**UNIVERSITÉ LIBRE DE BRUXELLES**
**Faculté de Sciences**
**Département d'Informatique**

Preparatory work for the Master Thesis

# Machine Learning for Probabilistic Robotics with Webots

Joan Gerard

**Promotor:** Prof. Gianluca Bontempi

**Academic year 2019**

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background and objectives of the thesis

Webots is a robot simulation tool created by the Swiss Federal Institute of Technology in Lausanne (EPFL) in December 1996. It is used since then in industry and academia for researching purposes. It became open source in December 2018 under the Apache 2.0 license [1]. This tool has more than 40 models of robots; moreover, it allows users to create new custom models.

Webots was used for different research projects using machine-learning techniques. Szabó[2], for instance, created a module for metric navigation using a modification of the Khepera robot. The objective was to build a map of a few-square-meter size environment with some obstacles on it. Szabó used these steps to build his module: sensor interpretation, integration over time, pose estimation, global grid build and exploration (See [3]).

The robot pose over the space environment is uncertain and it becomes difficult to estimate as a result of non-systematic errors. Even though it can be approximated using odometry techniques as it is shown in Szabó work, as times goes the estimated position is phased out due to the error accumulation until it becomes useless. The use of parametric and non-parametric filters can be useful to reduce this error. Thus, the long-term objective of the master thesis will be to exploit the simulation benefits of Webots to introduce Machine Learning techniques together with non-parametric filters for robot positioning estimation, independently from the kind of robot used for in-door environments. The selected programming language is Python 3.7 in a MacOS environment.

The purpose of this work is to present the state of the art, to show an introduction to Webots tool and non-parametric filters.

## 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 |

# Chapter 2

# State of the art

## 2.1 Webots

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; however, user support and consultancy are not. It is available for Windows, Ubuntu Linux and MacOS [1].

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 useful to develop the project. For a detailed description please refer to the user guide[2] or the reference manual[3].

### 2.1.1 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 2.1 shows the main graphic interface of Webots. It can be divided in 5 panels:

[1] 2019

[2] `https://cyberbotics.com/doc/guide/index`

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

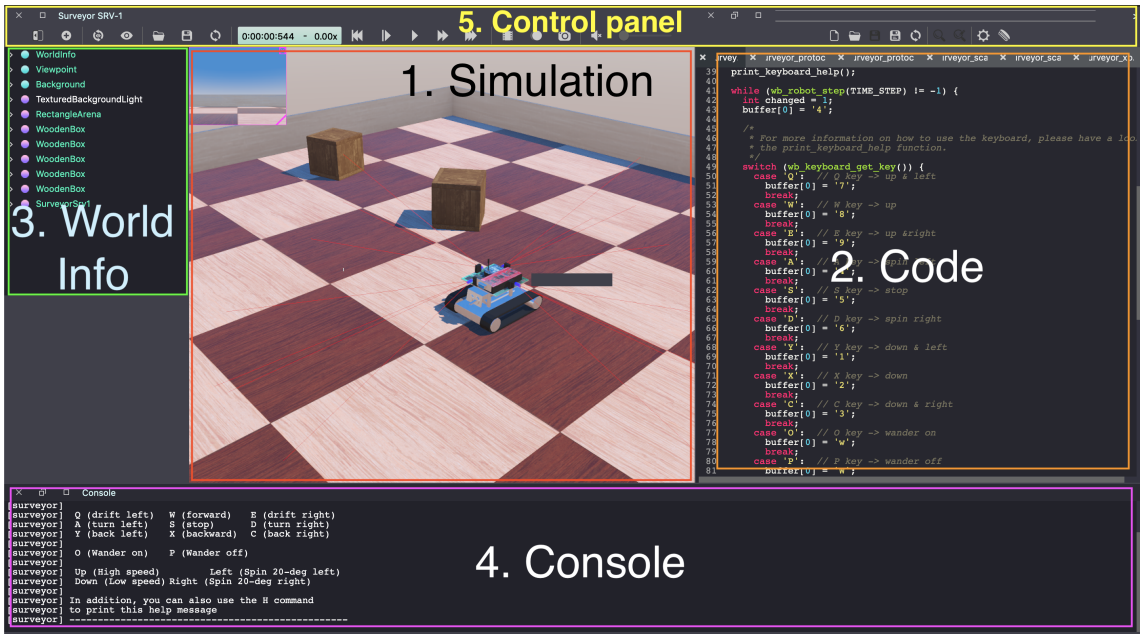



Figure 2.1: Webots graphic interface

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.
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.

### 2.1.2 Webots with Python 3.7 and TensorFlow

Python was created in 1990 by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands as a successor of a language called ABC[4]. Nowadays it has became one of the most popular programming languages for data science[5].

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[4]; 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 2.2.

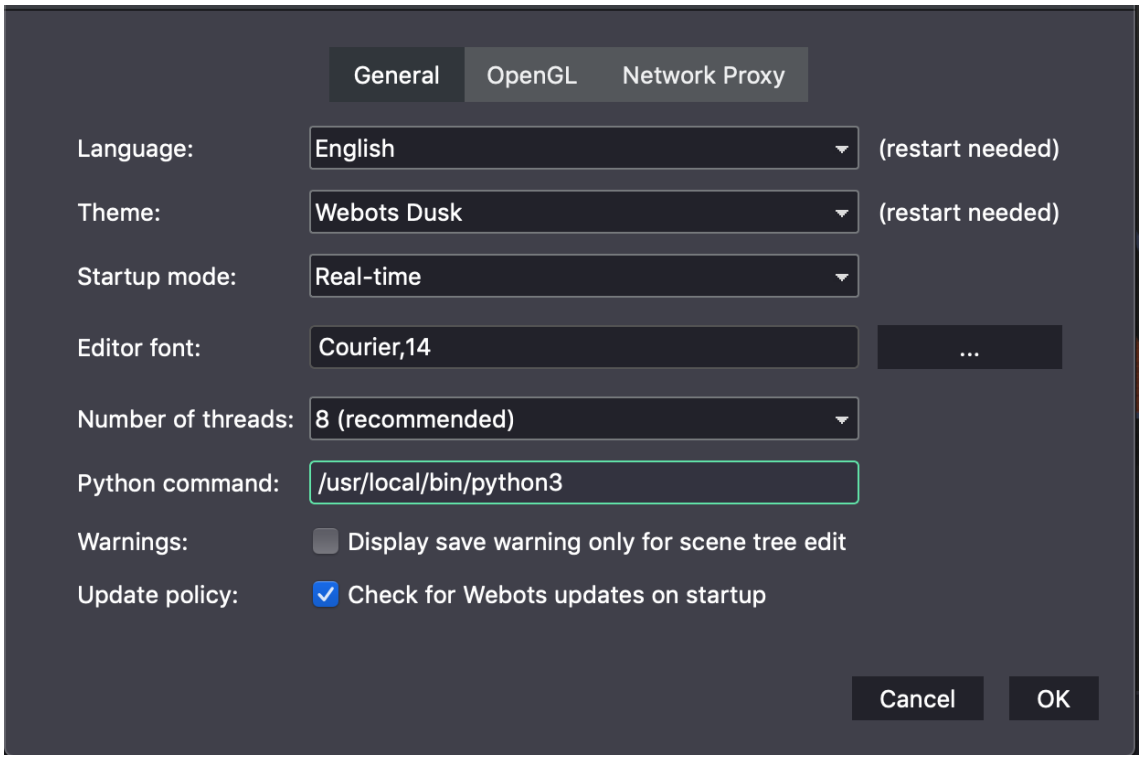



Figure 2.2: Configure Python3.7

Python allows to install third-party libraries like TensorFlow which 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[5].

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 2.1 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 2.1: Verify correct installation of TensorFlow

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

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.

### 2.1.3 Robots and world creation

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.

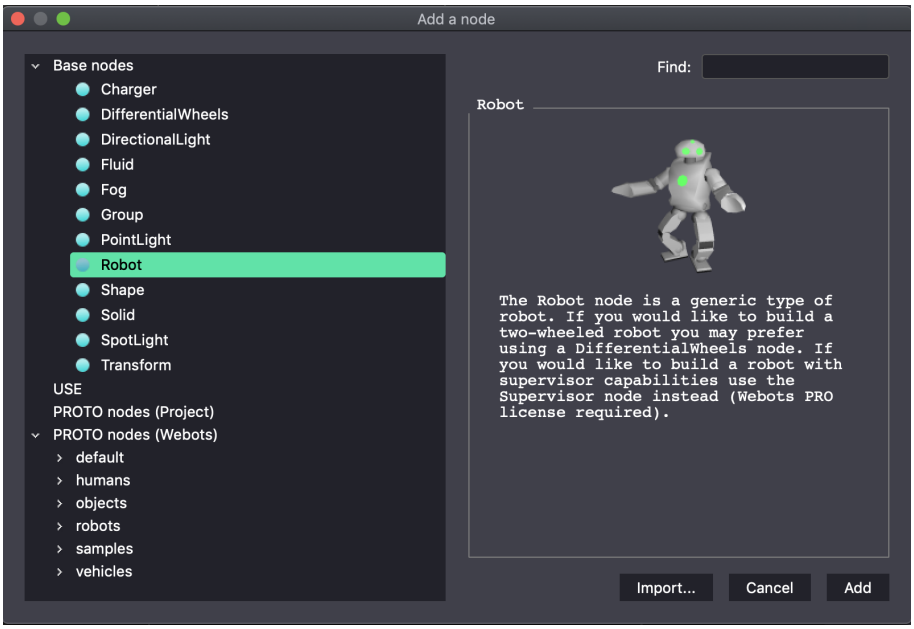
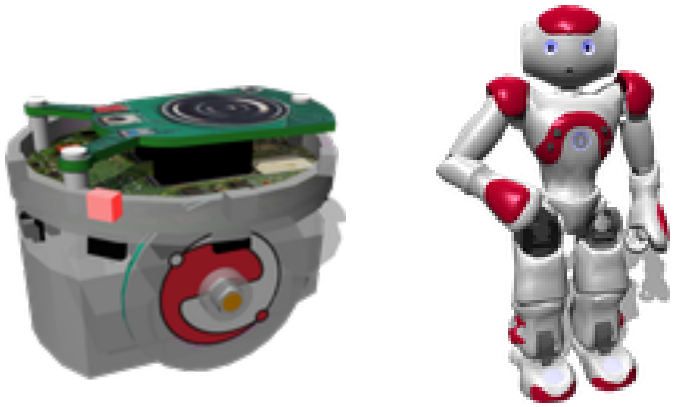



Figure 2.3: World structure

For adding a node into the world we can press the `Add object` button that will open the window shown in figure 2.3. 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 2.4a) model which is used very often in research, to more complex robots as the very well known humanoid Nao (figure 2.4b) are some examples of a total of 44 3D modeled robots that are available to use within the simulator tool.

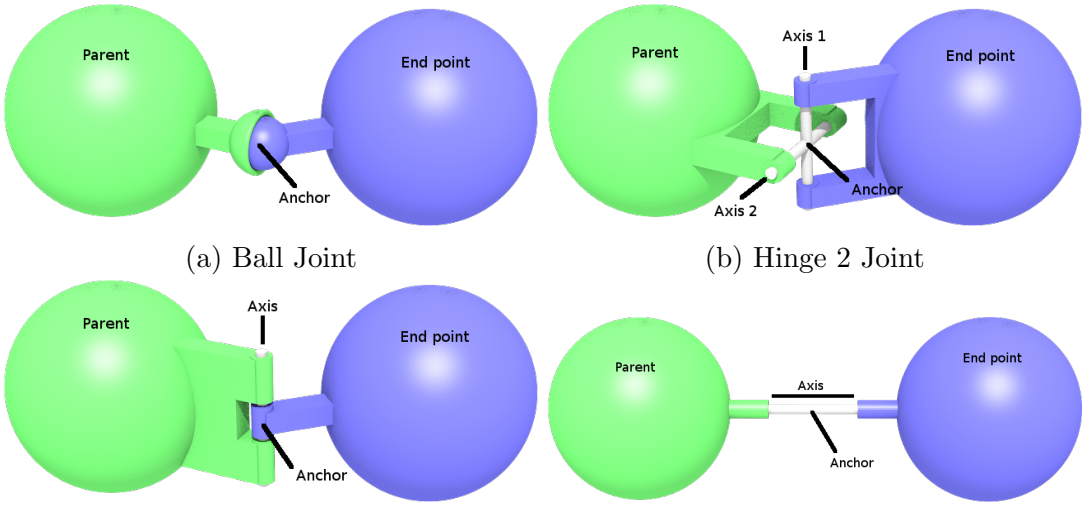
Another powerful feature of Webots is to create a custom robot model from scratch using a tree-based structure of solid nodes which are virtual objects with physical properties. Thus a robot is made of a set of solid nodes put them together using joint



(a) e-puck (b) Nao

Figure 2.4: Robots

nodes which are abstract nodes that models mechanical joints. Figure 2.5 shows the different types of joints that can be used.




(a) Ball Joint (b) Hinge 2 Joint

(c) Hinge Joint (d) Slider Joint

Figure 2.5: Different types of joint nodes

**Source**: Webots documentation

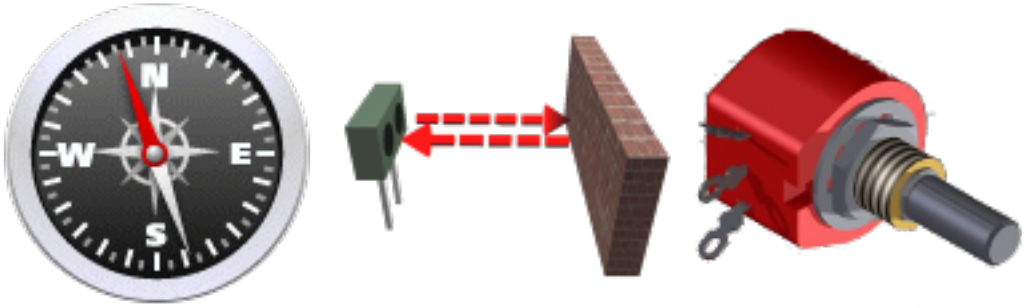
A position sensor can be added into the devices property of a joint node in order to monitor it. In like manner a rotational motor node can be added for actuating it. Thus, putting all together a simplistic version of a custom robot can be created based on the tree information and properties given to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

### 2.1.4 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 2.6: 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 2.1 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[1]. 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 2.1: Lookup table of distance sensor

**Source**: Webots documentation

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

Figure 2.7 shows the relation between the current sensor values and the mapped values within the associated noise.

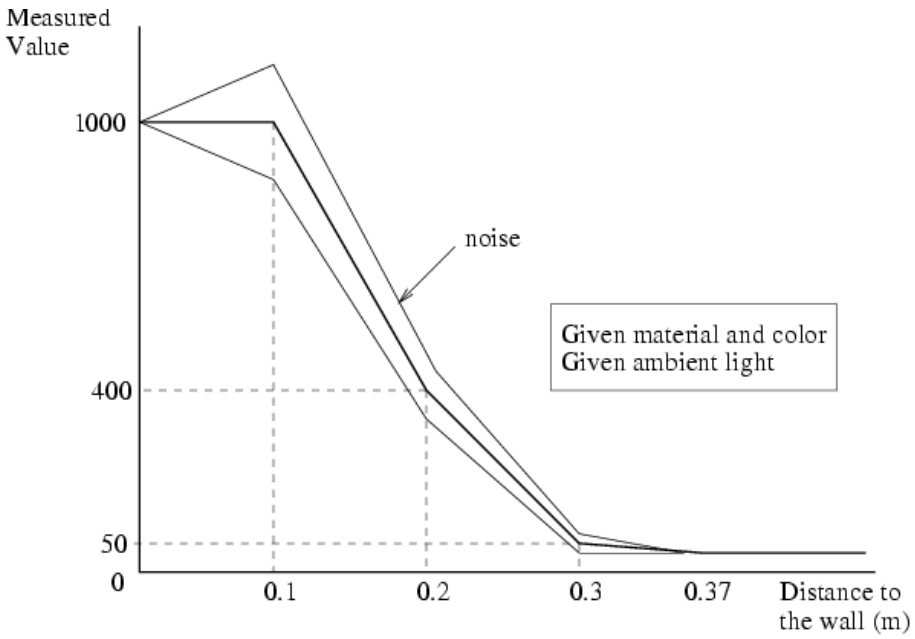



Figure 2.7: Sensor response versus obstacle distance

**Source**: Webots documentation

### 2.1.5 Actuators

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

## 2.2 Probabilistic Robotics

Robotics, according to Thrun et al.[6], 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[7], planet exploration[8], agriculture[9], construction[10], entertainment[11] 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[12], 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[13]. 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[14].

### 2.2.1 State

State, according to Thrun et al.[6], 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[15]. Figure 2.8 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 [16]).

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

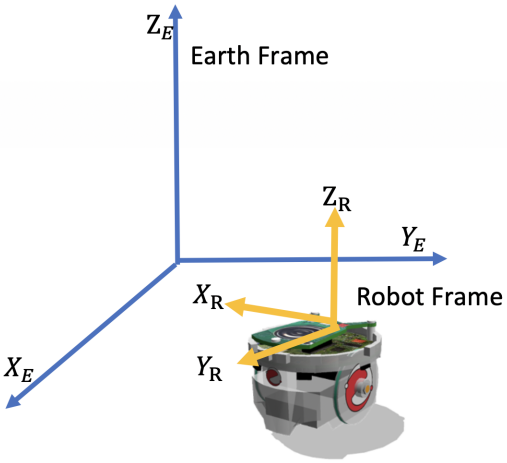



Figure 2.8: Local vs Global reference frame

### 2.2.2 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[15]. 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 [15][6].

The control data is information about the change of state in the environment[6]. 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 $z_{1:t} = z_1, z_2, ..., z_t$.

### 2.2.3 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)[17]. 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[13].

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

### 2.2.4 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 $t = 0, 1, ..., t-1$[18]. 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[17].

### 2.2.5 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[17].

There are two types of localization methods categorized by the increased degree of difficulty as stated in Ferreira et al.[12]:

- *Local techniques* (also called position tracking[6]), 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 [19]), 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.[6] 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[13]. An example of a global technique is shown in section 2.2.7 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 2.10 illustrates. A more extensive and explicative example can be found in [13] and [20] 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 2.6). 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.

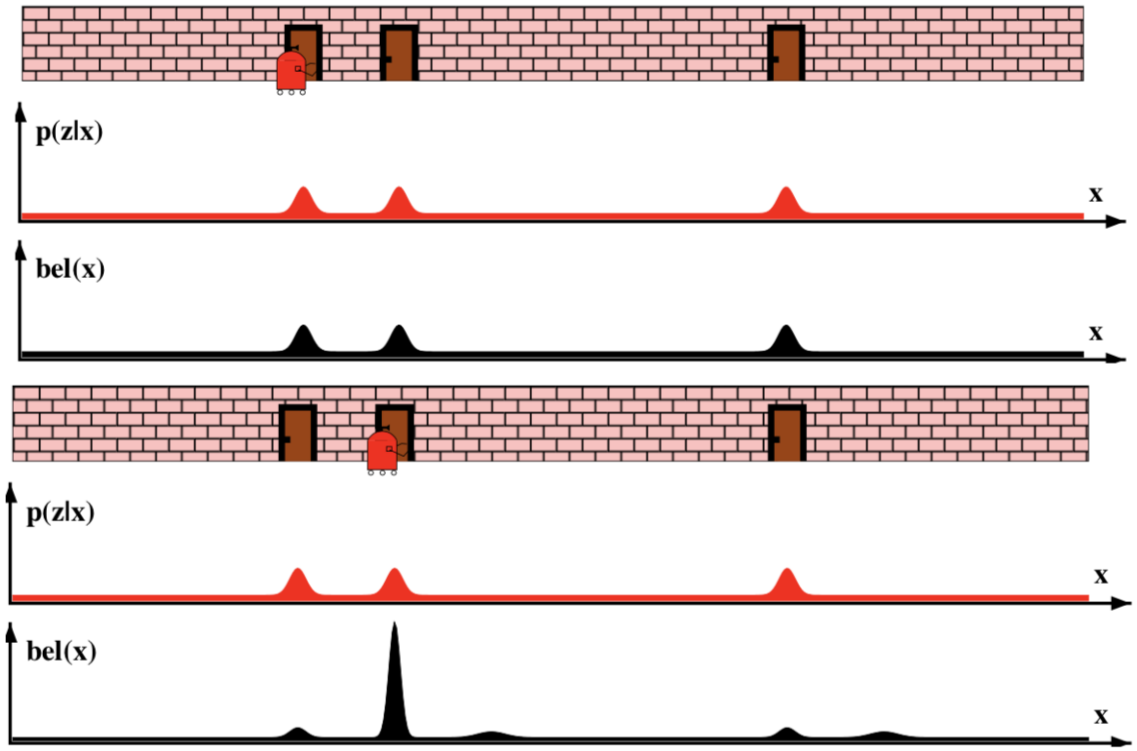



Figure 2.9: Probability density when robot senses.

**Source**: Extracted from Thrun et al. Probabilistic Robotics[13]

### 2.2.6 Bayes Filter

As claimed by Borriello et al.[20] 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[6]. This is generally done in two steps as it was mentioned in section 2.2.3: 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.

**Algorithm 1:** Bayes Filtering Algorithm[6]

**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** $bel(x_t)$

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[21] and was first implemented by NASA in the Apollo program to estimate the trajectory of the space capsule in real time[22]. It takes noisy data and put the noise apart in order to get information with less uncertainty[23]. It works with continuous states and it represents the beliefs by the first and second moment[20] 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 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)[24].

- 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[25][6]. 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[26][27]. Another improvement of EKF is the Invariant Extended Kalman Filter (IEKF) used for continuous-time systems with discrete observations[28].

- The *Grid-based Filters* split the environment into small grids, each containing a belief of the true position state[20]. 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[29].

- *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[20].

- 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[30].

### 2.2.7 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)[31]. Each particle has associated a weight that represents the contribution to the overall estimate of the posterior[30]. 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[6]. Figure 2.10 illustrates the use of MCL for robot localization.

**Algorithm 2:** Particles Filtering Algorithm. Adapted from [6].

**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[32], 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[33]. 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 $a_j$[34][30].

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[35].

Murray et al. present an alternative to accelerate the resampling technique eliminating the cumulative sum over weights using two schemes based on Metropolis[36] and rejection samplers making the problem of resampling parallelizable using graphical processing units (GPU)[37]. 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[38].

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

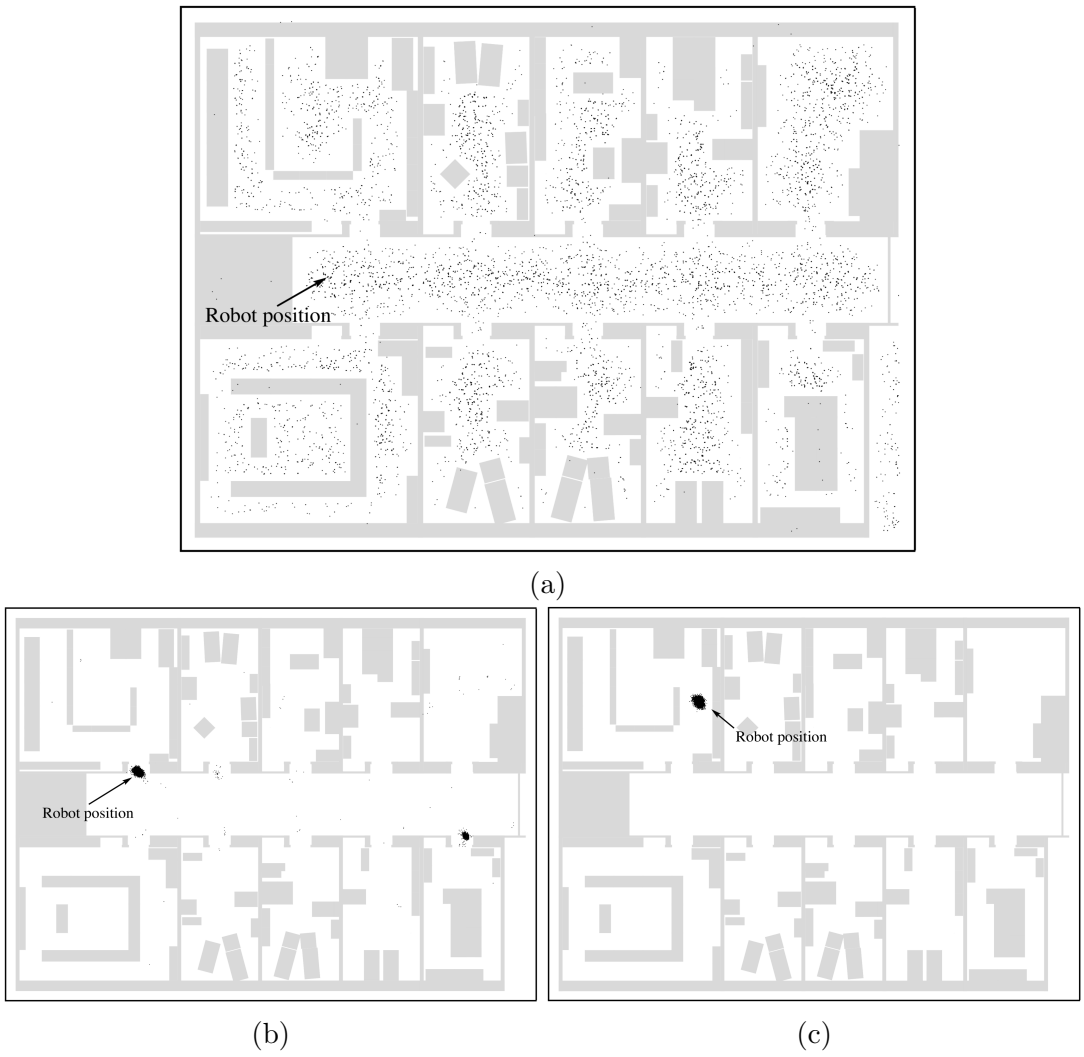



Figure 2.10: A robot localization example using PF. (a) The robot has a global uncertainty about its position in the map, (b) it handles some hypothesis about where it is after executing certain control actions, (c) it is quite sure where it is in the environment.

**Source**: Particles Filters in Robotics[39].

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[40]. Montemerlo et al. proposed FastSLAM, an algorithm that recursively estimates the full posterior distribution over robot pose and landmark locations[41] and it is one of the most robust algorithms for mapping in-door environments[39].

# Chapter 3

# Preliminary Results

This section is focused on showing the basic preliminary results obtained.

A short-term goal is to resolve the local positioning problem using Bayes Filter algorithm. Such algorithm has two steps: the prediction and the error correction. The first step will be creating a custom robot model on Webots, then programing a controller to move it around avoiding collision with the walls, implementing odometry techniques to determine an estimation of its position in a rectangular small arena and then creating a data set collecting sensor measurements data along with positioning in order to create a model that will allow the robot to correct the odometry data.

The source code of the implementation can be found on GitHub[1].

## 3.1 Robot And The Environment

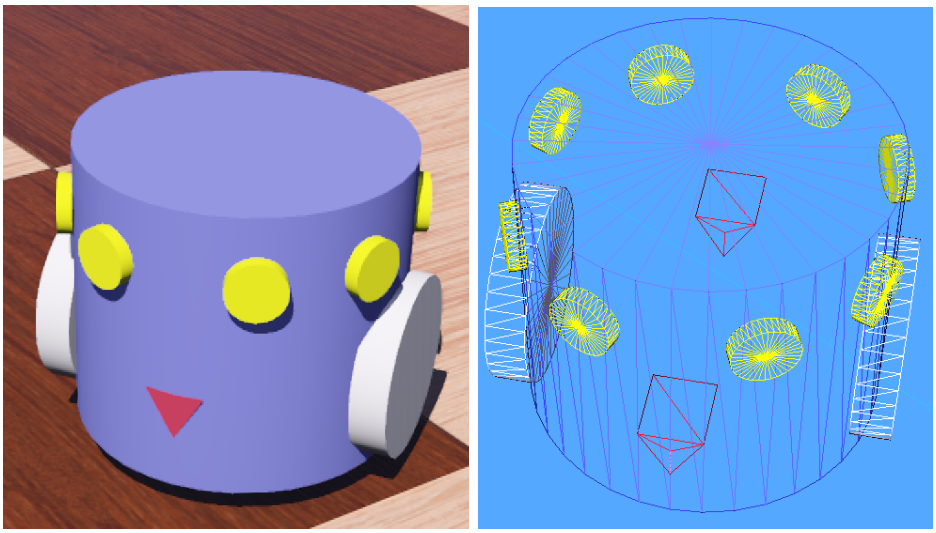
In order to implement the Bayes Filter algorithm an e-puck-style robot model will be created. The main benefit to do so is the high control and personalization on the robot model that brings when running different experiments. Thus a list with the main components is given below.

- A *cylindric body* in which all the components will be attached.
- Two *cylindric wheels* with hinge joints, each with a position sensor and a rotational motor. The former serves to obtain the position of the wheel at a given time step, the later controls the velocity and direction of the wheel.
- Eight *laser-type distance sensors* around the body of the robot that measure the distance between the robot and the closest wall.

[1] https://github.com/joangerard/webots-thesis/tree/master/src

- A *compas* sensor for measuring the robot orientation over the virtual north.

Figure 3.1 illustrates the created custom robot model under two different views. The white cylinders are the wheels, the purple is the body, and the yellow are the distance sensors. This robot is based on the version provided by the examples of custom robot models on the Webots Documentation. It is slightly personalized, mainly on the type of distance sensors used.



(a) Plain rendering (b) Wireframe rendering

Figure 3.1: Custom robot view

The robot is positioned at the center of an arena of 1.5m x 2m surrounded by walls. This position is communicated to the robot and it becomes its initial state. In order to avoid obstacles while moving, a simple algorithm has been used that turns around the robot when the front sensors tell the presence of a solid object near 15 cm of distance.

## 3.2 State Prediction

Webots introduces the concept of supervisor which is a class forming part of the Webots API that simulates human actions in the environment. For instance, it can restart the simulation and put the robot in a random position or it can measure the trajectory of the robot at each step. Notice that this last concept is strictly restricted to the Webots simulator and it is not part of the command line of any real robot. Combined along with the compass sensor data will be useful to get the true robot state $\mathbf{x_t} = [x_t, y_t, \theta_t]^T$ where $x_t$ and $y_t$ are the coordinates of the robot translated to the GRF and $\theta_t$ is the orientation of the robot over the virtual north at time $t$. The predicted state will be $\hat{\mathbf{x}}_\mathbf{t} = [\hat{x}_t, \hat{y}_t, \hat{\theta}_t]^T$ and it can be updated recursively based on the previous given state and thus at time $t+1$ the state will be $\hat{\mathbf{x}}_{\mathbf{t+1}} = \hat{\mathbf{x}}_\mathbf{t} + [\Delta x, \Delta y, \Delta\theta]^T$. This corresponds to the second line of the Bayes Filter algorithm (see algorithm 1), that is, $\overline{bel}(\mathbf{x_t})$.

Cyberbotics' Robot Curriculum website[42] provides a good explanation about how to implement odometry technique and which parameters need to be calibrated in order to have a precise implementation. These parameters need to be found experimentally since they are not measurable directly with enough precision: the distance of increment for the left wheel, the right wheel and the distance between both wheels. To have those values well configured four tests need to be performed:

- *Increments per tour:* the number of motor increments made per one complete wheel rotation. While the robot have gone forward, the wheel had made one complete tour.
- *Axis wheel ratio:* the diameter of the wheels divided by the distance between them. The robot turns around on its own axis and the goal is to end up at the same position that it began.
- *Wheels diameters:* the robot follows a square trajectory and it should end up at the same position where it started. If that is not the case, the diameter of one wheel can differ to the other as it is shown in figure 3.2.
- *Scaling factor:* it configures the scale of the trajectory.

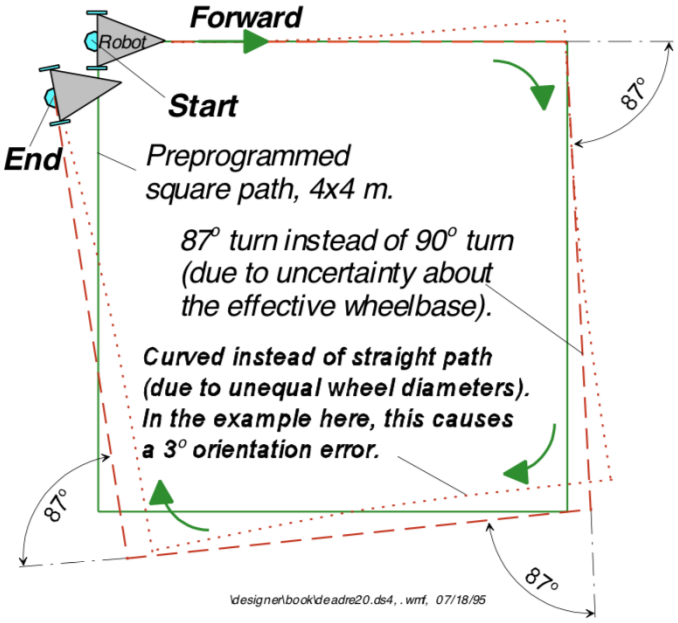
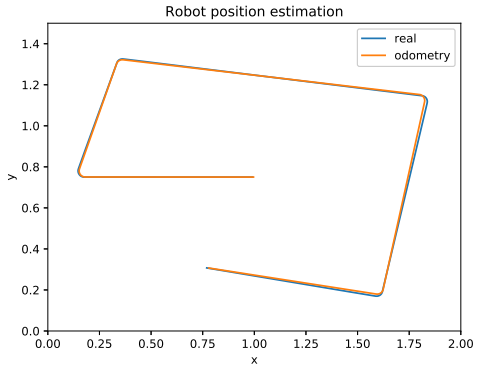



Figure 3.2: Wheel diameters test.

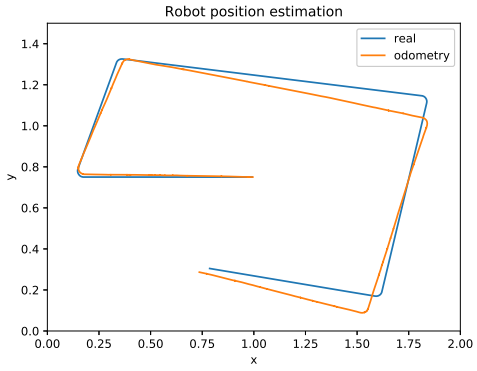
**Source**: Where am I?[19]

After performing the tests and having an estimation of the corresponding values the results are shown in figure 3.3. The plots were taken in an aerial-view way where the axis represents the width and height of the arena and the lines mark the trajectory of the robot. On the one hand the blue line represents the true state pose of the robot obtained by the supervisor and compass data, on the other hand the orange line represents the predicted state pose of the robot using odometry techniques. The left-side image was obtained by running one simulation of the mobile robot with zero

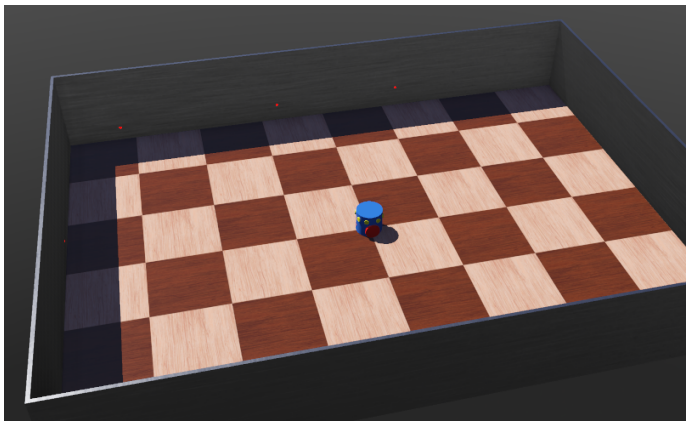
noise on its position sensors. Even though the odometry measurements approximate very well the true pose, they are still phased out due to the fact that the parameters used by odometry were found experimentally and thus some systematic error was introduced. The right-side image shows how the predicted robot state quickly starts to diverge from the true state due to noise associated with the position sensors that affects the data returned by the odometry technique. Thus as time goes the robot predicted state will be far from the true state; however, this problem should be resolved during the correction step in the Bayes Filter algorithm.




(a) Position Sensor Noise = 0




(b) Position Sensor Noise = 0.1



(c) Robot in the arena

Figure 3.3: Odometry results

## 3.3 State Correction

The second part of the Bayes Filter algorithm corresponds to the error correction state which implies calculating the current posterior distribution based on the previous posterior distribution which in practice corresponds to using the predicted state given by odometry and then using this data to calculate the probability of perceived the sensor measurements data in order to make the state correction. This corresponds to the third line of the Bayes Filter algorithm (see algorithm 1), that is $bel(x_t)$.

There are two steps for performing the state correction. First, the robot-moving

simulation runs for ten minutes in fast mode while collecting its distance sensor measurements along with true state pose data to train a model that can be capable of predicting distance sensor measurements given a certain state pose, that model will be represented by the function $\hat{z}(\mathbf{x_t})$. Second, given a predicted state $\hat{\mathbf{x}}_\mathbf{t}$ provided by the odometry technique and sensor measurements $z_t$, a set of $m$ hypothesis regarding the true robot position is created using some variations of the given predicted state (described in more detail in section 3.3.2), that is, $\mathcal{X} = \{\hat{\mathbf{x}}_\mathbf{t}^\mathbf{1}, \hat{\mathbf{x}}_\mathbf{t}^\mathbf{2}, ..., \hat{\mathbf{x}}_\mathbf{t}^\mathbf{m}\}$. The hypothesis state which is closer to the true state, named best state and denoted as $\hat{\mathbf{x}}_\mathbf{t}^*$, will be chosen minimizing the function in equation 3.1 and thus the odometry data will be corrected using this state value. Notice that this corresponds to searching the values of x that maximizes the conditional probability of distance sensor measurements given a state $x$ under the Bayes Filter approach.

$$\hat{\mathbf{x}}_\mathbf{t}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} (\hat{z}(\mathbf{x}) - z_t)^2 \approx \arg\max_x \eta\, p(z_t|x) \tag{3.1}$$

### 3.3.1 Collecting Pose and Sensor Data

A data set is constructed using the `pandas` library where the features are the true robot state ($\mathbf{x_t}$) and the target variables are the sensor measurements ($\mathbf{z_t}$), the objective to do so is to generalize and predict sensor measurements given a certain robot state.

| Features ($\mathbf{x_t}$) | | | Target ($\mathbf{z_t}$) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_t$ | $y_t$ | $\theta_t$ | $z_1$ | $z_2$ | $z_3$ | $z_4$ | $z_5$ | $z_6$ | $z_7$ | $z_8$ |
| 0.98 | 0.75 | 179.99 | 1.02 | 0.76 | 0.76 | 1.04 | 1.04 | 0.76 | 0.76 | 1.02 |
| 0.91 | 0.74 | 180.00 | 0.95 | 0.76 | 0.76 | 1.12 | 1.12 | 0.76 | 0.76 | 0.95 |
| 0.72 | 0.74 | 180.00 | 0.74 | | 0.76 | 1.33 | 1.33 | 0.76 | 0.76 | 0.74 |
| 1.03 | 1.04 | 208.22 | 0.99 | 0.67 | 0.40 | | 0.92 | 1.19 | 1.00 | 1.30 |
| ... | | | | | | | | | | |
| 0.29 | 0.58 | 212.58 | 0.25 | 0.31 | 0.88 | 1.06 | 1.68 | 0.98 | 0.54 | 0.47 |

Table 3.1: Built Data Set

Table 3.1 shows part of the collected data. Notice that some sensor measurements are missing intentionally. Once the simulation has run for ten minutes in fast mode the data set is saved into a csv file and then it was split in 80% training and 20% test. Then the `sklearn` library is used to create an ensemble of models employing the Random Forest technique with ten trees[2]. The training data is used to train the ensemble of models and finally the accuracy is measured to be 99%. This ensemble of models will be represented as a function $\hat{z}(\mathbf{x_t})$.

[2]Usually random forest works better with a bigger amount of trees but due to performance constrains only ten trees are used.

### 3.3.2 Correcting Odometry Data

The algorithm for correcting the odometry data is divided in three parts. First, it takes a range of state values built on the predicted state $\hat{\mathbf{x}}_\mathbf{t} = [\hat{x}_t, \hat{y}_t, \hat{\theta}_t]$, that is, $\mathcal{X} = \{\hat{\mathbf{x}}_\mathbf{t}^\mathbf{1}, \hat{\mathbf{x}}_\mathbf{t}^\mathbf{2}, ..., \hat{\mathbf{x}}_\mathbf{t}^\mathbf{m}\}$ these hypothesis are built based on all the possible state values near the predicted state defined by two parameters $\delta$ and $\omega$ for modifying the range of the $\hat{x}_t, \hat{y}_t$ coordinates and the range of the $\hat{\theta}_t$ angle respectively. The bigger these values, the higher the number of hypothesis to be evaluated. Second, it iterates over all hypothetic state values $\mathbf{x} \in \mathcal{X}$, and evaluates how probable is to be in state $\mathbf{x}$ comparing the data provided by the distance sensors of the robot $z_t$, with the predicted distance sensor data provided by the function $\hat{z}(\cdot)$ previously obtained by training the model. Finally, it selects the best state value, that is, the hypothetic state value $\hat{\mathbf{x}}_\mathbf{t}^*$ whose distance sensor prediction was more approximated to the real one using equation 3.1. The algorithm is illustrated below.

**Algorithm 3:** Correction State Algorithm: Calculate Best State Value

**input :** $\hat{x}_t$, $\hat{y}_t$, $\hat{\theta}_t$, $\delta$, $\omega$, $z_t$, $\hat{z}(\cdot)$
**output:** $\hat{\mathbf{x}}_\mathbf{t}^*$

$\mathcal{X} \leftarrow \emptyset$
$Z \leftarrow \emptyset$
$x_{range} \leftarrow [\hat{x}_t + \delta, \hat{x}_t - \delta]$
$y_{range} \leftarrow [\hat{y}_t + \delta, \hat{y}_t - \delta]$
$\theta_{range} \leftarrow [\hat{\theta}_t + \omega, \hat{\theta}_t - \omega]$

`// First part: creation of set` $\mathcal{X}$
**foreach** $x \in x_{range}$ **do**
  **foreach** $y \in y_{range}$ **do**
    **foreach** $\theta \in \theta_{range}$ **do**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{\langle x, y, \theta \rangle\}$
    **end**
  **end**
**end**

`// Second part: calculate predictions`
**foreach** $\mathbf{x} \in \mathcal{X}$ **do**
  $error \leftarrow (\hat{z}(\mathbf{x}) - z_t)^2$
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{error\}$
**end**

`// Third part: best state`
$i \leftarrow$ get index of $min(\mathcal{E})$
$\hat{\mathbf{x}}_\mathbf{t}^* \leftarrow \mathcal{X}[i]$
**return** $\hat{\mathbf{x}}_\mathbf{t}^*$

An implementation in Python can be found in appendix A.2.

## 3.4 Results

The model for predicting distance sensor measurements is trained when the simulation starts and the algorithm to calculate the best state value is executed each 450 robot steps, that is, approximately one meter of robot traveled distance. Additionally, the distance sensors are configured to have a standard deviation of 0.01 using a lookup table as it is shown in section 2.1.4 and the position sensors are configured to have a standard deviation of 0.1 using its noise attribute. The experiment runs with $\delta = 10$ and $\omega = 3$ and thus a square area of 20 cm around the predicted state is considered to make the correction step. Figure 3.4 shows the results. The green points represent the best state value found, that means that the algorithm was called three times during the simulation and thus the predicted data is then corrected.

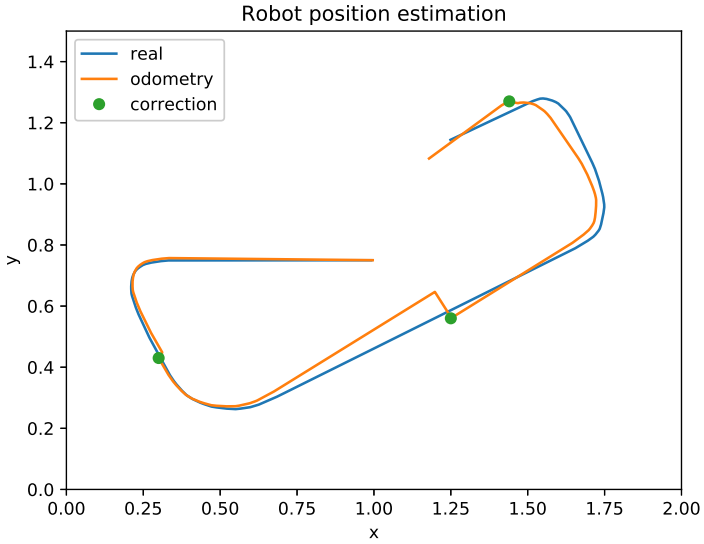



Figure 3.4: Pose correction

Notice that the number of hypothesis (the size of set $\mathcal{X}$) will be in this case $|x_{range}| \times |y_{range}| \times |\theta_{range}| = 20 \times 20 \times 3 = 1200$ and each hypothesis will be processed by the prediction function $\hat{z}(\cdot)$ which uses ten trees to take a decision and therefore this method is computationally heavy and thus, it requires high computational resources which is not usually the case for a simple robot.

## 3.5 Discussion and Further Work

The Random Forest technique shows a very high accuracy on the test data caused by highly repeated data which means that the robot could have been passed by the same place multiple times while collecting data, making easy to predict a value already seen during training. Additionally, figure 3.4 shows that the first two state corrections are very accurate but the third one is not. This can be caused because the data set used to train the model was obtained with only one run of the simulation

and thus the predictions are not very accurate when the robot visits a place that was not previously visited during the collection data process. A solution to the two previously aforementioned problems would be to execute multiple simulations while collecting data initializing the robot position into a random place in the arena, and to add randomness behavior to the control actions when moving the robot in order to obtain a more generic dataset.

The algorithm has a poor performance when the number of hypothetic state values is high. A solution to this problem could be executing the Correction State algorithm more periodically, using smaller values for the $\delta$ and $\omega$ parameters. Additionally, some other prediction techniques could be more efficient than Random Forest and thus a comparison among them should be performed.

The size of the arena is ideal for running simulations but real-world scenarios can be less predictive and far more complex. Thus a more realistic environment will be created.

# Conclusion

The main purpose of this work was to present the Webots simulator and to describe the state-of-the-art regarding robot localization techniques along with Bayesian filters and finally show some preliminar results.

The present work will be used next year for the master thesis in order to implement a more generic algorithm for robot localization using more advanced techniques under the Bayesian filters literature.

# 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[43].
- $p(z_t|z_{1:t-1}, u_{1:t})$ is the normalization constant defined as $\eta$

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.

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

## A.2 Correction State Algorithm in Python

The complete file can be found on GitHub [1].

```
1  def correct_state(x, y, theta, sensors_data, delta = 10, omega = 3):
2
3      # corresponds to the E set
4      errors = []
5
6      # corresponds to the X set
7      predictions = []
8
9      xrange = [1/100 for l in range(max(0, int(x*100) - delta), min(
       MAX_X*100, int(x*100) + delta), 1)]
10     yrange = [1/100 for l in range(max(0, int(y*100) - delta), min(int(
       MAX_Y*100), int(y*100) + delta), 1)]
11     thetarange = [l for l in range(max(0, int(theta) - omega), min(360,
        int(theta) + omega), 1)]
12
13     print("XRANGE----------")
14     print(x)
15     print(xrange)
16
17     print("YRANGE----------")
18     print(y)
19     print(yrange)
20
21     print("THETARANGE----------")
22     print("theta: ", theta)
23     print(thetarange)
24
```

[1] https://github.com/joangerard/webots-thesis/blob/master/src

```
25      for i in xrange:
26          for j in yrange:
27              for k in thetarange:
28                  error, bad_data = predictor.predict(i, j, k,
    sensors_data)
29                  if not bad_data:
30                      predictions.append([i, j, k])
31                      errors.append(math.log(error))
32
33      if len(errors) > 0:
34          ix = errors.index(min(errors))
35          return predictions[ix]
36
37      return -1
```

# Bibliography


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

[2] Richárd Szabó. "Navigation of simulated mobile robots in the Webots environment". In: *Periodica Polytechnica, Electrical Engineering* 47 (Jan. 2003).

[3] Sebastian Thrun. "Learning metric-topological maps for indoor mobile robot navigation". In: *Artificial Intelligence* 99 (Feb. 1998), pp. 21–71.

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

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

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

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

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

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

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

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

[21] 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.

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

[23] 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.

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

[25] 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.

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

[27] 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.

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

[29] 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).

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

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

[32] 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).

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

[34] 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.

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

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

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

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

[39] 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.

[40] 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.

[41] 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.

[42] *Cyberbotics' Robot Curriculum/Advanced Programming Exercises*. 2018. URL: `https://en.wikibooks.org/wiki/Cyberbotics%5C%27_Robot_Curriculum/Advanced_Programming_Exercises`.

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