UNIVERSITÉ LIBRE DE BRUXELLES
Faculté de Sciences
Département d'Informatique

Master Thesis

Machine Learning for Probabilistic Robotics with Webots

Joan Gerard

Promotor: Prof. Gianluca Bontempi

Academic year 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and objectives of the thesis

1.2 Notations

$t \in \mathbb{Q}$	Discrete or continuos time
$D \in \mathbb{N}$	Space dimension
$P \in \mathbb{N}$	Number of variables: univariate, multivariate
$S \in \mathbb{N}$	Number of sensors
$x_t \in \mathbb{R}^{D \times P}$	Matrix representing state x at time t
$z_t \in \mathbb{R}^S$	Vector of measurement z at time t
$u_t \in \mathbb{R}$	Action u at time t
$\hat{z} : \mathbb{R}^{D \times P} \to \mathbb{R}^S$	Function that predicts sensor measurements given a state
$\hat{x}_t \in \mathbb{R}^{D \times P}$	Matrix representing predicted state $\hat{x}$ at time t
$\hat{x}_t^* \in \mathbb{R}^{D \times P}$	Matrix representing best predicted state $\hat{x}$ at time t
$\theta_t \in \mathbb{R}$	Angle of orientation at time t
$bel(x_t)$	Belief of state x at time t
$\overline{bel}(x_t)$	Prediction belief of state x at time t
$p(x)$	Probability of continuous or discrete random variable x
$p(x\|y)$	Conditional probability of x given y
$x_{1:t}$	Sequence containing $\{x_1, x_2, ..., x_t\}$
M	Number of Particles used in the Particles Filter
$x_t^{[m]}$	State x at time t of particle m
$a \approx b$	a is approximately equal to b
$a \sim b$	a is similar to b
$a \propto b$	a is proportional to b

1.3 Abbreviations

EPFL	Swiss Federal Institute of Technology in Lausanne
ROS	Robot Operative System
3D	Three-Dimensional
DOF	Degrees Of Freedom
pdf	Probability Density Function
KF	Kalman Filter
PF	Particles Filter
EKF	Extended Kalman Filter
MCKF	Maximum Correntropy Kalman Filter
MMSE	Minimum Mean Square Error
MCC	Maximum Correntropy Criterion
IEKF	Invariant Extended Kalman Filter
PCC	Pearson Correlation Coefficient
PPF	Pearson Particles Filter
GPU	Grafical Processing Unit
GPGPU	General Purpose Graphical Processing Unit
MCL	Monte Carlo localization
SLAM	Simultaneous Localization And Mapping Problem
DP	Differentiable Programming
DPFs	Differentiable Particle Filters
PF-net	Particle Filter Network
HF	Histogram Filter
DMN	Differentiable Mapping Network
DPFRL	Discriminative Particle Filter Reinforcement Learning
DVRL	Deep Variational Reinforcement Learning
GRU	Gated Recurrent Unit
RNN	Recurrent Neural Networks
PF-RNN	Particles Filter Recurrent Neural Networks

Chapter 2

State of the art

2.1 Probabilistic Robotics

Robotics, according to Thrun et al.[1], is the science of perceiving and manipulating the physical world through computer-controlled mechanical devices and it is taking place in wide areas such as medicine[2], planet exploration[3], agriculture[4], construction[5], entertainment[6] among others.

The robotics that is known nowadays has being evolved across the years. From robotic systems designed to automatize highly repetitive and physically demanding human tasks in the early-to-mid-1940s[7], passing through researches making the strong assumption of having exact models of robots and environments in the 1970s, to probabilistic robotics in the mid-1990s[8]. Thus a robot needs to deal with uncertainty most of the time. As an example, a robot ordered to walk 1 meter forward from its current position will not end up exactly 1 meter far from its initial position due to some errors, if the robot speed was high and it slipped or the noise produced by the actuators is significant. That is why it needs to be capable to deal with uncertainty, predicting and preserving its current position and orientation within the environment[9].

2.1.1 State

State, according to Thrun et al.[1], refers to the set of all aspects of the robot and its environment that can influence to the future. Two types of state can be defined:

- **Dynamic state** changes its position over time. For instance, the people or other robots location.
- **Static state** does not change its position over time. For example, the walls or other moveless objects location.

Some instances of state are:

- The position of physical static entities in the environment as walls, boxes, doors, etc.
- The location and speed of mobile entities in the environment as people, other robots, etc.
- The robot pose is usually defined by its position and its orientation relative to a global reference frame. A robot moves on a *fixed frame* which is attached to the ground and does not move. This frame is called the *global reference frame* (GRF). Additionally, a robot is linked within a frame which moves along with the robot. This frame is referred as *local reference frame* (LRF). Communication between the coordinate frames is known as *transformation of frames* and it is a key concept when modeling and programming a robot[10]. Figure 2.1 illustrates the difference between GRF and LRF where the X_R axis points to the right side of the robot, the Y_R axis is aligned to its longitudinal axis and the Z_R axis points upward. Besides to these Cartesian coordinates, the robot's angle orientation is defined by the Euler angles: Yaw, Pitch and Roll (see [11]).

The notation used to represent a state at time t will be denoted as x_t.

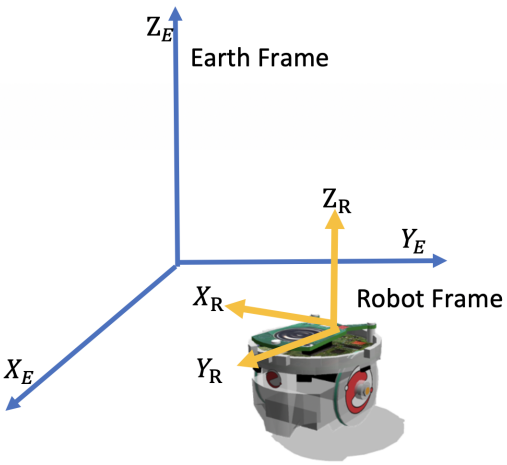


Figure 2.1: Local vs Global reference frame

2.1.2 Robot Versus The Environment

A robot can interact with the environment in two ways. First, it can modify the environment state using its actuators which are drivers acting as muscles that let the robot change its configuration[10]. For instance, a rotational motor is an actuator that can power a joint and thus changing the robot pose. As another example, a robotic arm can be used to move objects from one position to another. Second, a robot can sense the environment using its sensors which are elements, usually

attached to the robot, for obtaining information about internal and environmental states [10][1].

The control data is information about the change of state in the environment[1]. It can be, for instance, the velocity or the acceleration of the robot at a given time t and thus it will be represented as u_t. The sensor data provided at time t is denoted as z_t and therefore a sequence of sensor observations will be $z_{1:t} = z_1, z_2, ..., z_t$.

2.1.3 Belief Representation

A belief is a representation of the internal state of the robot about its position in the environment whereas the true state is where the robot truly is in the environment. In other words it is the probability that a robot at time t is at location x given previous sensor data $z_1, z_2, ..., z_t$ and previous control actions $u_1, u_2, ..., u_t$. A belief can be expressed as the conditional probability function of state x_t given sensor data $z_{1:t}$ and control actions $u_{1:t}$. Thus it assigns a probability to each location in the environment, dealing with a single-hypothesis belief (robot tracks a single possible solution) or multiple-hypothesis beliefs (robot tracks an infinite set of positions)[12]. Following Thrun et al. notation, a belief over a state variable x_t will be denoted as $bel(x_t)$ which is the posterior density conditioned on all past measurements and control actions.

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.1}$$

Equation 2.2 shows the posterior without including the last measurement data. It is usually called *prediction* in the probabilistic filtering area.

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.2}$$

The probability $bel(x_t)$ can be calculated using the previous posterior belief before measurement z_t and after taking action u_t, that is $\overline{bel}(x_t)$. This is known as correction[8].

The relation between equation 2.1 and 2.2 is shown in appendix A.1.

2.1.4 The Markov Property

The *MarkovProperty* states that a future state depends on the present state only and thus it resumes all the past states information.

Formally, let x_t be a stochastic discrete process that can take on a discrete set of values, where $t \in \mathbb{N}$. A value in process x_t has the *Markov Property* if for all t such

that $t \geq 1$, the probability distribution of x_{t+1} is determined by the state x_t of the process at time t, and does not rely on past values x_t for $t = 0, 1, ..., t-1$[13]. That is:

$$p(x_{t+1}|x_t, x_{t-1}, ..., x_0) = p(x_{t+1}|x_t) \tag{2.3}$$

The Markov Property is an approximation, commonly used in mobile robotics since it simplifies tracking, reasoning and planning and hence it has been found to be very robust for such applications[12].

2.1.5 Localization Methods

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. According to Siegwart et al. what is important for a robot to succeed in navigation is to succeed likewise in these four blocks of navigation:

- *Sense:* the robot perceives the environment though its sensors and extract meaningful data in order to process and obtain information.
- *Localization:* the robot knows where it is in the environment.
- *Cognition:* the robot creates an execution plan in how to act to reach its goals.
- *Control Action:* the robot executes the execution plan through modulating its output motors.

Sensors and actuators take part in determining the robot's localization but both are subjected to noise and thus the problem of localization becomes difficult. Another problem with sensors is that they usually does not provide enough information content to determine the position of the robot and therefore make it easy for the robot to be in an ambiguous location. This problem is known as sensor aliasing and along with sensors and actuators noise turns the localization problem into a difficult task[12].

There are two types of localization methods categorized by the increased degree of difficulty as stated in Ferreira et al.[7]:

- *Local techniques* (also called position tracking[1]), when the robot knows its initial position and it is not able to recover it if the robot loses track of it.

- *Global techniques* (see also [14]), when the robot does not have any information about its initial position and it is able to estimate it even in a global uncertainty. This technique includes the *kidnapped robot problem* in which a robot that knows its position is kidnapped and take it into another unknown location and its task is to recover its position. According to Thrun et al.[1] the later should be categorized as a third localization technique given its difficulty. This technique is more powerful than the former because it deals with global uncertainty and consequently a robot is able to recover its position even if its error is significant.

Local techniques approximate the pose using unimodal techniques contrary to global techniques where the robot's internal knowledge about the state of the environment is represented by a probability density function (pdf) over the space of all locations[8]. An example of a global technique is shown in section 2.1.7 where a robot handles multiple ambiguous positions. Then after moving, it can sense other factors in the environment that can lead to disambiguate its pose and put most of the probability in a single location and therefore having a better idea of where truly is as image 2.3 illustrates. A more extensive and explicative example can be found in [8] and [15] where the global localization problem is illustrated in a one-dimension space with a robot that has an hypothetical sensor that senses the presence of doors, when the robot finds the first door, the probability is distributed to all the places where there are doors (see figure 3.9). Thus the probability to be near a door is higher when the sensor detects a door but the robot does not know at this point which door is facing. Moving the robot forward makes the sensor to sense another door and thus increasing the probability of being at the second door. This example illustrates some important properties of the Bayes Filter.

2.1.6 Bayes Filter

As claimed by Borriello et al.[15] Bayes Filter is a statistically estimator of dynamic system's states based on noisy observations. In other words it is an algorithm that calculates beliefs distribution based on measurements and control data[1]. This is generally done in two steps as it was mentioned in section 2.1.3: the prediction and the correction step and thus each time a robot receives the sensors measurement data the robot controller software needs to compute the posterior densities shown in equation 2.2 but notice that such task is computationally heavy and consequently its time complexity grows exponentially because the amount of sensors measurements increasing over time. A solution to this problem is to assume that such dynamic system has the Markovian Property. That is the future state x_{t+1} depends on the present state x_t because the assumption that x_t carries all the needed information is made.

Bayes filter algorithm is recursive. It needs the previous posterior distribution to calculate the current posterior distribution. The algorithm is described below and its correctness is shown in appendix A.1.

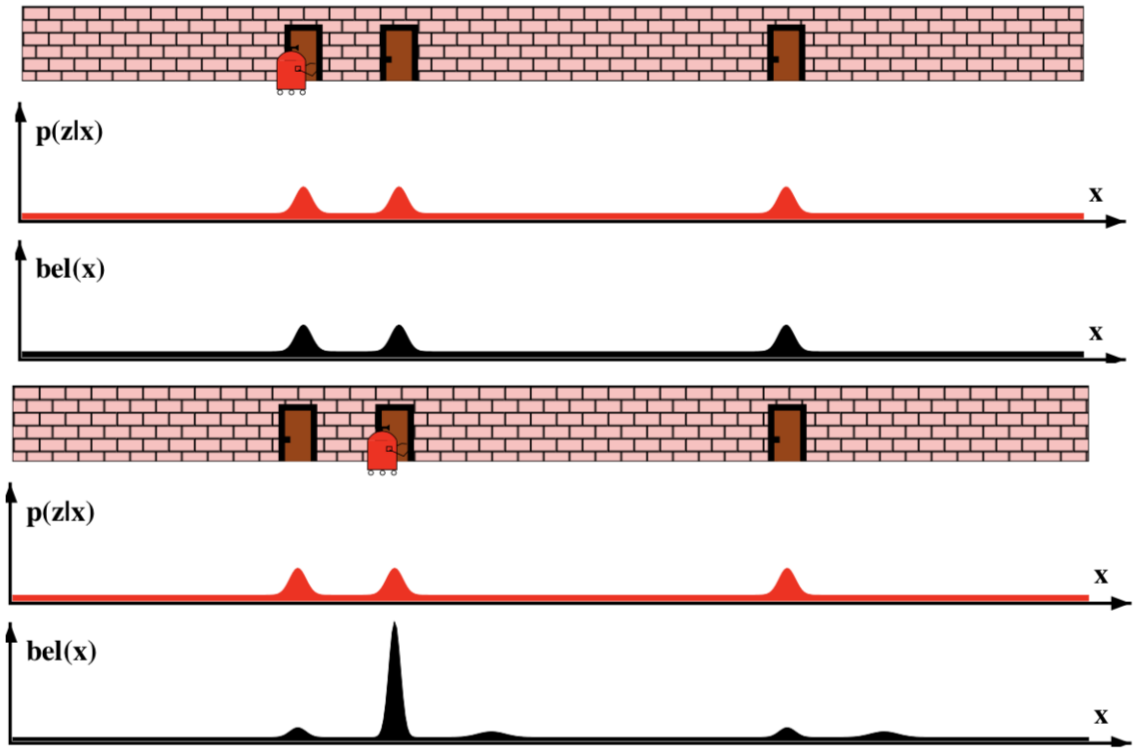


Figure 2.2: Probability density when robot senses.

Source: Extracted from Thrun et al. Probabilistic Robotics[8]

There are many algorithms that apply Bayes Filtering. Some examples are stated below.

- The *Kalman Filter* (KF) is a Gaussian technique invented in 1950 by Rudolph Emil Kalman[16] and was first implemented by NASA in the Apollo program to estimate the trajectory of the space capsule in real time[17]. It takes noisy data and put the noise apart in order to get information with less uncertainty[18]. It works with continuous states and it represents the beliefs by the first and second moment[15] from multivariate normal distributions. Chen et al. have proposed another version of KF called Maximum Correntropy Kalman Filter (MCKF) which is an effective and robust algorithm for non-Gaussian signals

Algorithm 1: Bayes Filtering Algorithm[1]

input : $bel(x_{t-1})$, u_t, z_t
output: $bel(x_t)$

foreach x_t **do**
  $\overline{bel}(x_t) \leftarrow \int p(x_t|u_t, x_{t-1})\, bel(x_{t-1}) dx_{t-1}$
  $bel(x_t) \leftarrow \eta\, p(z_t|x_t)\, \overline{bel}(x_t)$
end
return $bel(x_t)$

and heavy-tailed impulsive noise. Instead of using the Minimum Mean Square Error (MMSE) as KF, its optimality criterion is the Maximum Correntropy Criterion (MCC)[19].

- The *Extended Kalman Filter* (EKF) does not assume any linearity as KF does and thus the next state and the measurement probabilities are nonlinear functions[20][1]. Some variants of this method include Unscented Kalman Filter, where the state distribution is approximated by a Gaussian random variable as in EKF, but now a minimal set of sample points are cautiously chosen for representing the state distribution[21][22]. Another improvement of EKF is the Invariant Extended Kalman Filter (IEKF) used for continuous-time systems with discrete observations[23].

- The *Grid-based Filters* split the environment into small grids, each containing a belief of the true position state[15]. For instance, Burgard et al. proposed a bayesian approach based on certainty grids where each cell has associated a probability of the robot to be on that cell and thus they successfully predicted the robot absolute position and orientation in a global localization problem using standard sensors in complex environments[24].

- *Topological* approaches represent the environment using graphs where each node is a representation of a location and the edges are the connectivity between two locations[15].

- The *Particles Filter* (PF) according to Rekleitis, uses many copies of the variable of interest, each of them is associated with a weight representing the quality of that specific particle. Each particle executes the same control action plus some noise, they are reevaluated, its weight is recalculated and therefore the particles with low weight are eliminated in a process called resampling[25].

2.1.7 Particles Filter

Particles Filter is a nonparametric implementation of the Bayes Filter algorithm where the posterior is approximated by a set of M samples called *particles* where M is usually a large number(e.g. 2000). Under the context of localization, it is also known as Monte Carlo localization (MCL)[26]. Each particle has associated a weight that represents the contribution to the overall estimate of the posterior[25]. Thus equation 2.4 shows the believe at time t approximated by a set of particles and weights of M particles.

$$bel(x_t) \approx S_t = \{\langle x_t^{[m]}, w_t^{[m]} \rangle | 1 \leq m \leq M\} \tag{2.4}$$

Where x denotes the state of particle m, and w represents the weight of such particle.

According to Thrun et al. the probability of including a particle $x_t^{[m]}$ will be proportional to its posterior belief (see equation 2.5) and thus more particles in a region

of the environment means that the true state is more likely to be in that region.

$$x_t^{[m]} \sim p(x_t|z_{1:t}, u_{1:t}) \tag{2.5}$$

Table 2 illustrates the Particles Filter algorithm. The algorithm has two loops, representing the prediction and resampling steps respectively. The prediction step obtains M hypothesis about the true state using sampling techniques. All of these hypothesis together correspond to $\overline{bel}(x_t)$ in the Bayes algorithm. Furthermore the weight or importance factor for each particle is calculated as a conditional probability of sensing z_t given the state of that specific particle representing the posterior $bel(x_t)$ in the Bayes algorithm. The resampling step consists in selecting with replacement M particles from the temporal data set $\bar{S}_t$ with probability proportional to each particles' weight and thus the particles with lower weights have less probability to be selected[1]. Figure 2.3 illustrates the use of MCL for robot localization.

Algorithm 2: Particles Filtering Algorithm. Adapted from [1].

input : S_{t-1}, u_t, z_t
output: S_t

$\bar{S}_t \leftarrow \emptyset$
$S_t \leftarrow \emptyset$
for $m = 1$ *to* M **do**
  sample $x_t^{[m]} \backsim p(x_t|u_t, x_{t-1}^{[m]})$
  $w_t^{[m]} \leftarrow p(z_t|x_t^{[m]})$
  $\bar{S}_t \leftarrow \bar{S}_t \cup \{\langle x_t^{[m]}, w_t^{[m]} \rangle\}$
end
for $m = 1$ *to* M **do**
  $\langle x_t, w_t \rangle \leftarrow$ draw i from $\bar{S}_t$ with probability $\propto w_t^{[i]}$
  $S_t \leftarrow S_t \cup \{\langle x_t, w_t \rangle\}$
end
return S_t

Rekleitis et al. mention other resampling techniques: Linear Time Resampling and Resampling by Liu et al. The former uses the method of simulating order statistics[27], that is, obtaining $M+1$ random numbers, applying the negative log number and then calculating their cumulative sum, then compare them to the cumulative sum over the weights in order to generate a list of indexes corresponding to the particles with higher weights[28]. Resampling by Liu et al. proposed the use of a function of the weights ($a_j = f(w_j)$), for instance the square root $f(w_j) = \sqrt{w}_j$ and then normalize this quantity regarding the number of particles M. Then if a_j is greater or equal to one, the jth particle is duplicated k times, where $k = \lfloor a_j \rfloor$; otherwise the particle is selected with a probability equal to a_j[29][25].

Zhou et al. proposed an improved PF algorithm called the Pearson Particles Filter (PPF) because it is based on the Pearson Correlation Coefficient (PCC) which is a statistical technique to determine the linear dependence between two random

variables and thus it is used to decide how close the hypothetical particle state is to the true state value. Even though this technique solves the degeneracy and sample impoverishment problem present in PF, it is computationally complex[30].

Murray et al. present an alternative to accelerate the resampling technique eliminating the cumulative sum over weights using two schemes based on Metropolis[31] and rejection samplers making the problem of resampling parallelizable using graphical processing units (GPU)[32]. Nicely et al. go even further, introducing a general purpose graphical processing units (GPGPUs) implementation which takes advantage of the properties of the cumulative sum and the evolutionary nature of the particle weighting process to parallelize the re-index section of the algorithm[33].

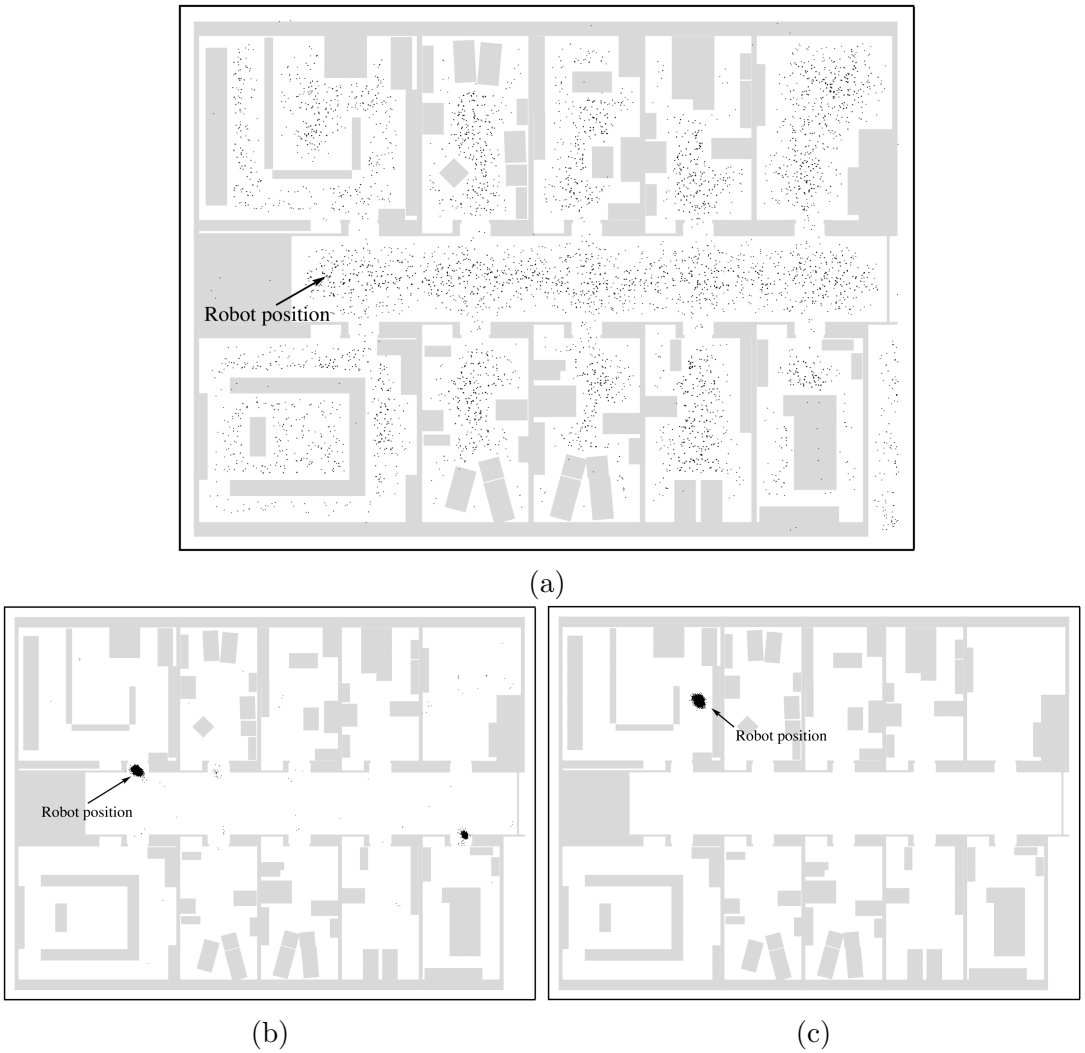


Figure 2.3: A robot localization example using PF. (a) The robot has a global uncertainty about its position in the map, (b) it handles some hypothesis about where it is after executing certain control actions, (c) it is quite sure where it is in the environment.

Source: Particles Filters in Robotics[34].

As claimed by Thrun, one drawback of PF is that the performance of the algorithm highly depends on the environment dimension, that is, higher the dimension,

worst its performance is. For instance, simultaneous localization and mapping problem (SLAM) is a clear example in which the robot does not have a previous built map, deals with a high-dimensional environment and does not have any information about its initial position. Consequently, SLAM cannot be resolved using conventional MCL algorithms due to the absence of an initial map. Doucet et al. find a solution to this problem with a variation of PF called Rao-Blackwellised Particle Filter resolving the problem efficiently up until 100000 dimensions[35]. Montemerlo et al. proposed FastSLAM, an algorithm that recursively estimates the full posterior distribution over robot pose and landmark locations[36] and it is one of the most robust algorithms for mapping in-door environments[34].

2.2 Related Work

Models nowadays claim for conditional branching, loops and recursion over trees[37]. That is why the idea of Differentiable Programming (DP) has emerged recently in the machine learning community, popularized by Yann LeCun[38]. DP refers to a programming model that is constructed using neural network blocks with data-dependent branches and recursion, being trainable using back propagation and gradient descent[39]. For instance, the differentiable version of branching can be implemented using the Sigmoidal function, the loops can be represented through Recursive Neural Networks, arithmetic operations using the Neural Arithmetic Logic Unit, etc.[40].

Jonschkowski et al.[41] presented the Differentiable Particle Filters (DPFs), a differentiable version of the classic particle filter algorithm with end-to-end learnable models. The recursive state prediction and correction steps are encoded in the DPFs structure which is made of a recurrent network representing the filtering loop. Even though their experiments reduces the error rates by 80% compared to algorithmic priors, it presents the limitation of resampling as a non-differentiable operation which stops the gradient computation after a single loop iteration limiting the scope of the implementation to supervised learning.

The popularity of DP has been extended through the field of Probabilistic Robotics. Karkus et al.[42] proposed the Particle Filter Network (PF-net), a fully differentiable system model that encodes the particle filter algorithm trained end-to-end from data. Unlike DPFs, PF-net solves the limitation of resampling as a non-differentiable operation presenting a differentiable approximation. PF-Net is used for Visual Localization with different kind of cameras: RGB, depth, RGB-D, simulated 2D Lidar, and a Lidar-W. Their observation model receives a spatial transformation of a part of the 2D floor map, and the output of the transition model and the images of the camera. It feeds a neural network architecture composed by convolutional and fully connected neural networks obtaining the particle likelihood. It also includes semantic information on the map as labels for doors and room types to improve the localization task. They conduct experiments at various levels of uncertainty: local, semi-grobal and grobal localization using the House3D data set[43] that contains a large collection of realistic buildings, obtaining better results compared to

alternative network architechtures such as Histogram Filter (HF) networks[44], Long Short-term Memory networks (LSTM)[45], PF, and odometry.

PF-net is used to develop further works on the domain of Probabilistic Robotics. For instance, Differentiable Mapping Networks (DMNs)[46] go one step further targeting the construction of a spatially structured view-embedding map which combined with PF-nets can be used for subsequent visual localization. DMN generates a structured latent map representation based on a set of image-pose pairs. The map representation is made up of pairs of viewpoint coordinates and learned image embeddings. This end-to-end differentiable model is evaluated in 3D simulated environments[47][48] and in more real-world environments using the Street View dataset[49], demonstrating strong performance in both.

Ma et al.[50] introduced a reinforcement learning framework for complex partial observations called Discriminative Particle Filter Reinforcement Learning (DPFRL). It uses a DPF in the neural network policy for reasoning with partial observations over time. It is composed of two main components: a discriminatively trained particle filter that tracks a latent belief, and an actor network that learns a policy given the belief. DPFRL are benchmarked using different problems in different domains. First, the Mountain Hike Task[51], consisting of an agent that goes from a start to a goal position in a map that contains an irregular terrain with partial visibility showing that DPFRL learns faster compared to other models such as Deep Variational Reinforcement Learning (DVRL)[51] and Gated Recurrent Unit (GRU)[52]. Second, Atari games using an introduced domain called Natural Flickering Atari Games that combine two challenges: the Flickering Atari Game, the partially observable version of the Atari games, where the observations are completely black frames half of the time and the Natural Atari Games where the black background is replaced by a sampled video stream. Results show that DPFRL outperformed in 7 out of 10 games. Finally, visual navigation using an RGB-D camera in the Habitat Environment[53] with the Gibson dataset[54] in which the robot navigates to a goal in previously unseen environments significantly outperforms both DVRL and GRU.

Ma et al.[55] extended Recurrent Neural Networks (RNN) to use particles filters. Differently to RNNs that approximates the belief as a long latent vector, updating it using a deterministic nonlinear function, Particles Filter Recurrent Neural Networks (PF-RNNs) approximates the belief using a set of weighted particles and the stochastic particle filter algorithm to update them. PF-RNNs are applied to LSTM and GRU RNNs architectures, called PF-LSTM and PF-GRU respectively. They are evaluated for the robot localization task in three custom 2D synthetic symmetric maps. The robot uses the distance to a set of landmarks to simulate sensor measurements. PF-LSTM is able to converge to the true pose even though it has previously converged towards a wrong distant pose which is unusual while using the particles filter algorithm by itself. The results show that PF-RNNs perform better than traditional RNNs architectures on sequence prediction tasks.

Chapter 3

Webots

3.1 Introduction

Some datasets and simulators have been popularized lately among researchers in the probabilistic robotics field to handle tasks such as visual robot localization or the embodied question and answering problem[56]. For instance, House3D[43] is a virtual 3D environment containing 45000 indoor 3D scenes with any kind of 3D labeled objects. As another example, AI Habitat[53] is a simulation platform for embodied AI agents training in a highly photorealistic and efficient 3D simulator. In this section Webots is presented as another simulator tool that can be used in the probabilistic robotics field. It offers the advantage to be highly customizable regarding to the robots and the world creation, it can be easily extended through custom robot windows plugins, and it is not oriented to visual robot localization only but any general robot localization task.

Webots was created by Cyberbotics Ltd. a spin-off company from the EPFL. It has been developing it since 1998. The company currently[1] employs 6 people in Lausanne, Switzerland to continuously develop Webots according to customers needs. Cyberbotics provides consulting on both industrial and academic research projects and delivers open-source software solutions to its customers. It also provides user support and training to the users of the Webots software. The source code and binary packages are available for free, their team provides support to the users through a Discord channel[2]. The program is available for MacOS, Ubuntu Linux, and Windows operative systems[57].

Webots website has a reference manual that describes nodes and API functions. It has a complete user guide as well endowed with examples of simulations that show the use of actuators, creation of different environments, geometries primitives, complex behaviors, functionalities and advanced 3D rendering capabilities. This section is oriented to describe the basics of Webots and is focused on what will be

[1] 2019

[2] `https://discordapp.com/invite/nTWbN9m`

useful to develop the project. For a detailed description please refer to the user guide[3] or the reference manual[4].

For this project Webots R2020a version is used.

3.2 Configuration of the Local Environment

3.2.1 Graphic Interface

Webots supports the following programming languages: C, C++, Python, Java, MATLAB and ROS. Additionally, it offers the possibility of creating a custom interface to third-party software such as Lisp™ or LabView™ using TCP/IP protocol.

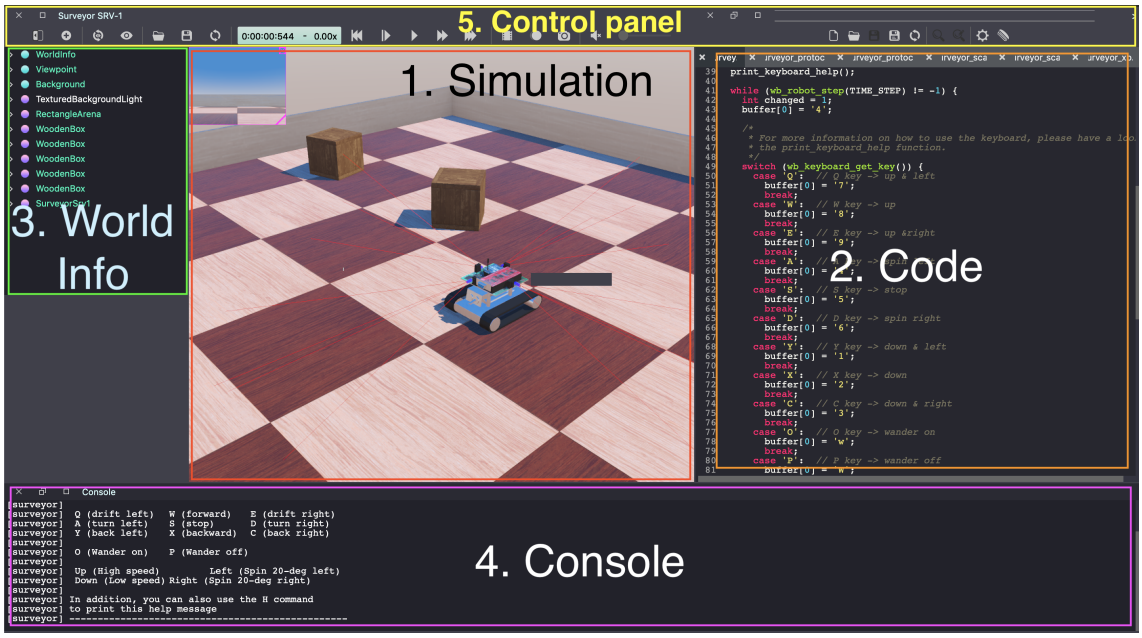


Figure 3.1: Webots graphic interface

Figure 3.1 shows the main graphic interface of Webots. It can be divided in 5 panels:

1. Simulation: graphic visualization of the simulated world objects.
2. Code visualization: robot controller code editor.
3. World Information: information represented in a tree structure about the simulated world including the robot components (sensors, actuators, physical characteristics), world objects, textures, etc.
4. Console: program execution output stream.

[3] https://cyberbotics.com/doc/guide/index
[4] https://cyberbotics.com/doc/reference/index

5. Control panel: set of buttons that controls the simulation execution.

The program allows users to create highly personalized simulated environments which are called worlds, from scratch using pre-built 3D object models such as robots, wood boxes, walls, arenas, etc. A robot needs to be associated with a controller program that contains the source code with the desired behavior. This controller can be easily edited in the code panel and once it is saved it is automatically reloaded into all the robots associated with it. For running the simulation, users can play, stop or reset it, among other options, using the control panel set of buttons. The output stream of the controller execution will be displayed in the console panel.

3.2.2 Webots with Python 3.7

Python was created in 1990 by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands as a successor of a language called ABC[58]. Nowadays it has became one of the most popular programming languages for data science[59].

The Python API of Webots was created from C++ API and it supports Python 3.7. It is possible to configure Webots to use Python 3.7 which should be previously installed from the Python website[5]; however, Webots does not work properly with Python versions installed from package managers as Brew. By default Webots is configured to use the default installed version of Python. In order to use another version, access to the menu options in Webots: `Webots/Preferences`; the *Python command* label should point out to the installation path of Python3.7 as it is shown in figure 3.2.

3.2.3 Debug Controller

When running a simulation is important to debug the code in order to catch some possible bugs on the program and to figure out why the program is not running as it should. Webots does not allow to debug code while running a simulation but it allows to run a robot controller from an external program such from an Integrated Development Environment (PyCharm, IntelliJ, Eclipse etc.), Python command lines, etc. This allows users to debug the controller.

IntelliJ will be used as an example on how to configure an external controller to run the simulation on Webots using Python 3.7 code.

The robot node has an option labeled `controller` that specifies what controller the robot should use. This option should be configured selecting the `<extern>` property as it is shown in figure 3.3. This property tells Webots that the robot controller will be initiated using another application.

[5]Download it from `https://www.python.org`

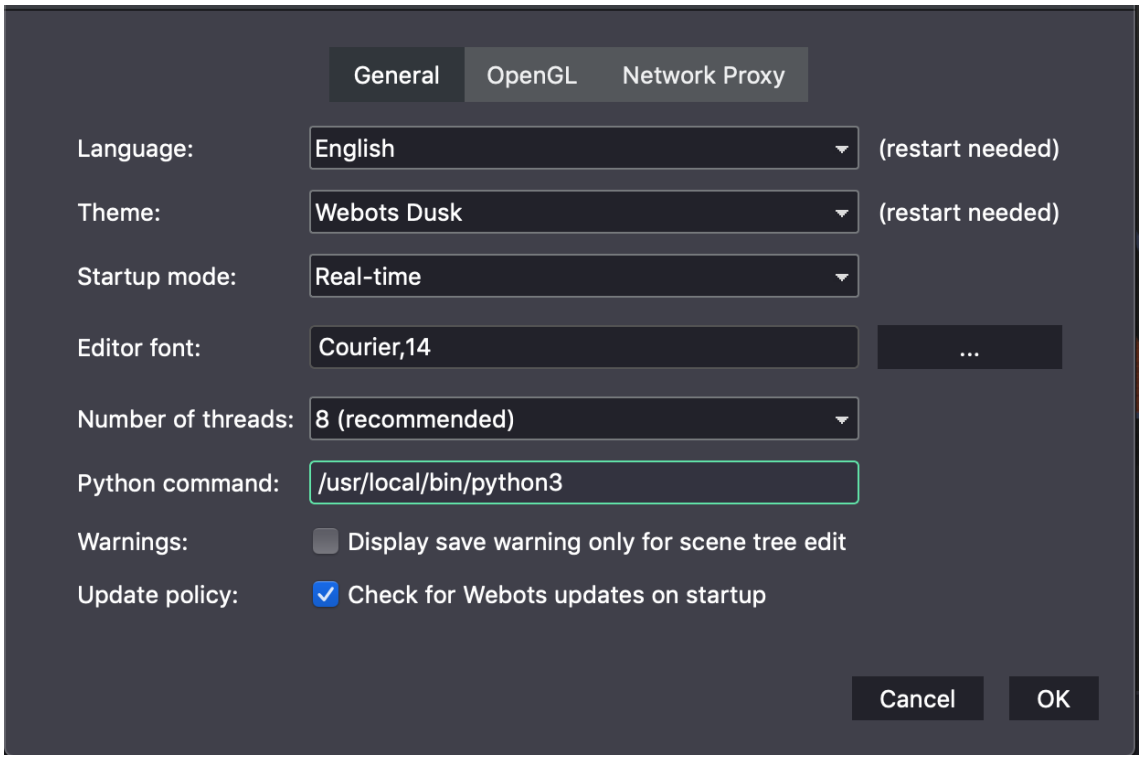


Figure 3.2: Configure Python3.7

Environment Variable	Value
WEBOTS_HOME	/Applications/Webots.app
DYLD_LIBRARY_PATH	add ${WEBOTS_HOME}/lib/controller
PYTHONPATH	add ${WEBOTS_HOME}/lib/controller/python37

Table 3.1: Environment Variables Configuration for MacOS

Table 3.1 show the environment variables needed to be configured for MacOS[6].

In order to configure IntelliJ to use the Webots Python API a new project will be created, accessing to `File/Project Structure.../Modules` and Add Content Root button can be used to add a new folder to the path, selecting the `WEBOTS_HOME/lib/controller/python37` folder as it is shown in figure 3.4.

A final configuration is needed accessing to the menu option `Run/Run...` then `/Edit Configurations...`, and adding the needed environment variables as it is shown in figure 3.5.

Once the configuration has been made IntelliJ is ready to run the controller. It can be made pressing on the `Run` button. Debugging is also possible with the `Debug` option.

[6]For a complete list refer:
`https://cyberbotics.com/doc/guide/running-extern-robot-controllers?tab-os=macos&tab-language=python`

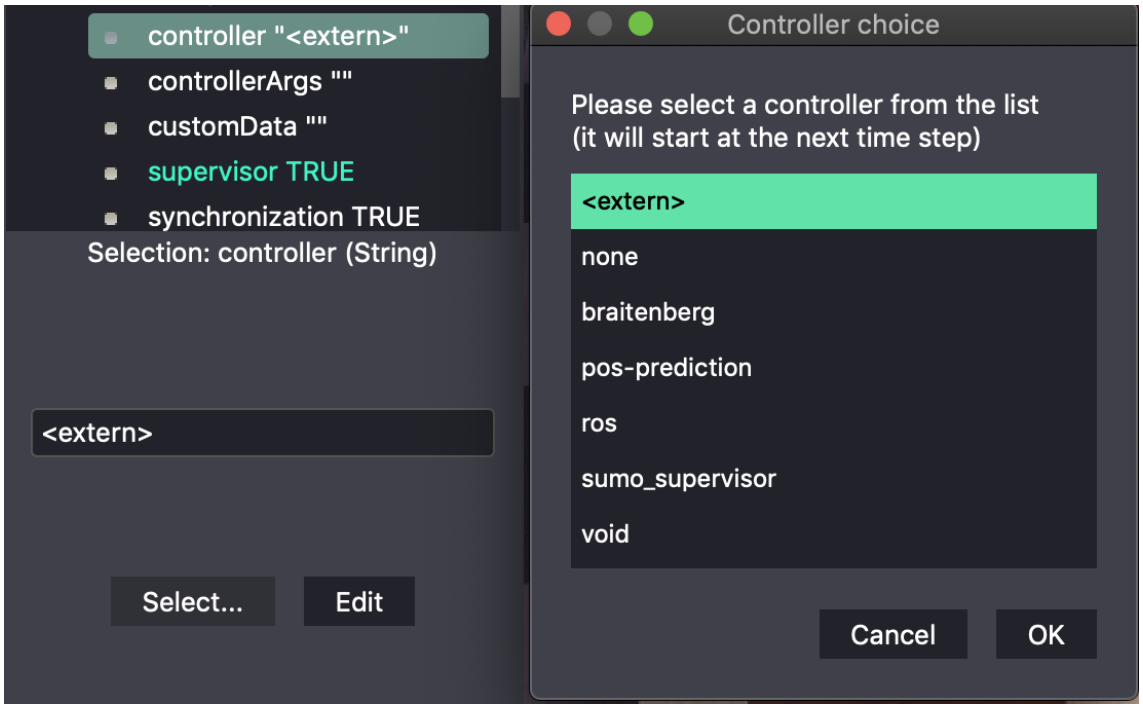


Figure 3.3: Extern Controller in Webots

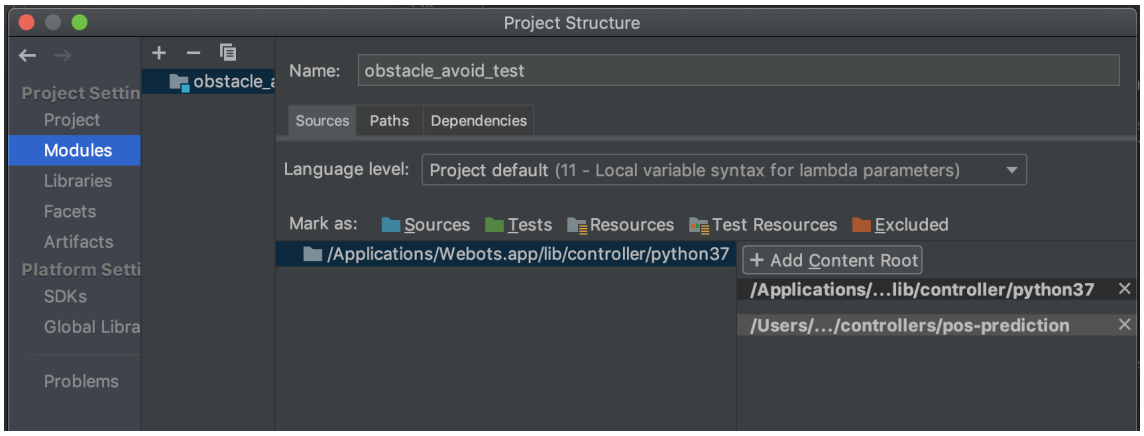


Figure 3.4: Extern Controller in Webots

3.2.4 Install Python Libraries

Installing the required python libraries for the project inside a python virtual environment is a good practice. To create a python virtual environment the command `pip3 install virtualenv` can be used.

The most-used libraries in this project are TensorFlow and Keras.

TensorFlow is a Python-friendly open-source library developed by the researchers and engineers of the Google Brain team for internal use only and then released in November 2015 under a permissive open source license. It implements machine learning algorithms and deep learning wrappers[59].

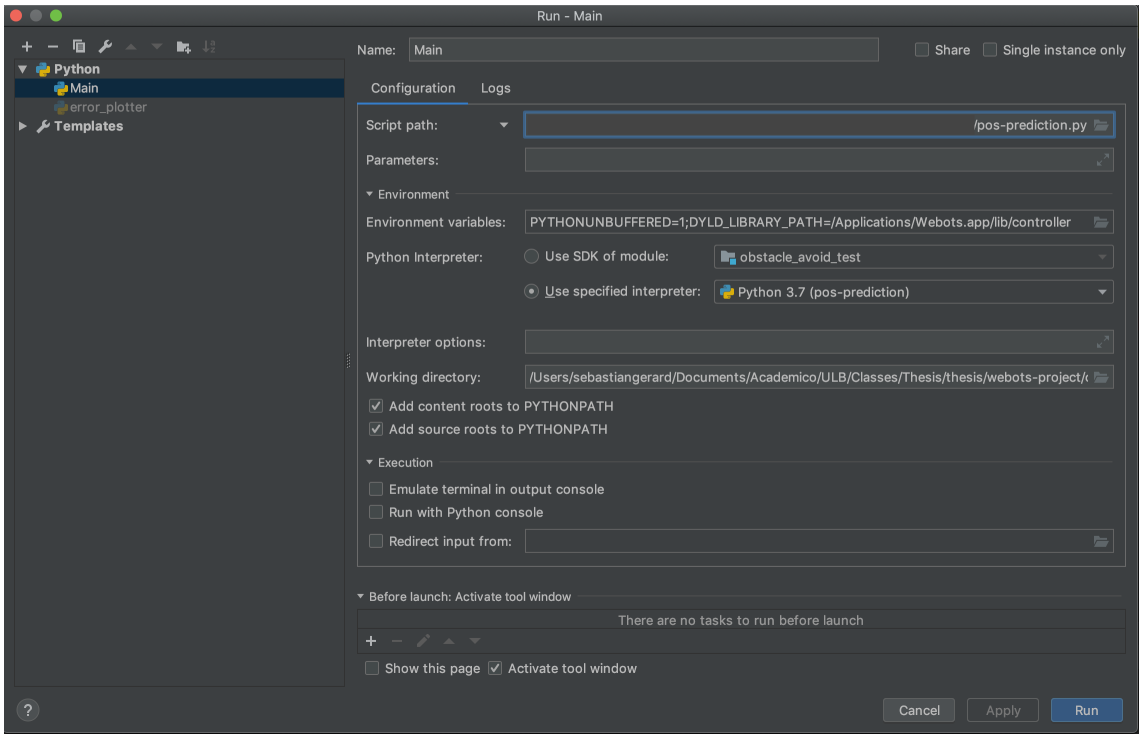


Figure 3.5: Environment Variables Configuration IntelliJ

The `pip3` command allows to install any library for Python 3.7. For installing TensorFlow type `pip3 install tensorflow` in the console terminal. For verifying the correct installation, the code displayed in listing 3.1 can be put inside a controller application in Webots.

```
1 import tensorflow as tf
2
3 verifier = tf.constant('TensorFlow was installed correctly.')
4 sess = tf.Session()
5 print(sess.run(verifier))
```

Listing 3.1: Verify correct installation of TensorFlow

If TensorFlow is correctly installed, after the execution of the simulation, a message in the console panel will be displayed informing about its correct installation.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It was developed with a focus on enabling fast experimentation[60]. Similarly to TensorFlow, it can be installed using the `pip3 install keras` command.

3.3 Webots and the Virtual World

Webots allows creating large simulated worlds. The world description and content is presented as a tree structure where each node represents an object in the world, those objects have themselves nodes and sub-nodes within a name and a value indicating different physical characteristics or components.

3.3.1 Robots and World Creation

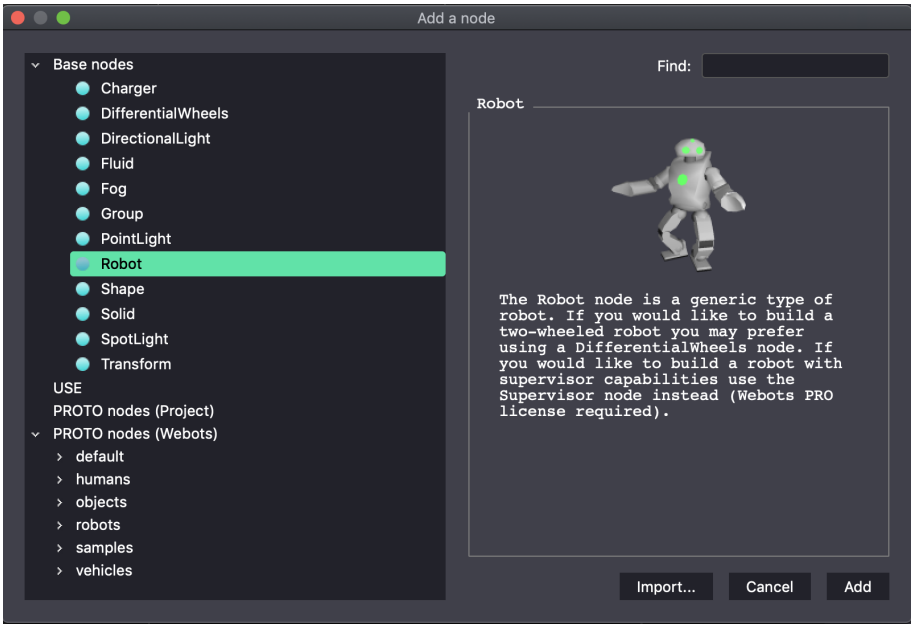
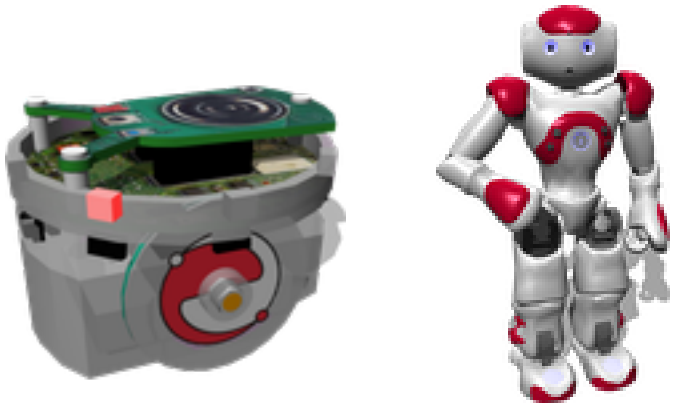


Figure 3.6: World structure

For adding a node into the world we can press the `Add object` button that will open the window shown in figure 3.6. The base nodes are the basic objects that can be part of the world. Moreover, they are simpler models than the others. The USE nodes are objects that were already created in the world and can be reused. For instance, if a wood box was added into the world with some specific physical characteristics, a name can be assigned to its USE attribute and then it can be reused instead of creating a new one with same characteristics. The PROTO nodes contains a set of 3D models as robots, objects, vehicles, etc. Fruits, toys, drinks, plants, chairs, stairs and buildings are only a small part of the big set of modeled objects. The robots node offers as well a big diversity of models. From simple robots as the e-puck (figure 3.7a) model which is used very often in research, to more complex robots as the very well known humanoid Nao (figure 3.7b) are some examples of a total of 44 3D modeled robots that are available to use within the simulator tool.

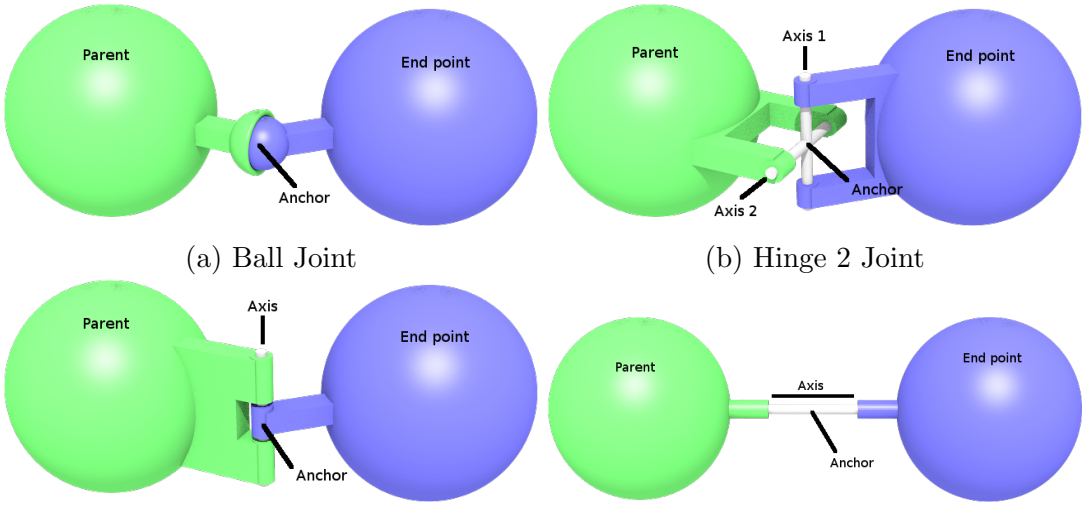
Another powerful feature of Webots is to create a custom robot model from scratch using a tree-based structure of solid nodes which are virtual objects with physical properties. Thus a robot is made of a set of solid nodes put them together using joint



(a) e-puck (b) Nao

Figure 3.7: Robots

nodes which are abstract nodes that models mechanical joints. Figure 3.8 shows the different types of joints that can be used.



(a) Ball Joint (b) Hinge 2 Joint

(c) Hinge Joint (d) Slider Joint

Figure 3.8: Different types of joint nodes

Source: Webots documentation

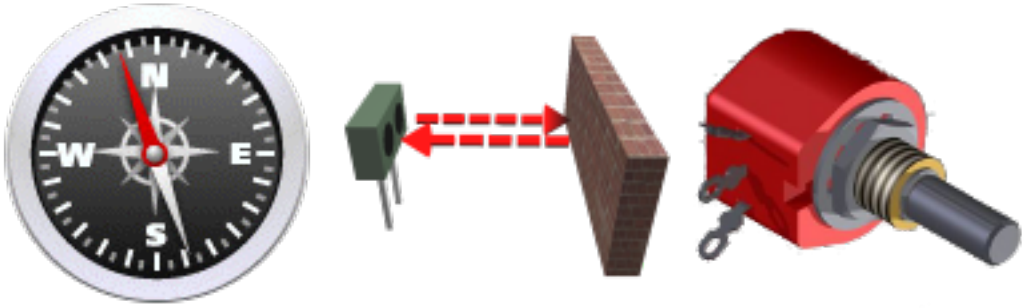
A position sensor can be added into the devices property of a joint node in order to monitor it. In like manner a rotational motor node can be added for actuating it. Thus, putting all together a simplistic version of a custom robot can be created based on the tree information and properties given to the simulator.

3.3.2 Sensors

Sensors allows the robot to sense the environment. Webots has a wide range of generic and commercially available sensors that can be plugged into a custom robot. On one side, the compass, distance sensor and position sensor are some few examples of generic sensors. On the other side, camera, lidar, radar and range finder sensors

built by different manufacturers as Hokuyo, Velodyne and Microsoft are offered as subcategories of commercially available sensors.

- The **compass sensor** can be used to express the angle orientation of the robot over the position of the virtual north as a 1, 2 or 3-axis vector (roll, pitch and yaw angles). The virtual north can be specified in the WorldInfo node by the `northDirection` field. The major fields that can be customized are: the xAxis, yAxis, zAxis, lookup table and resolution field.
- The **distance sensor** measures the distance between the sensor and an object throughout rays collision with objects. Webots models different types of distance sensors as: generic, infra-red, sonar and laser. All of these has a lookup table, number of rays cast by the sensor, aperture angle, gaussian width and resolution field that can be personalized according to the needs.
- A **position sensor** can be inserted into the device field of a Joint or a Track. It measures the mechanical joint position. Moreover, the noise and resolution of the sensor can be customized. This sensor is useful for estimating the position of a robot given the current position of a mechanical joint, this technique is known as Odometry.



(a) Compass sensor (b) Distance sensor (c) Position sensor

Figure 3.9: Sensors

Source: Webots documentation

Sensors have some fields in common. For instance, the resolution field that indicates the sensitivity of the measurement. When it is set to -1 means that it will measure the infinitesimal smallest perceived change. As another example, the lookup table is a field that maps a value from the sensor read data to another custom value. Additionally, the standard deviation noise introduced by the sensor can be specified for a given measure.

Table 3.2 shows the possible values that can be associated with the lookup table field. The first column represents the data measured by the sensor, the second column is the mapped value and the third column represents the noise as a gaussian random number whose range is calculated as a percentage of the response value[57]. For instance, for the row $[0.3, 50, 0.1]$ when the sensor senses an object to 0.3m of

Value	Mapped value	Noise
0	1000	0
0.1	1000	0.1
0.2	400	0.1
0.3	50	0.1
0.37	30	0

Table 3.2: Lookup table of distance sensor

Source: Webots documentation

distance from the robot, it will return a value of 50 ± 5, where 5 represents the standard deviation, that is the 10% of 50.

Figure 3.10 shows the relation between the current sensor values and the mapped values within the associated noise.

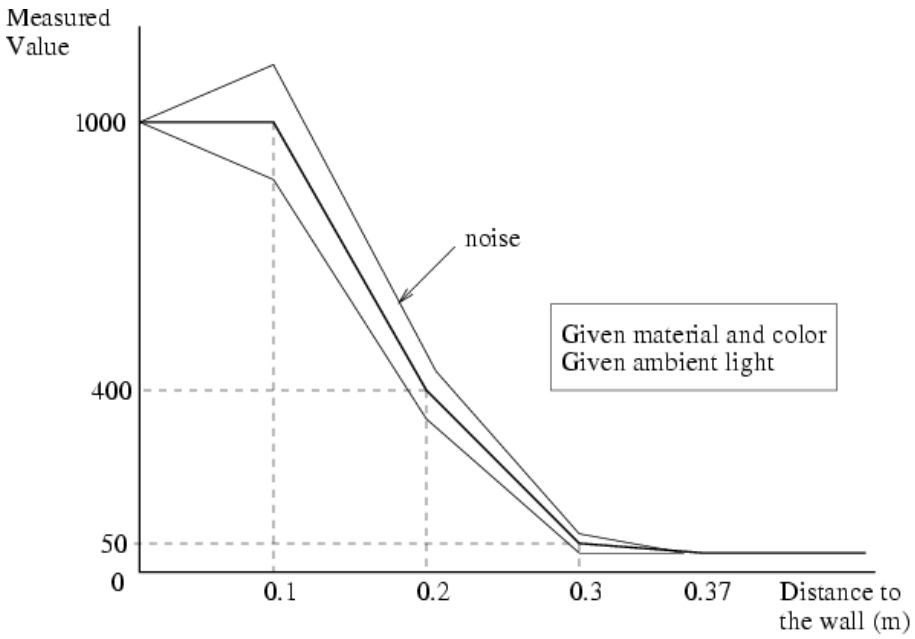


Figure 3.10: Sensor response versus obstacle distance

Source: Webots documentation

3.3.3 Actuators

Actuators allows the robot to modify the environment. Webots can simulate rotational motors, linear motors, speakers, LEDs, etc.

3.4 Webots Controller Plugins

Webots allows the creation of custom user-implemented windows interfaces for the robots which are integrated to Webots using a controller plugin.

3.4.1 Robot Window

A robot window is a custom window that can be made using HTML and Javascript and then associated to a specific robot using its `window` property. The robot windows need to be created under the `plugin/robot_windows/<window_name}>` directory. The window's name should be the same as the controller's name; otherwise, Webots does not recognize it and therefore it does not display it under the windows list.

The controller can send and receive messages to the window robot using the command `wwiSendText(text)` and `wwiReceiveText()` respectively. Similarly the window robot can communicate with the controller with the `webots.window(" <window name> ").receive` and `webots.window(" <window name> ").send` commands.

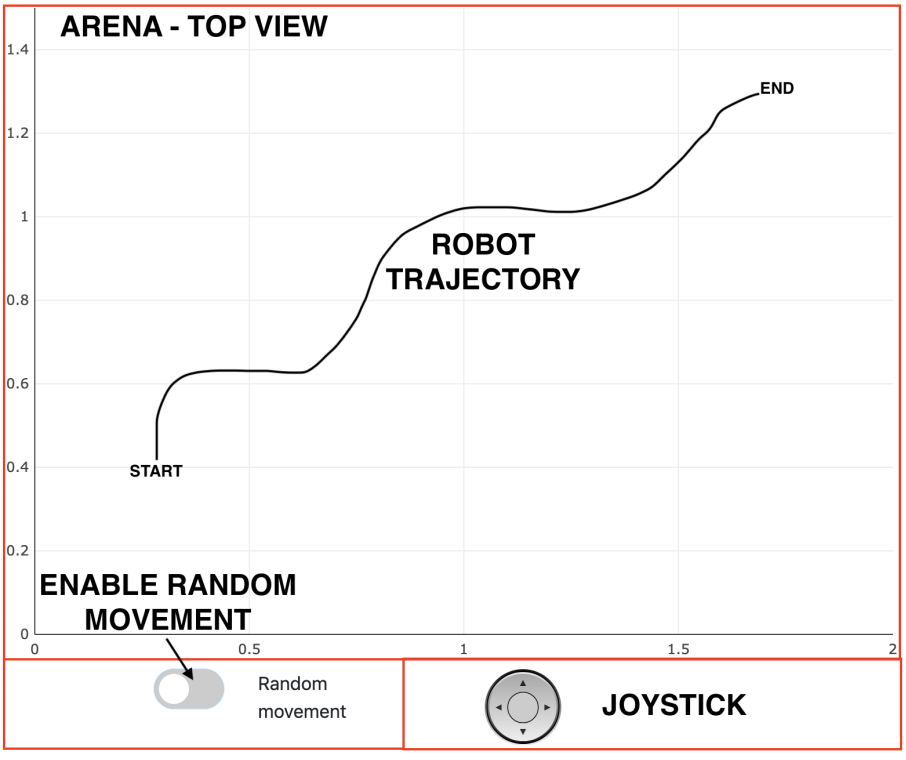


Figure 3.11: Custom robot window for visual robot localization

3.4.2 Robot Window to Visualize Robot Localization

A robot window can be created to visualize data associated with the robot localization task such as the real robot positioning compared to the odometry data.

Moreover, a joystick can be implemented to control the robot actuators.

Figure 3.11 shows three components of the robot window. First, the top-view arena visualization where the robot trajectory is traced while the simulation is running. This component was made using the Plotly open source JavaScript library[7] to draw charts. Second, the random movement switch that allows to disable/enable the robot random movement. If the switch is turned off, a joystick component will be displayed that will allow to control the robot movements.

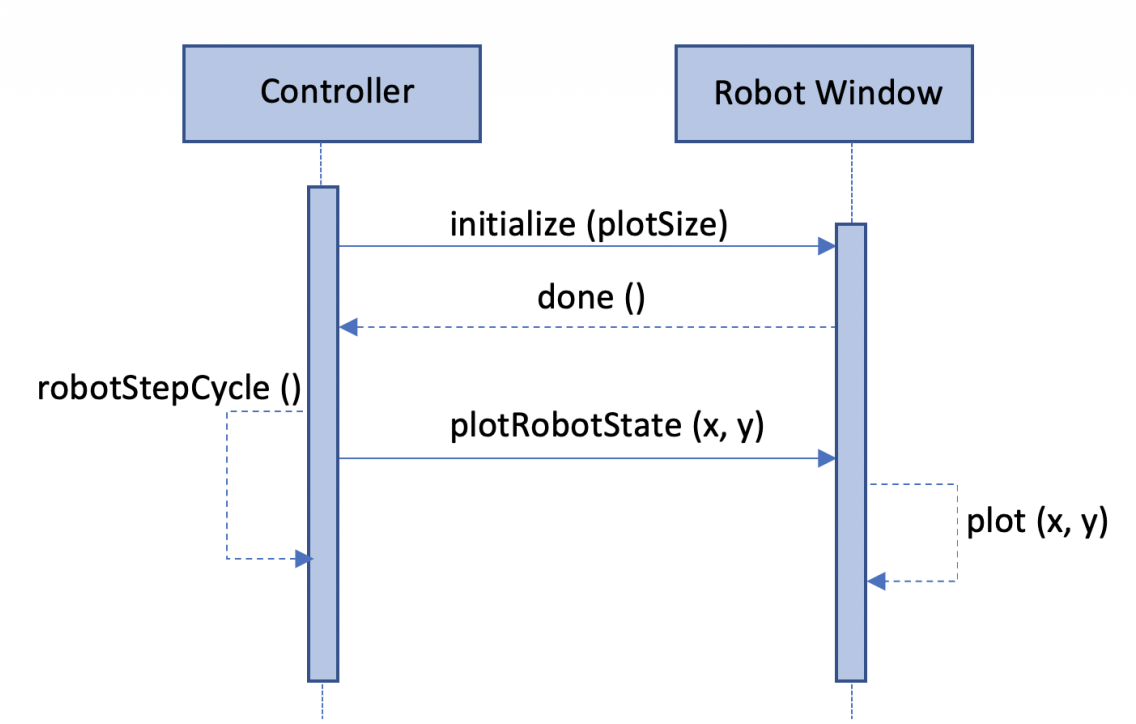


Figure 3.12: Sequence diagram for plotting robot positioning.

Figure 3.12 shows through a sequence diagram how the controller communicates with the robot window. First the controller sends an initialization command to the robot window together with some parameters such as the arena size. Once the simulation starts at each robot step the controller sends the robot positioning coordinates to the robot window which is in charge of plotting them.

The random robot movement is disabled when the user clicks on the switch component. This is made through the robot window which afterwards sends a message to the robot controller to stop moving the robot. Additionally, a joystick is displayed to the user who has four possibilities: move the robot up, down, left or right. Once one of these options is selected the robot window sends a message to the controller with the desired movement to do, then the latter translates this action to the robot wheels actuators. This process is presented in figure 3.13. When the random robot movement is enabled again the controller is communicated about this action and starts moving the robot randomly.

[7] `https://plotly.com/javascript/`

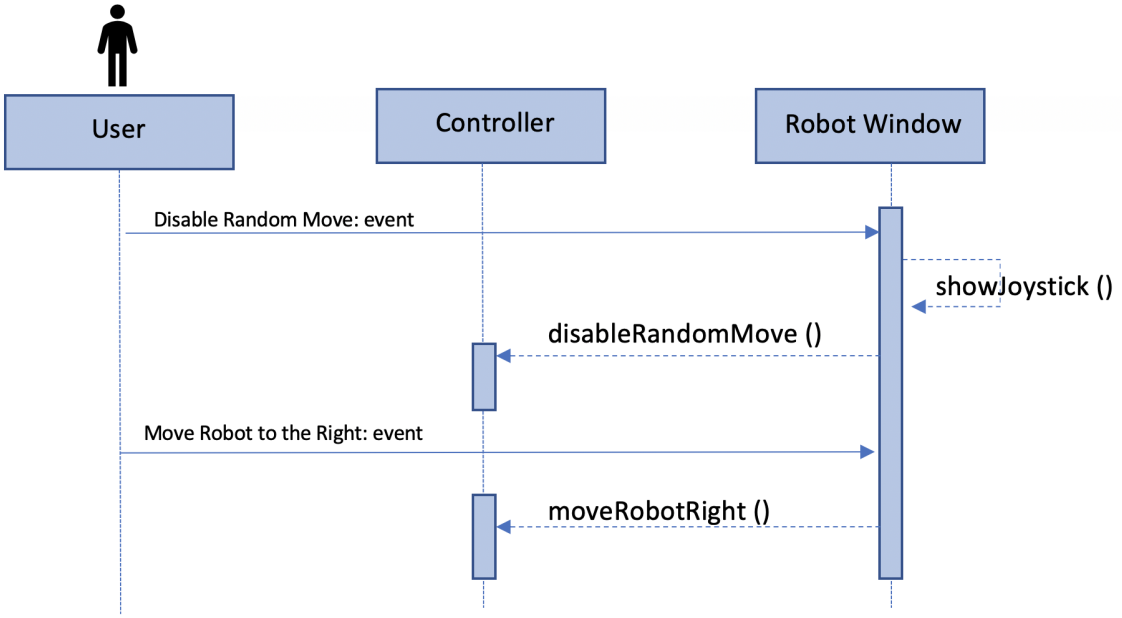


Figure 3.13: Sequence diagram for robot random movement deactivation.

This robot window offers some control over the robot in order to see the result of the experiments while running the simulation.

Chapter 4

Algorithms

Chapter 5

Experiments

Chapter 6

Conclusion

Appendix A

Appendix

A.1 Correctness of the Bayes Filter Algorithm

The current posterior distribution is calculated as it is shown in equation A.1.

$$p(x_t|z_{1:t}, u_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})\, p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \tag{A.1}$$

Where,

- $p(x_t|z_{1:t-1}, u_{1:t})$ is the prior distribution. That is the information of state x_t before seen the observation at time t.
- $p(z_t|x_t, z_{1:t-1}, u_{1:t})$ is the likelihood model for the measurements. A causal, but noisy relationship[61].
- $p(z_t|z_{1:t-1}, u_{1:t})$ is the normalization constant defined as η

Thus equation A.1 can be summarized as follows:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta\, p(z_t|x_t, z_{1:t-1}, u_{1:t})\, p(x_t|z_{1:t-1}, u_{1:t}) \tag{A.2}$$

The prediction of observation z_t based on the state x_t, the previous observation $z_{1:t-1}$ and the control action $u_{1:t}$ has a conditional independence regarding the previous observation and the control action because they do not aport any information while predicting z_t. Thus the likelihood model for the measurements can be simplified as follows:

$$p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{A.3}$$

Therefore equation A.2 reduces to:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta\, p(z_t|x_t)\, p(x_t|z_{1:t-1}, u_{1:t}) \tag{A.4}$$

In equation A.5 the prior distribution is expanded.

$$p(x_t|z_{1:t-1}, u_{1:t}) = \int p(x_t|x_{t-1}, u_t)\, p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) dx_{t-1} = \overline{bel}(x_t) \tag{A.5}$$

Replacing equation A.5 in A.4, equation A.6 is obtained which is calculated by the algorithm in list 1, third line.

$$p(x_t|z_{1:t}, u_{1:t}) = \eta\, p(z_t|x_t)\, \overline{bel}(x_t) \tag{A.6}$$

Bibliography

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[2] Azad Shademan et al. "Supervised autonomous robotic soft tissue surgery". In: *Science Translational Medicine* 8 (May 2016), 337ra64–337ra64.

[3] Geoffrey Landis. "Robotic exploration of the surface and atmosphere of Venus". In: *Acta Astronautica* 59 (Oct. 2006), pp. 570–579.

[4] Redmond Shamshiri et al. "Research and development in agricultural robotics: A perspective of digital farming". In: *International Journal of Agricultural and Biological Engineering* 11 (July 2018), pp. 1–14.

[5] Pileun Kim, Jingdao Chen, and Yong Cho. "SLAM-driven robotic mapping and registration of 3D point clouds". In: *Automation in Construction* 89 (May 2018), pp. 38–48.

[6] Morris K.J. et al. "Robot Magic: A Robust Interactive Humanoid Entertainment Robot." In: *Recent Trends and Future Technology in Applied Intelligence. IEA/AIE 2018. Lecture Notes in Computer Science.* 10868 (2018), pp. 38–48.

[7] João Filipe Ferreira and Jorge Dias. *Probabilistic Approaches to Robotic Perception*. Jan. 2014, pp. 3–36.

[8] Sebastian Thrun. "Is Robotics Going Statistics? The Field of Probabilistic Robotics". In: *Communications of The ACM - CACM* (Jan. 2001).

[9] Nikos Vlassis, B Terwijn, and B Krose. "Auxiliary Particle Filter Robot Localization from High-Dimensional Sensor Observations." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. Feb. 2002, 7–12 vol.1. ISBN: 0-7803-7272-7.

[10] Reza Jazar. *Theory of applied robotics: Kinematics, dynamics, and control (2nd Edition)*. Jan. 2010, pp. 7, 17–20.

[11] G Cook. *Mobile robots: Navigation, control and remote sensing*. 2011, pp. 79–92.

[12] R. Siegwart et al. *Introduction to autonomous mobile robots.* 2011, pp. 265–366.

[13] N. Privault. *Understanding Markov chains: Examples and applications.* 2018, pp. 89–108.

[14] L (Liqiang) Feng, Bart Everett, and J (Johann) Borenstein. *Where am I? : sensors and methods for autonomous mobile robot positioning.* Apr. 1996.

[15] G. Borriello et al. "Bayesian Filtering for Location Estimation". In: *IEEE Pervasive Computing* 2.03 (July 2003), pp. 24–33. ISSN: 1536-1268.

[16] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering, 82 (Series D)* (Mar. 1960), pp. 35–45.

[17] Axel Barrau and Silvère Bonnabel. "Invariant Kalman Filtering". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (May 2018).

[18] Matthew Rhudy, Roger A Salguero, and Keaton Holappa. "A Kalman Filtering Tutorial for Undergraduate Students". In: *International Journal of Computer Science & Engineering Survey* 08 (Feb. 2017), pp. 01–18.

[19] BD Chen et al. "Maximum Correntropy Kalman Filter". In: *Automatica* 76 (Sept. 2015).

[20] Paul Zarchan and Howard Musoff. *Fundamentals of Kalman Filtering: A Practical Approach (Third Edition).* Progress in Astronautics and Aeronautics. American Institute of Aeronautics and Astronautics, Inc., 2009.

[21] Simon J. Julier and Jeffrey K. Uhlmann. "A New Extension of the Kalman Filter to Nonlinear Systems". In: *Proc. SPIE* 3068 (Feb. 1999).

[22] E. A. Wan and R. Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373).* Oct. 2000, pp. 153–158.

[23] Axel Barrau and Silvère Bonnabel. "The Invariant Extended Kalman Filter as a Stable Observer". In: *IEEE Transactions on Automatic Control* 62 (Oct. 2014).

[24] Wolfram Burgard et al. "Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids". In: *Proceedings of the National Conference on Artificial Intelligence* 2 (Mar. 2000).

[25] Ioannis Rekleitis. "A particle filter tutorial for mobile robot localization". In: *Statistics for Engineering and Information Science* (Jan. 2004).

[26] S. Thrun et al. "Robust Monte Carlo Localization for Mobile Robots". In: *Artificial Intelligence* 128.1-2 (2000), pp. 99–141.

[27] Ioannis Gerontidis and R.L. Smith. "Monte Carlo Generation of Order Statistics from General Distributions". In: *Journal of the Royal Statistical Society. Series C. Applied Statistics* 31 (Jan. 1982).

[28] James Carpenter, Peter Cliffordy, and Paul Fearnhead. "An Improved Particle Filter for Non-linear Problems". In: *IEE Proc. Radar, Sonar Navig.* 146 (Sept. 2000).

[29] Jun Liu, Rong Chen, and T Logvinenko. "A theoretical framework for sequential importance sampling and resampling". In: *Sequential Monte Carlo Methods in Practice* (Jan. 2001), pp. 1–24.

[30] Haomiao Zhou et al. "A new sampling method in particle filter based on Pearson correlation coefficient". In: *Neurocomputing* 216 (July 2016).

[31] Nicholas Metropolis et al. "Equation of State by Fast Computing Machines". In: *jcph* 21 (June 1953), pp. 1087–.

[32] Lawrence Murray, Anthony Lee, and Pierre Jacob. "Parallel Resampling in the Particle Filter". In: *Journal of Computational and Graphical Statistics* 25 (July 2015).

[33] M. A. Nicely and B. E. Wells. "Improved Parallel Resampling Methods for Particle Filtering". In: *IEEE Access* 7 (2019), pp. 47593–47604.

[34] Sebastian Thrun. "Particle Filters in Robotics". In: *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*. UAI'02. Alberta, Canada: Morgan Kaufmann Publishers Inc., 2002, pp. 511–518. ISBN: 1-55860-897-4.

[35] Arnaud Doucet et al. "Rao-blackwellised Particle Filtering for Dynamic Bayesian Networks". In: *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*. UAI'00. Stanford, California: Morgan Kaufmann Publishers Inc., 2000, pp. 176–183. ISBN: 1-55860-709-9.

[36] M. Montemerlo et al. "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem". In: *Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI, 2002.

[37] Michael Innes et al. "On Machine Learning and Programming Languages". In: Feb. 2018.

[38] Yann LeCunn. *Yann LeCunn about Differentiable Programing*. `https://www.facebook.com/yann.lecun/posts/10155003011462143`. 2018.

[39] Fei Wang et al. "Backpropagation with Callbacks: Foundations for Efficient and Expressive Differentiable Programming". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 10180–10191. URL: `http://papers.nips.cc/paper/8221-backpropagation-with-callbacks-foundations-for-efficient-and-expressive-differentiable-programming.pdf`.

[40] Simone Scardapane. *Deep learning from a programmer's perspective (aka Differentiable Programming)*. `https://towardsdatascience.com/deep-learning-from-a-programmers-perspective-aka-differentiable-programming-ec6e8d1b7c60`. 2019.

[41] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. "Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors". In: *CoRR* abs/1805.11122 (2018). arXiv: `1805.11122`. URL: `http://arxiv.org/abs/1805.11122`.

[42] Peter Karkus, David Hsu, and Wee Sun Lee. *Particle Filter Networks with Application to Visual Localization*. 2018. arXiv: `1805.08975 [cs.RO]`.

[43] Yi Wu et al. "Building Generalizable Agents with a Realistic and Rich 3D Environment". In: (Jan. 2018).

[44] Rico Jonschkowski and Oliver Brock. "End-to-End Learnable Histogram Filters". In: 2017.

[45] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[46] Peter Karkus et al. "Differentiable Mapping Networks: Learning Structured Map Representations for Sparse Visual Localization". In: *International Conference on Robotics and Automation (ICRA)*. 2020.

[47] S. Eslami et al. "Neural scene representation and rendering". In: *Science* 360 (June 2018), pp. 1204–1210. DOI: 10.1126/science.aar6170.

[48] Dan Rosenbaum et al. "Learning models for visual 3D localization with implicit mapping". In: *ArXiv* abs/1807.03149 (2018).

[49] Piotr Mirowski et al. "Learning to Navigate in Cities Without a Map". In: (Mar. 2018).

[50] Xiao Ma et al. *Discriminative Particle Filter Reinforcement Learning for Complex Partial Observations*. 2020. arXiv: 2002.09884 [cs.LG].

[51] Maximilian Igl et al. "Deep Variational Reinforcement Learning for POMDPs". In: (June 2018).

[52] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[53] Manolis Savva et al. "Habitat: A Platform for Embodied AI Research". In: *CoRR* abs/1904.01201 (2019). arXiv: 1904.01201. URL: http://arxiv.org/abs/1904.01201.

[54] Fei Xia et al. "Gibson Env: Real-World Perception for Embodied Agents". In: *CoRR* abs/1808.10654 (2018). arXiv: 1808.10654. URL: http://arxiv.org/abs/1808.10654.

[55] Xiao Ma et al. "Particle Filter Recurrent Neural Networks". In: *CoRR* abs/1905.12885 (2019). arXiv: 1905.12885. URL: http://arxiv.org/abs/1905.12885.

[56] Abhishek Das et al. "Neural Modular Control for Embodied Question Answering". In: *CoRR* abs/1810.11181 (2018). arXiv: 1810.11181. URL: http://arxiv.org/abs/1810.11181.

[57] *Cyberbotics*. 2019. URL: https://cyberbotics.com.

[58] *Python*. 2019. URL: https://docs.python.org.

[59] Sebastian Raschka. *Python Machine Learning*. Packt Publishing, 2015.

[60] *Keras*. https://keras.io. [Online; accessed 11-April-2020]. 2020.

[61] Simo Särkkä. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013.