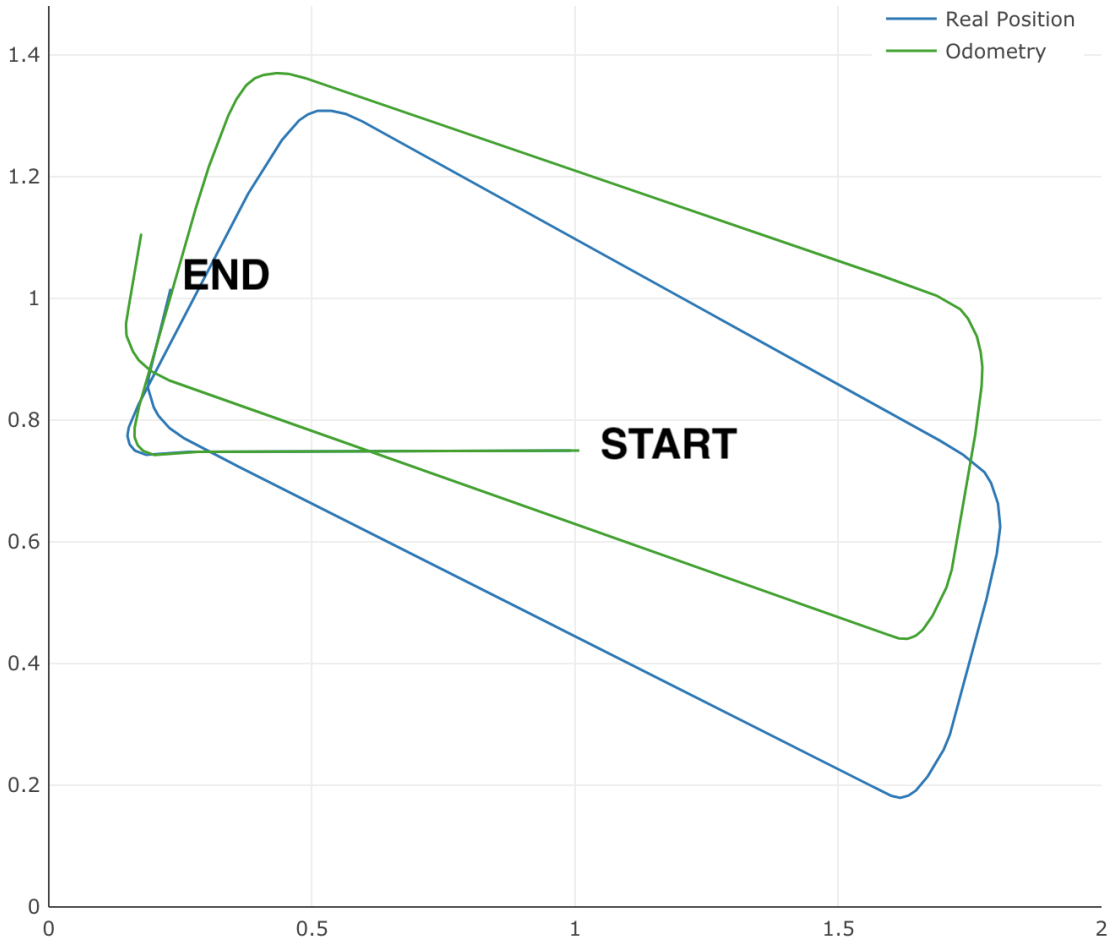
- Robot walks in the arena avoiding to hit the walls.
- The experiment was run during 2200 robot steps.
- The experiment was repeated each time with different number of particles: 30, 100, 500, 2000.
  - At each robot step two errors are calculated: one for the (x, y) coordinates and another for the angle θ.

$$Error_{xy} = \sqrt{(x - \hat{x})^2 + (y - \hat{y})^2} \quad (1)$$

$$Error_{\theta} = 180 - |180 - |\theta - \hat{\theta}|| \quad (2)$$

Table of Contents

Odometry vs Real position

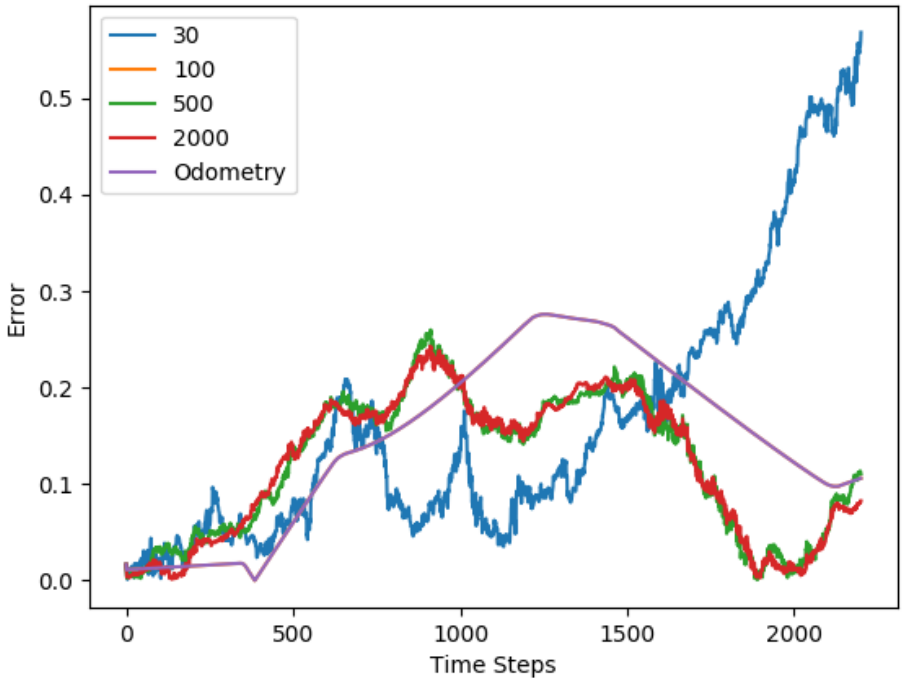


Particles vs Real position

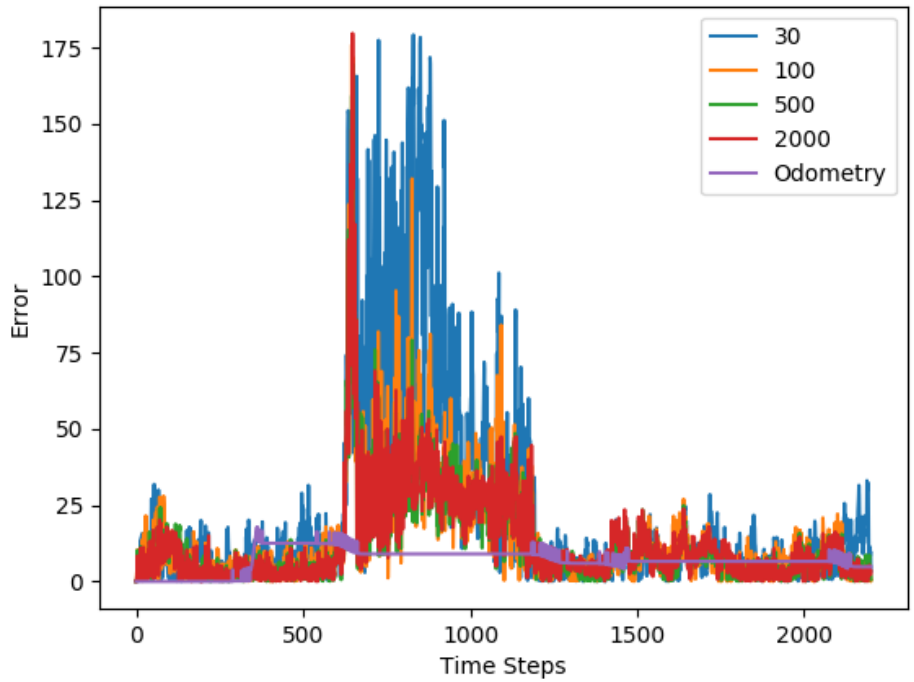
- 1000 particles used

Results

$Error_{xy}$



$Error_{\theta}$



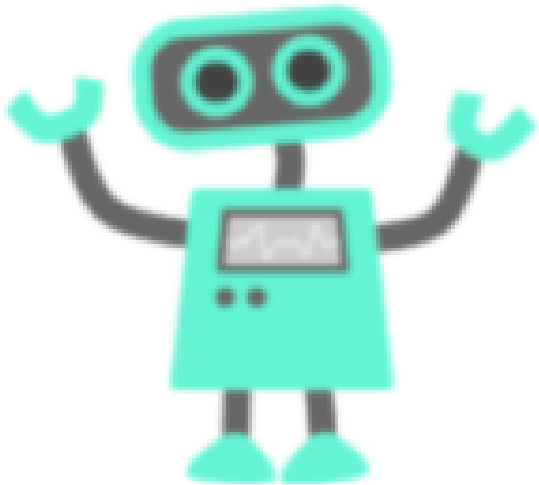
Drawbacks

- The problem requires to train 8 models which is time consuming.
- It does not generalize for unseen environments.
- The particle's filter algorithm works slow for a big amount of particles.
- Model highly dependent of odometry data.

Table of Contents

- Find an efficient model that receives as input the sensor measurements and predicts the robot position. Combine it with the previous model to improve the performance.
- Train a model that receives $(\delta x, \delta y, \delta\theta)$ as input and predicts $(\delta s_1, ..., \delta s_8)$ and equivalently that receives $(\delta s_1, ..., \delta s_8)$ as input and predicts $(\delta x, \delta y, \delta\theta)$.
- Produce results for different training sizes, different robots and room configurations.
- Implement online learning to improve gradually the models.

Any question?



(Suggestions and critics are also welcomed)