Lab class: Autonomous robotics
Software information

Institut für Neuroinformatik

August 16-20, 2021

1 Installing the Software

If you run into any problems while installing, do not hesitate to contact jan.tekuelve@ini.rub.de via mail or Element *before* the start of the course on Monday.

1.1 Getting Webots

For this course, we will be using the Webots 3D robot simulation environment. It can be obtained at
    https://cyberbotics.com/#download
Install it by following the instructions at
    https://cyberbotics.com/doc/guide/installing-webots
A more extensive tutorial then is provided here can be found at
    https://cyberbotics.com/doc/guide/getting-started-with-webots
and a technical documentation of the software at
    https://cyberbotics.com/doc/reference/nodes-and-api-functions

1.2 Getting Python

For this course, we will use the Python programming language.[1] If you haven't already installed python there are many tutorials on the internet explain how to do this in detail. If you are a Windows user the most straightforward way is likely to use the python package in the Microsoft Store.

If Webots has problems to find python after the installation, check whether **Tools→Preferences→Python command** points to your installation directory, something like "C:\Program Files\Python39\python".

---

[1] https://www.python.org/downloads/

2 Details on Webots

2.1 The World

Webots is organized around the scene tree at the left side of the window which contains a complete description of the simulated environment. Each scene tree starts with a *WorldInfo* and a *Viewpoint* node. The former contains global information like the strength of gravity or the friction parameters between materials. It also holds the *basicTimeStep* field that determines the length of each step in the physics simulation. The *Viewpoint* meanwhile has information about the camera position and whether to follow a certain robot while it moves.

Other nodes that are usually found are the *TexturedBackground* and the *TexturedBackgroundLight* which provide a more complex environment and the *RectangleArena* which describes the ground the robot moves on. New nodes can be added to the scene tree via the ⊕ button at the top of the window.

Some important types of nodes are:

Group   A simple type of node that contains several child nodes.

Transform This node rotates and moves its children.

Solid   Most physical objects are represented by a solid node. This type of node has a field *boundingObject* which describes the shape used by the collision detection. The field *physics* indicates, whether the solid is moved by external forces like collisions.

Shape   A *Solid* requires a *Shape* to be properly represented in the physical world. This shape consist of a *Geometry* that dictates the actual form the *Solid* takes and an *Appearance* that determines its color and texture.

Robot   A *Robot* is represented by a collection of *Solids* connected by (motorized) *Hinges*. It further contains the field *controller* which specifies the program that determines its behavior. We will exclusively use the E-Puck robots which come prepackaged with Webots and only needs to have a controller build and assigned. Note, that the E-Puck also comes in two slightly different versions. We will be using version 1.

2.2   The controller

Once the world is loaded and the simulation begins the robot starts the controller that is specified in the *controller*-field. Webots supports controllers written in C, C++, java, python, and Matlab. In this course, we will use python.

One can create a new controller by using the field **New Robot controller** under **Wizards**, but we already provide template project files, where you will fill in your code.

Since the editor provided in Webots is rather rudimentary it is advisable to use an external editor of choice.

To understand how a controller works consider this minimal example:

```python
from controller import Robot

robot = Robot()
timestep = int(robot.getBasicTimeStep())

motor = robot.getDevice('motorname')

ds = robot.getDevice('distance_sensor_name')
ds.enable(timestep)

while robot.step(timestep) != -1:
        val = ds.getValue()
        # Process sensor data here.
        motor.setPosition(10.0)
```

The first step is always importing the *Robot* module from the controller library to have the tools to "talk" to Webots. Then one needs to instantiate the robot object. If there is only one Robot in the world it does not need

to be specified. The line following this one extracts the time that passes in each simulation step. Usually, this value equals 32ms and it is required to initialize sensors and computing controller steps.

Next, we need to create handles for all the devices we will use. For this, we use the *getDevice()* function with the device name in the argument. Note that the name is different from the DEF you see in the scene tree and can be found in the *name*-field of the device. Sensors additionally need to be enabled by passing them the timestep-variable.

Now starts the actual work. Each time *robot.step(timestep)* is invoked, the simulation progresses one timestep and returns a value denoting the successful execution of the step. Progression of the simulation creates new sensor readings and applies motor commands. Therefore this function is usually called in a while-loop, to retrieve new sensor readings and issue new motor commands each timestep. Note that you can also use other loops if you just want to issue a fixed series of commands.

The motors can be controlled in three different ways, by specifying either the desired position, velocity, or torque (or force for linear motors). While the minimal example above uses the position control mode we will mostly use the velocity control mode. To do so we need to include the line

```
motor.setPosition(float('+inf'))
```

once directly after initializing the motor device and use

```
motor.setVelocity(v)
```

in the main loop to set the velocity to the value v.

2.3   List of relevant controller commands

`Robot()`	returns a handle to the robot
`int(robot.getBasicTimeStep())`	gets the current time step of the world
`robot.step(timestep)`	synchronizes the sensor and actuator data between Webots and the controllers and advances the simulation by one step. Must be used in every controller.

`robot.getDevice("left wheel motor")`	returns a handle to the left motor (analogous for right motor)
`motor.setVelocity(vel)`	sets the velocity of the motor in radians per second
`robot.getDevice("left wheel sensor")`	returns a handle to the left wheel encoder (analogous for right)
`robot.getDevice("ps"+str(i))`	returns a handle to the i-th infrared sensor, starting from i=0 for IR1
`sensor.enable(timestep)`	enables the sensor (proximity sensor or wheel encoder) for the current time step, which is required for reading out values
`sensor.getValue()`	reads out the value of the sensor