UNIVERSITÉ LIBRE DE BRUXELLES
Faculté de Sciences
Département d'Informatique

Preparatory work for the Master Thesis

Machine Learning for simulated control tasks 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 that was created in 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, 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 [2]. 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 to the kind of robot used. 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
$x_t \in \mathbb{R}$	State x at time t
$z_t \in \mathbb{R}$	Measurement z at time t
$u_t \in \mathbb{R}$	Action u at time t
$bel(x_t)$	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 $\{x_1, x_2, ..., x_t\}$

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

Chapter 2

State of the art

2.1 Webots

Webots was created by Cyberbotics Ltd. a spin-off company from the EPFL and has been developing it since 1998. It 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 it 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 about the simulated world.
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.

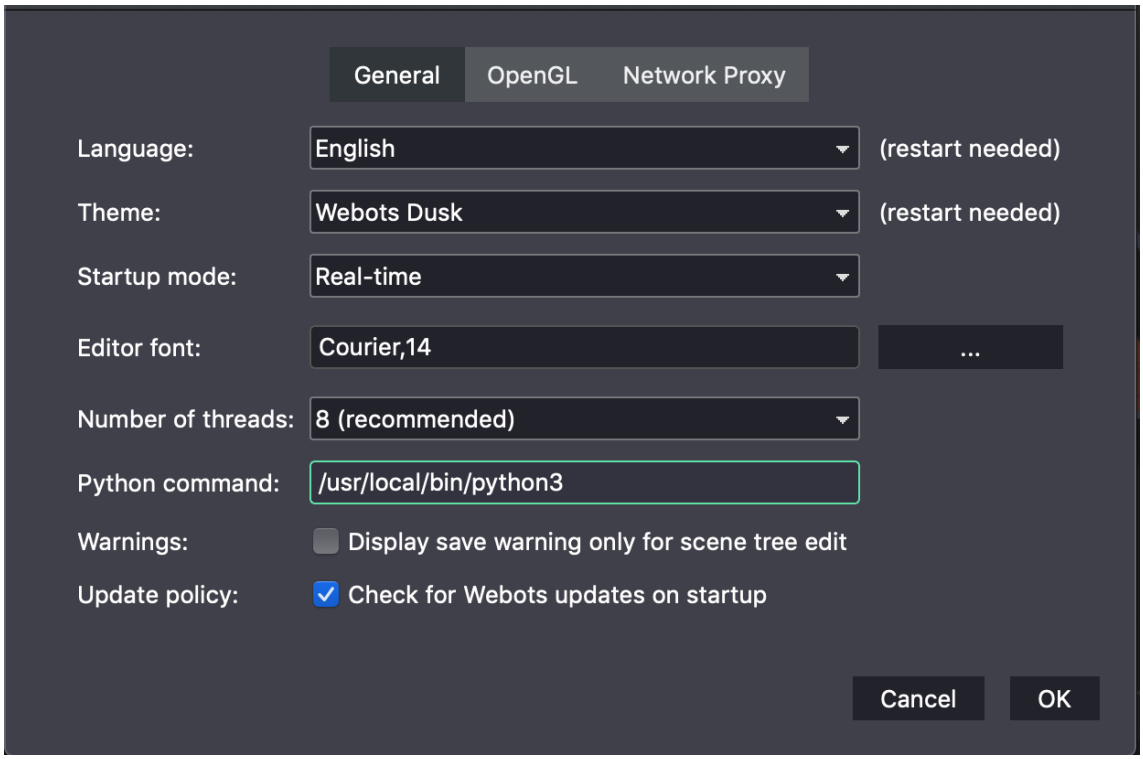


Figure 2.2: Configure Python3.7

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 was 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.

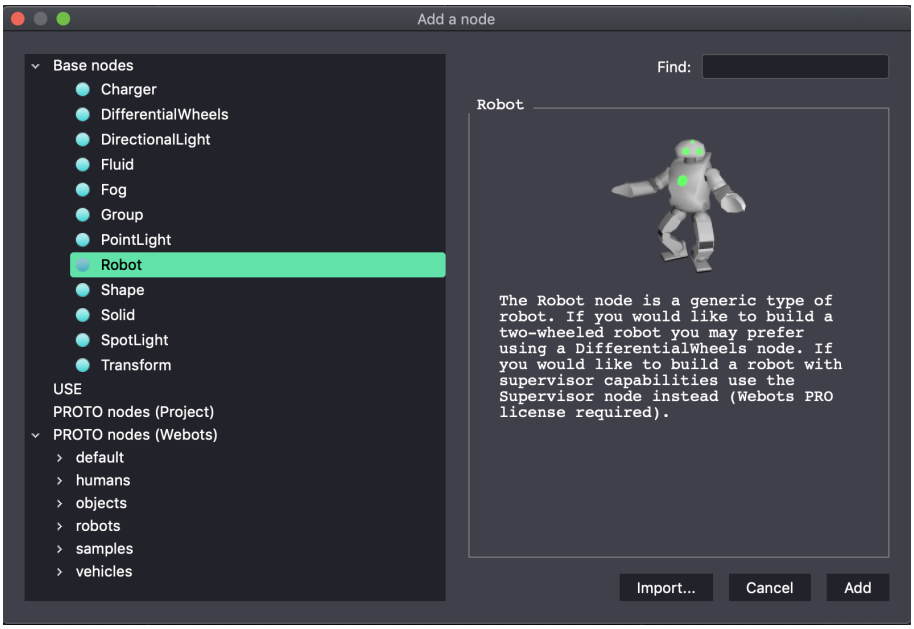
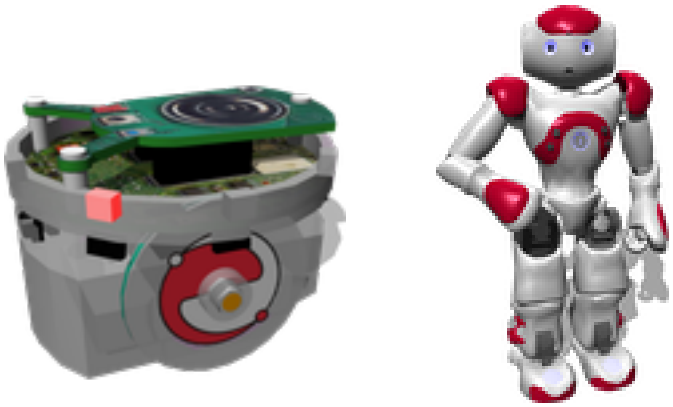


Figure 2.3: World structure

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.

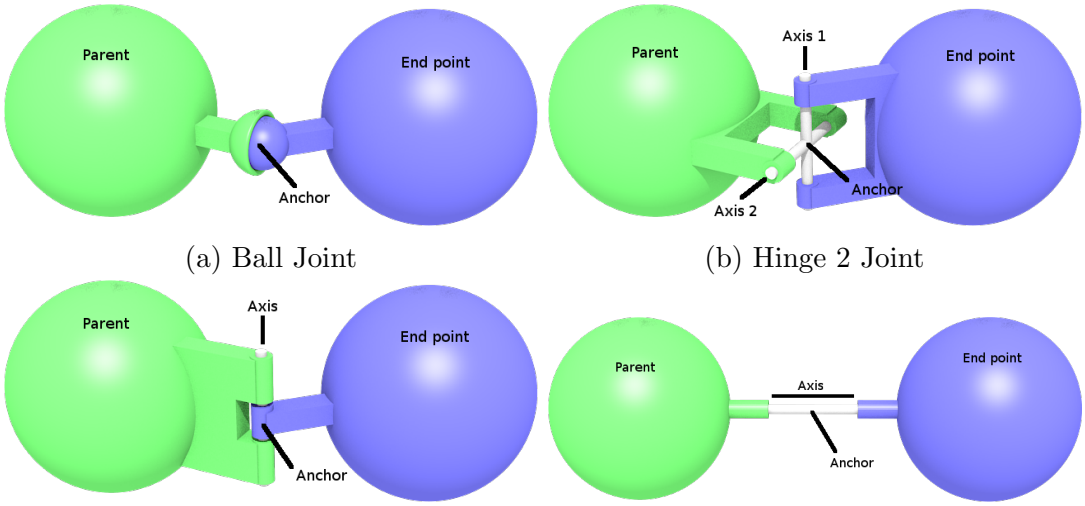
Another powerful feature of Webots is to create a custom robot model from scratch using a tree-based structure of solid nodes which are virtual objects with physical properties. Thus a robot is made of a set of solid nodes put them together using joint



(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

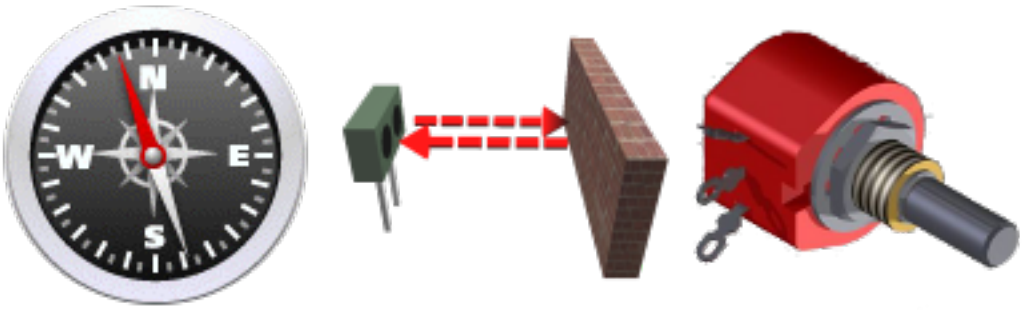
A position sensor can be added into the devices property of a joint node in order to monitor it. In like manner a rotational motor node can be added for actuating it. Thus, putting all together a simplistic version of a custom robot can be created based on the tree information and properties given to 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 position of the virtual north as a 1, 2 or 3-axis vector in relation to the position of the robot. 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 percent 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 distance from

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

the robot, it will return a value of 50 ± 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.

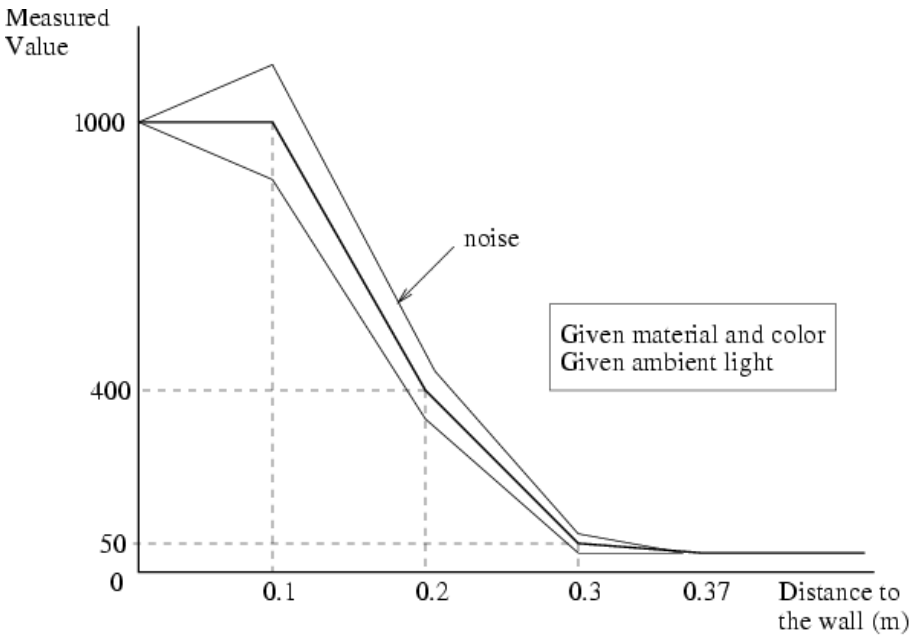


Figure 2.7: Sensor response versus obstacle distance

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 that it has is ordered to walk 1 meter forward from its current position; even if it might be expected to cover that distance, due to some errors whether the robot speed was high and it slipped or the noise produced by the actuators is significant, after finishing executing the control action the robot is found in an uncertain position. 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.

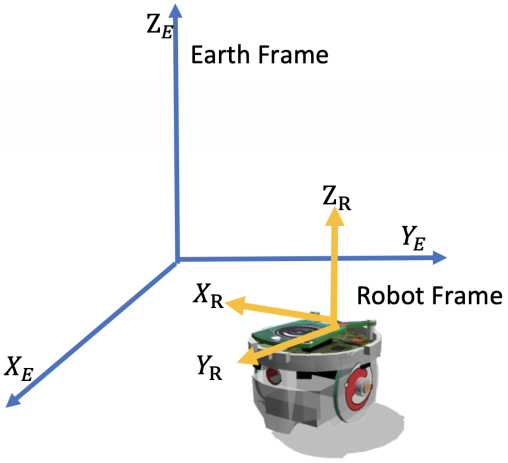


Figure 2.8: Local vs Global reference frame

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 a single point or multiple points in the map. The former is known as single-hypothesis belief and the later as multiple-hypothesis belief[17]. Thus a belief assigns a probability to each location in the environment. The higher the probability, more approximate the position of the robot is to the true state in that specific location. 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].

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, very well known 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 Localisation 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.[17] 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.
- *Localisation:* 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 localisation but both are subjected to noise and thus the problem of localisation 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 localisation problem into a difficult task.

There are two types of localisation 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 localisation 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]. That means that a robot can handle multiple ambiguous positions. 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. 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.

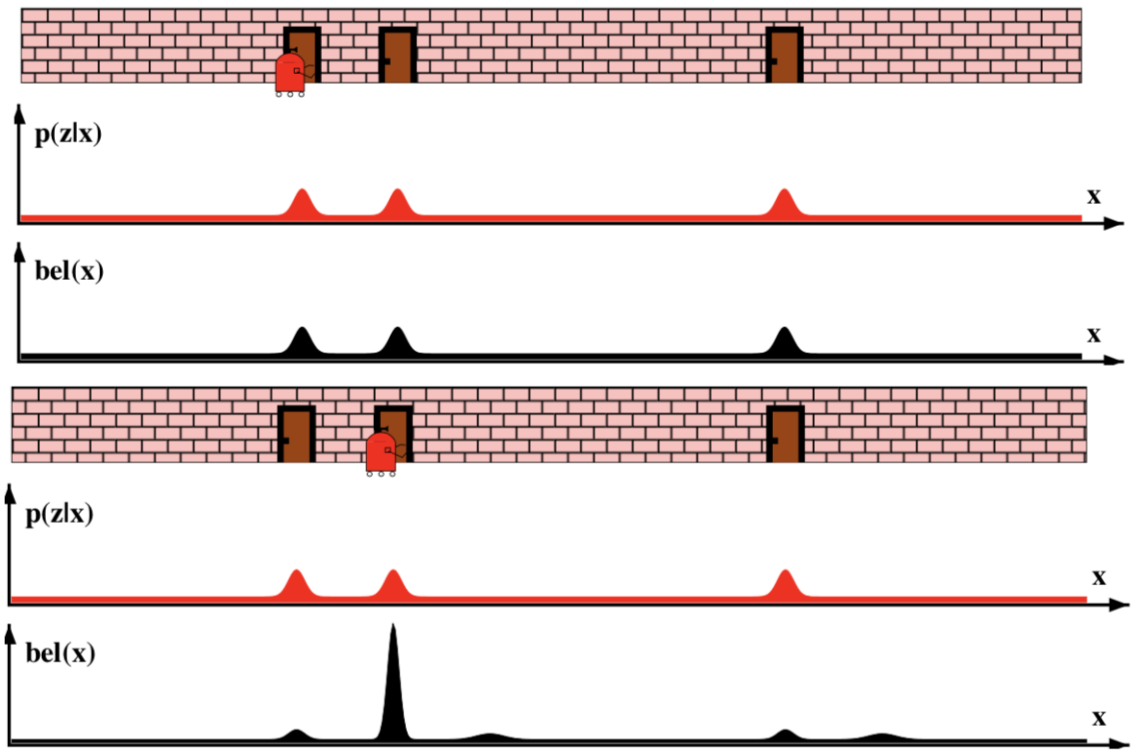


Figure 2.9: Probability density when robot senses.

Source: Extracted from Probabilistic Robotics [13]

2.3 Bayes Filter

As claimed by Borriello et al. 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[20].

Bayes filter algorithm is recursive. It needs the previous posterior distibution to calculate the current posterior distribution. The algorithm is described below.

Algorithm 1: Bayes Filtering Algorithm[6]

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

foreach *state* x *in* 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 $bel(x_t)$

The current posterior distribution is calculated as it is shown in equation 2.4.

$$p(x_t|z_{1:t}, u_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})\, p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \tag{2.4}$$

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[21].
- $p(z_t|z_{1:t-1}, u_{1:t})$ is the normalization constant defined as η

Thus equation 2.4 can be summarized as follows:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta\, p(z_t|x_t, z_{1:t-1}, u_{1:t})\, p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.5}$$

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 to

predict z_t. Thus the likelihood model for the measurements can be simplified as follows:

$$p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.6}$$

Therefore equation 2.5 reduces to:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta\, p(z_t|x_t)\, p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.7}$$

In equation 2.8 the prior distribution is expanded.

$$p(x_t|z_{1:t-1}, u_{1:t}) = \int p(x_t|x_{t-1}, u_t)\, p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) dx_{t-1} = \overline{bel}(x_t) \tag{2.8}$$

Replacing equation 2.8 in 2.7, equation 2.9 is obtained which is calculated by the algorithm in list 1 third line.

$$p(x_t|z_{1:t}, u_{1:t}) = \eta\, p(z_t|x_t)\, \overline{bel}(x_t) \tag{2.9}$$

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. DOI: 10.1126/scitranslmed.aad9398.

[8] Geoffrey Landis. "Robotic exploration of the surface and atmosphere of Venus". In: *Acta Astronautica* 59 (Oct. 2006), pp. 570–579. DOI: 10.1016/j.actaastro.2006.04.011.

[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. DOI: 10.25165/j.ijabe.20181104.4278.

[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. DOI: 10.1016/j.autcon.2018.01.009.

[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. DOI: 10.1007/978-3-319-92058-0_23.

[12] João Filipe Ferreira and Jorge Dias. *Probabilistic Approaches to Robotic Perception*. Jan. 2014, pp. 3–36. DOI: 10.1007/978-3-319-02006-8_1.

[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. DOI: 10.1109/ROBOT.2002.1013331.

[15] Reza Jazar. *Theory of applied robotics: Kinematics, dynamics, and control (2nd Edition)*. Jan. 2010, pp. 7, 17–20. DOI: 10.1007/978-1-4419-1750-8.

[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] L. Liao et al. "Bayesian Filtering for Location Estimation". In: *IEEE Pervasive Computing* 2.03 (July 2003), pp. 24–33. ISSN: 1536-1268. DOI: 10.1109/MPRV.2003.1228524.

[21] Simo Särkkä. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013. DOI: 10.1017/CBO9781139344203.