Webots

What is we bot?

Simulation softeware package, offers a rapid prototyping enviornment allowes user to create 3D virtual worls with pysics properties.

What we can do with webots?

- Mobile robot prototyping for academic research, automotive industry, aeronautics, etc
- Robot locomotion research (quadrupeds, legged robots, etc)
- Multi-agent research (swarm and collaborative robots)
- Adaptive behaviou research - genatic algorithm
- Teaching robotics
- Robot contests

Need to know to use Webots?

- Basic C, C++, Java. Python or matlab
- 3D computer graphics and VRML97 descriptive language if would like to create your own robot models or modify existing enviornments

Webot simulation:

1. World file (.wbt): defines onw or several robots and their enviornment, my depend on PROTO files(.proto)
2. Controller programs: use C/C++/Java/Python/MATLAB
3. Physics plugin: used to modify webots regular physics behavior, use C/C++

What is World?

A world is 3D description of the properties os robots and os their enviornment and containsdescription of every object: position, geometry, appearance, physical properties, typr of objects, etc.

It has hierarchical structures where objects contain other object. For example, a robot can contain two wheels, a distance sensor and a joint which itself contains a camera, etc. it only specifies the name of the controller that is required for each robot but it does not contains controller code. world file (.wbt) strored in the 'worlds' subdirectory of each webots projects

what is a controller?

Controllers can be written in any of the programming languages supported by Webots: C, C++, Java, Python or MATLAB. When a simulation starts, Webots launches the specified controllers, each as a separate process, and it associates the controller processes with the simulated robots.

Some programming languages need to be compiled (C and C++) other languages need to be interpreted (Python and MATLAB) and some need to be both compiled and interpreted (Java). For example, C and C++ controllers are compiled to platform-dependent binary executables (for example ".exe" under Windows). Python and MATLAB controllers are interpreted by the corresponding run-time systems (which must be installed). Java controller need to be compiled to byte code (".class" files or ".jar") and then interpreted by a Java Virtual Machine. The source files and binary files of each controller are stored together in a controller directory. A controller directory is placed in the "controllers" subdirectory of each Webots project.

Part 1: Install and run first simulation

Step 1. Download Webots from the link website

Step 2. Install and open the software in your system

Windows : follow the documentation

Step 3. Start new project

The wizards menu makes it easier to create new project and new controllers it has following menu items

- **New Project directory** - It creates a project directory. A project directory contains several subdirectories that are used to store the files related to a particular Webots project, i.e. world files, controller files, data files, plugins, etc. Webots remembers the current project directory and automatically opens and saves any type of file from the corresponding subdirectory of the current project directory.

- **New Robot Controller** - Create a new controller program. You will first be prompted to choose between a C, C++, Java, Python or MATLABTM controller. If you choose C or C++ on Windows, Webots will offer you the possibility to create a Makefile / gcc project or a Visual Studio project. Then, Webots will ask you to enter the name of your controller and finally it will create all the necessary files (including a template source code file) in your current project directory.
- **New Physics Plugin** - Let you create a new physics plugin for your project. Webots asks you to choose a programming language (C or C++) and a name for the new physics plugin. Then it creates a directory, a template source code file and a Makefile in your current project.

lets create new project Go to `Options menu` –> `Wizards` –> `New Project Directory` and follow the instructions:

1. Name the project directory `bug` instead of the proposed `my_project`.
2. Name the world file `bug.wbt` instead of the proposed `empty.wbt`.
3. Click all the tick boxes, including the "Add a rectangle arena" which is not ticked by default.

Webots GUI is composed of four principal windows: the **3D window** that displays and allows you to interact with the 3D simulation, the **Scene tree** which is a hierarchical representation of the current world, the **Text editor** that allows you to edit source code, and finally, the **Console** that displays both compilation and controller outputs.

Step 4 Add Robot (E-puck)

Before we add robot lets Change Areana Size from 1x1m to 2x2 go to `RectangleArena` –> `floorSize` and change the value of x and y to 2

Now to add a robot click on the `Add` button at the top of the scene tree view. In the dialog box, choose `PROTO nodes (Webots Projects) / robots / gctronic / e-puck / E-puck (Robot)`. An e-puck robot should appear in the middle of the arena. Double-click on it in the scene tree to open its fields.

Don't forget to save your changes before you run your simulation otherwise will have to do the same process again.

Notice it has `translation` and `rotation` fields, which changes if you change the robot position. also it has default Controller `e-pick_avoid_obstacles` so if you run the simulation robot able to move and avoid obstacles.

Part 2: Create controller

Lets create simple controller to move robot forward.

Go to Options menu –> Wizards –> New Robot Controller select your language (python) and name the controller move_forward. it will open .py file in **Text Editor** with default structure.

move_forward.py Code

```python
"""move_forward controller."""

# You may need to import some classes of the controller module. Ex:
#  from controller import Robot, Motor, DistanceSensor
from controller import Robot

robot = Robot()

time_step = 32
max_speed = 6.24


# lets create motor instances
left_motor = robot.getMotor('left wheel motor')
right_motor = robot.getMotor('right wheel motor')

# set motor position to inf and velocity to 0
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))
left_motor.setVelocity(0.0)
right_motor.setVelocity(0.0)



while robot.step(time_step) != -1:

    left_speed = max_speed *0.5
    right_speed = max_speed *0.25


    left_motor.setVelocity(left_speed)
    right_motor.setVelocity(right_speed)
```

To change the dafault controller of e-puck go to e-puck –> controller and select a controller from the list, you will able to see your controller as move_forward in there. Run the simulation to see the behaviour of your controller.

Try to move the robot in circular path by changing the controller code.

Part 3: Line following

For line following we need IR sensors, to add IR sensors ro robot go to `e-puck` –> `groundSensorsSlot` and click add icon, from `base node` select `DistanceSensors` and add. a distance sensor will be added to your epuck exactly at the front center but we need to make it offset by -1cm in y axis for our use, you can do so by going to `distance sensors`–> `translation` and change y to -0.01m.

Since we need IR sensor we will have to change sensor type from `distance sensors`–> `type` and select `infra-red`.

whenever we create a distance sensors it will asign its name as `diatance sensor` but lets change it to `ir0` so it will easy to call it in controller from `distance sensors`–> `name`.

Repeat the same process to add second ground sensors but this time make it 2cm offset in y direction.

Bug0

```python
"""bug0 controller."""

# You may need to import some classes of the controller module. Ex:
#  from controller import Robot, Motor, DistanceSensor
from controller import Robot

my_robot = Robot()

# def run_robot(robot):

time_step = 32
max_speed = 6.24


#motors
left_motor = robot.getMotor('left wheel motor')
right_motor = robot.getMotor('right wheel motor')
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))
left_motor.setVelocity(0.0)
right_motor.setVelocity(0.0)
```

```
left_ir = robot.getDistanceSensor('ir1')   # get the distance sensors from the robot
left_ir.enable(time_step)   # enable them with the time stamp
right_ir = robot.getDistanceSensor('ir0')
right_ir.enable(time_step)

while robot.step(time_step) != -1:

    left_ir_value = left_ir.getValue()
    right_ir_value = right_ir.getValue()

    print("left: {} rigt: {}".format(left_ir_value, right_ir_value))

    left_speed = max_speed *0.25
    right_speed = max_speed *0.25

    if (left_ir_value > right_ir_value) and (6 < left_ir_value < 15):
        print("go left")
        left_speed = -max_speed *0.25
    elif (right_ir_value > left_ir_value) and (6 < right_ir_value < 15):
        print("go right")
        right_speed = -max_speed *0.25

    left_motor.setVelocity(left_speed)
    right_motor.setVelocity(right_speed)
```