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

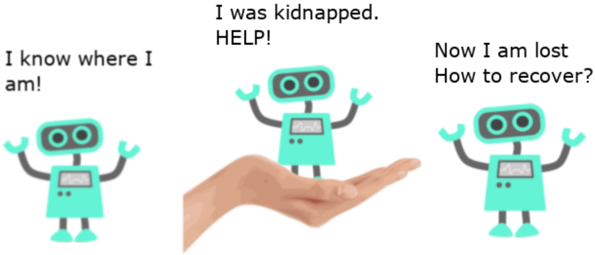
- A robot needs to deal with uncertainty most of the time.
- It needs to predict and preserve its current position and orientation within the environment.

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

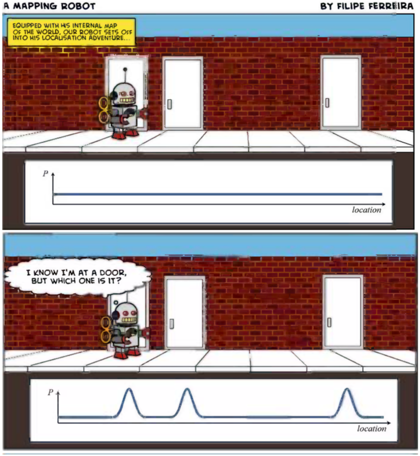
Robot Positioning Problem

- 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.

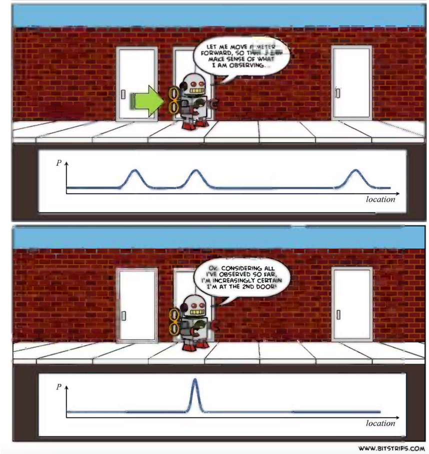


Bayes Filter

- Prediction

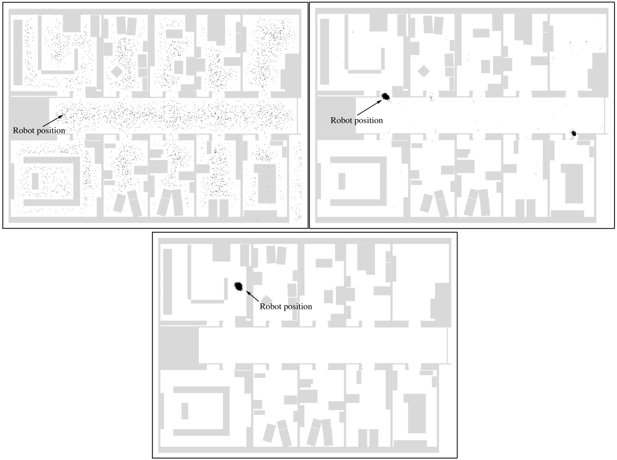


- Error Correction



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.



Particles Filter Algorithm

- Train/test a prediction model: robot state $\rightarrow$ distance sensor data
- Compare it with the true sensor data and determine a weight for each particle
- More close to the real sensor data more weight will have

---

input : particles, controlAction, sensorData

output: nextParticles

nextParticles $\leftarrow \emptyset$

foreach *particle* $\in$ *particles* **do**

  particle.state $\leftarrow$ predictState(particle.state, controlAction)

  particle.weight $\leftarrow$ calculateWeight(particle.state, sensorData)

end

for $m = 1$ *to* $m = |particices|$ **do**

  newParticle $\leftarrow$ draw i from *particles* with probability $\propto$ *particles*$[i]$.*weight*

  nextParticles.add(newParticle)

end

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

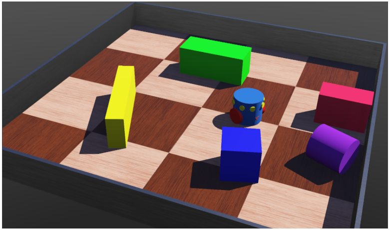
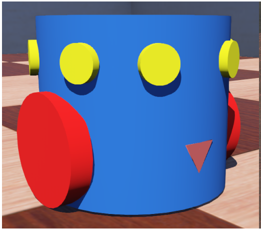




Table of Contents

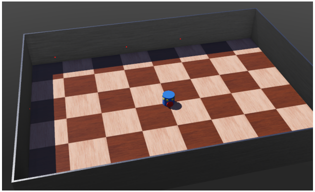
Main Objective

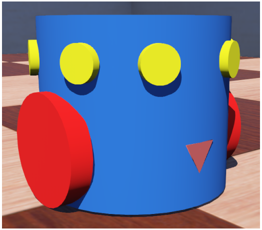
- The objective will be to exploit the simulation benefits of Webots to introduce Machine Learning techniques together with non-parametric filters (such as Particles Filter) for robot positioning estimation for in-door environments.

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 at the middle of the arena and it 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 the true robot position 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 capture each 20 robot steps.

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 : robotState (deltaMove), sensorsData
output: particles

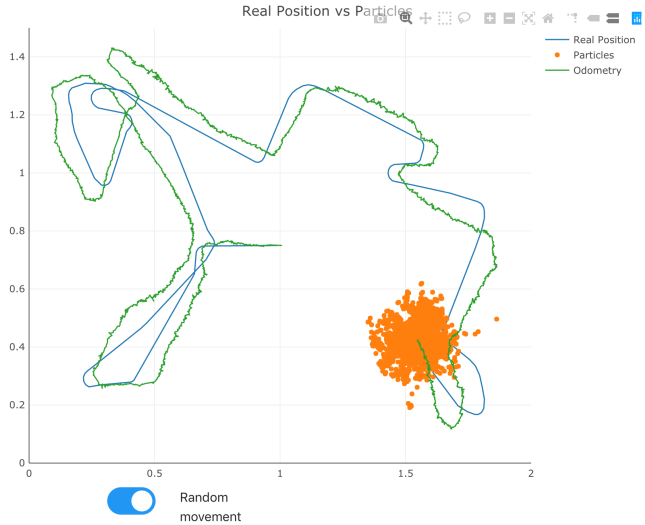
```
particles.state = particles.state + deltaMove
addRandomMove(particles.state)
foreach particle ∈ particles do
    particle.weight ← predictError(particle.state, sensorsData)
end
normalizeWeights(particles)
particles = resamplingBasedOnWeights(prob=particles.weight,
  replace=True)
return particles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

Correction Step

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

Table of Contents

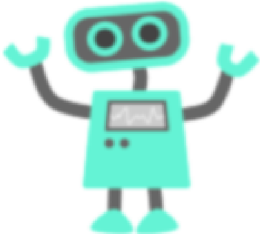
- 1000 particles used.



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.

Any question?



(Suggestions and critics are also welcomed)