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 Abbreviations

EPFL	Swiss Federal Institute of Technology in Lausanne
ROS	Robot Operative System
3D	Three-dimensional

Chapter 2

State of the art

2.1 State of the art

2.1.1 Webots

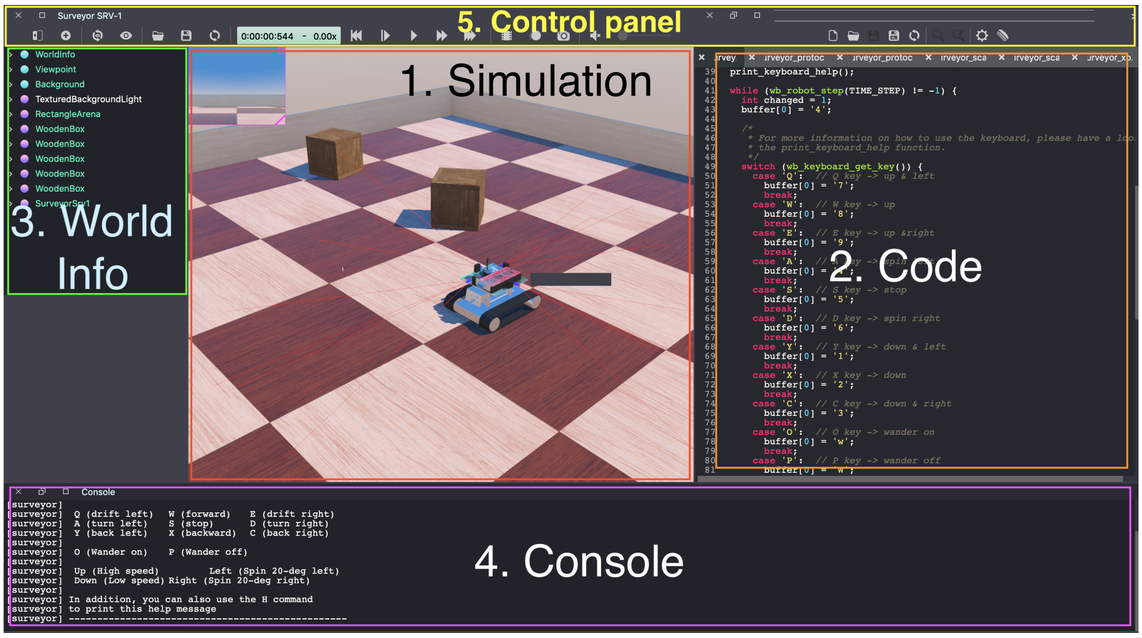
Webots was created by Cyberbotics Ltd. a spin-off company from the EPFL and has been developing it since 1998. It currently[1] employs 6 people in Lausanne, Switzerland to continuously develop Webots according to customers needs. Cyberbotics provides consulting on both industrial and academic research 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].

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.

[1] 2019
[2] `https://cyberbotics.com/doc/guide/index`
[3] `https://cyberbotics.com/doc/reference/index`



```
print_keyboard_help();

while (wb_robot_step(TIME_STEP) != -1) {
  int changed = 1;
  buffer[0] = '4';

  /*
   * For more information on how to use the keyboard, please have a loo
   * the print_keyboard_help function.
   */
  switch (wb_keyboard_get_key()) {
    case 'Q':  // Q key -> up & left
      buffer[0] = '7';
      break;
    case 'W':  // W key -> up
      buffer[0] = '8';
      break;
    case 'E':  // E key -> up &right
      buffer[0] = '9';
      break;
    case 'A':  // A key -> spin left
      buffer[0] = '4';
      break;
    case 'S':  // S key -> stop
      buffer[0] = '5';
      break;
    case 'D':  // D key -> spin right
      buffer[0] = '6';
      break;
    case 'Y':  // Y key -> down & left
      buffer[0] = '1';
      break;
    case 'X':  // X key -> down
      buffer[0] = '2';
      break;
    case 'C':  // C key -> down & right
      buffer[0] = '3';
      break;
    case 'O':  // O key -> wander on
      buffer[0] = 'w';
      break;
    case 'P':  // P key -> wander off
      buffer[0] = 'w';
```

```
[surveyor]
[surveyor]  Q (drift left)   W (forward)    E (drift right)
[surveyor]  A (turn left)    S (stop)       D (turn right)
[surveyor]  Y (back left)    X (backward)   C (back right)
[surveyor]
[surveyor]  O (Wander on)    P (Wander off)
[surveyor]
[surveyor]  Up (High speed)         Left (Spin 20-deg left)
[surveyor]  Down (Low speed) Right (Spin 20-deg right)
[surveyor]
[surveyor] In addition, you can also use the H command
[surveyor] to print this help message
[surveyor] ---------------------------------------------------
```

Figure 2.1: Webots graphic interface

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

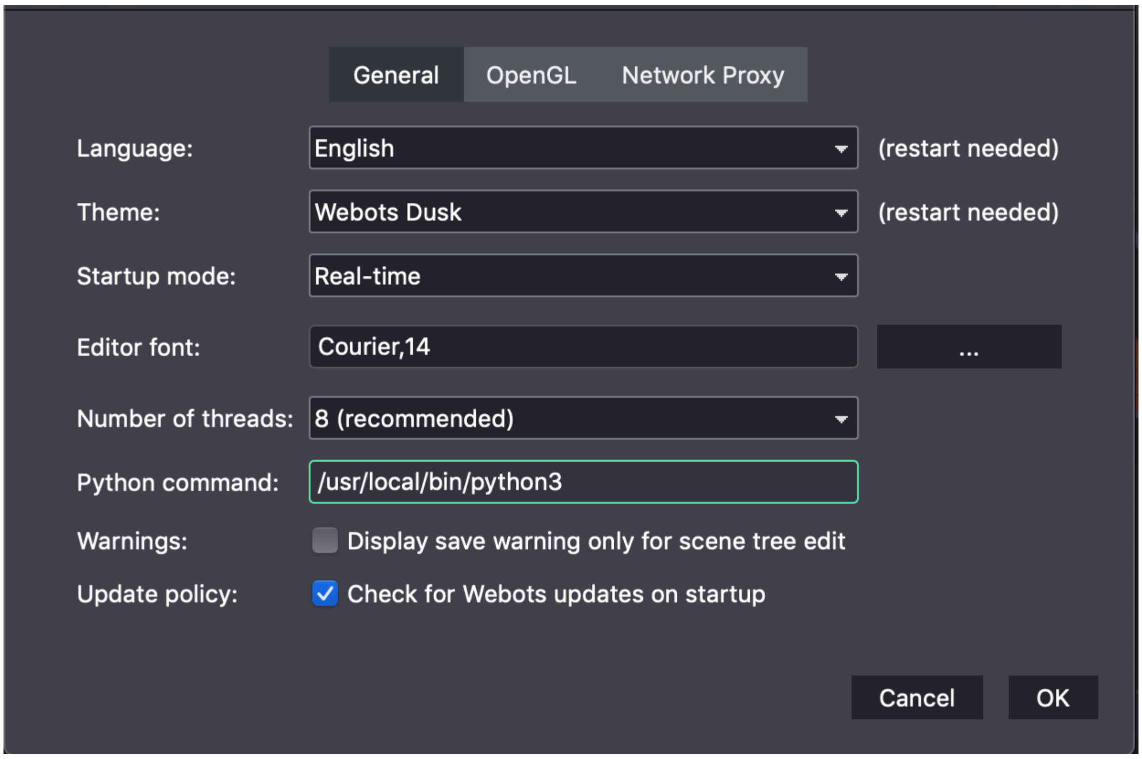
1. Simulation: graphic visualization of the simulated world objects.
2. Code visualization: robot controller code editor.
3. World Information: information about the simulated world.
4. Console: program execution output stream.
5. Control panel: set of buttons that controls the simulation execution.

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.

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.



Language: English (restart needed)
Theme: Webots Dusk (restart needed)
Startup mode: Real-time
Editor font: Courier,14 ...
Number of threads: 8 (recommended)
Python command: /usr/local/bin/python3
Warnings: ☐ Display save warning only for scene tree edit
Update policy: ✓ Check for Webots updates on startup

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.

```python
import tensorflow as tf

verifier = tf.constant('TensorFlow was installed correctly.')
sess = tf.Session()
print(sess.run(verifier))
```

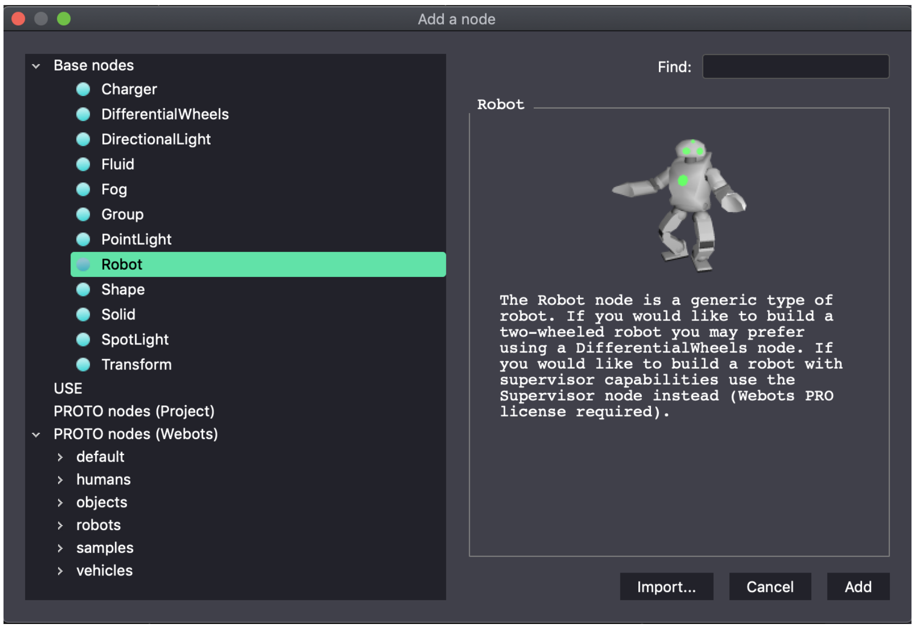
Listing 2.1: Verify correct installation of TensorFlow

[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.

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.

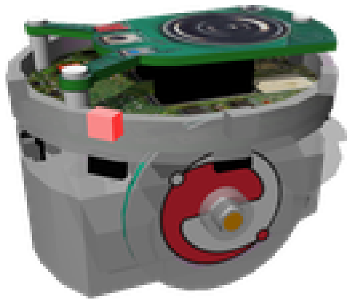


The Robot node is a generic type of robot. If you would like to build a two-wheeled robot you may prefer using a DifferentialWheels node. If you would like to build a robot with supervisor capabilities use the Supervisor node instead (Webots PRO license required).

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

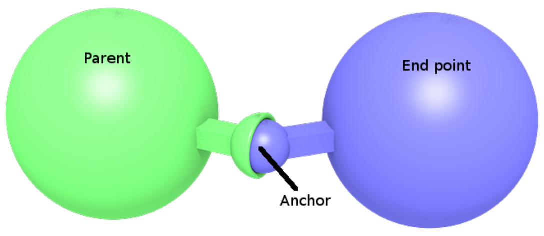


(a) e-puck

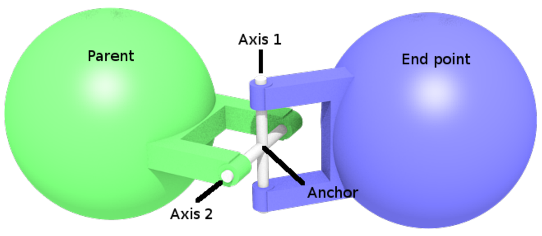
(b) Nao

Figure 2.4: Robots

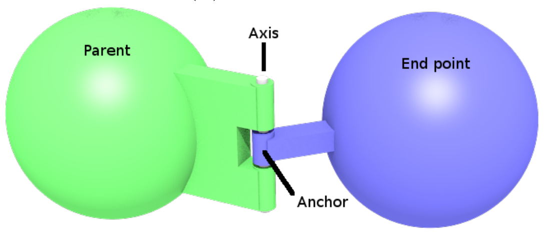
nodes which are abstract nodes that models mechanical joints. Figure 2.5 shows the different types of joints that can be used.



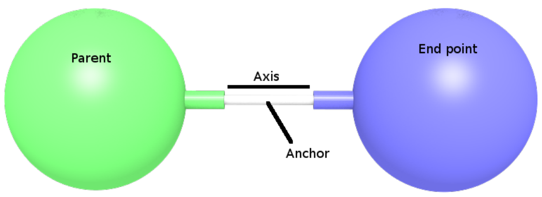
(a) Ball Joint



(b) Hinge 2 Joint



(c) Hinge Joint



(d) Slider Joint

Figure 2.5: Different types of joint nodes

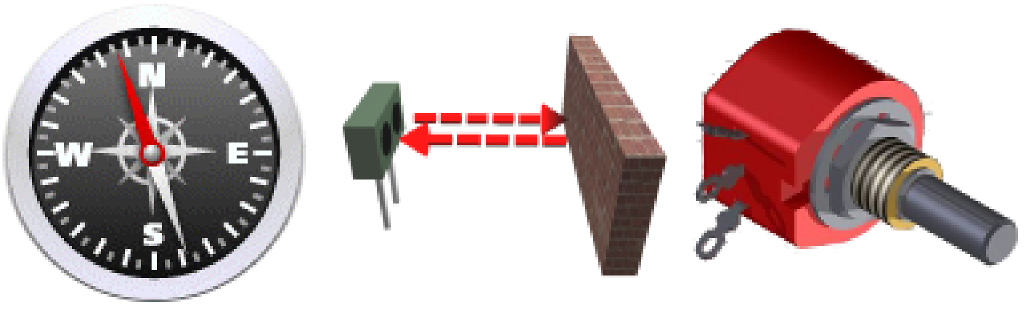
Source: Webots documentation

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.

Sensors and actuators

Webots has a wide range of generic and commercially available sensors that can be plugin into a custom robot. On the 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.

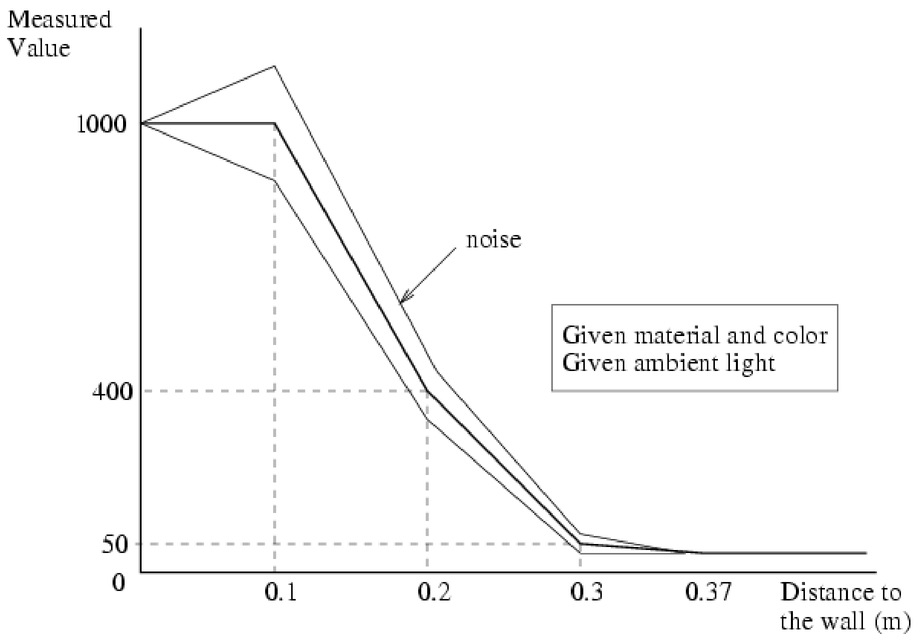


Figure 2.7: Sensor response versus obstacle distance

Source: Webots documentation

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.