Exercise 1: Introduction to Webots

Duration: 1 weeks

Requirements: Access to a license of Webots EDU, Version 6.2.4 or better, e.g. on CSEL

Goals: Understanding the user interface and 3D CAD functionalities of Webots, writing a controller in Java, understanding 3D rendering and the camera node, and understanding the physics engine integrated in Webots.

This exercise is taken from the Webots manual, Chapter 7, pages 167-183, © Olivier Michel, Cyberbotics Ltd., and is reproduced here for your convenience with minimal changes that reflect the scope and setup of this class at CU Boulder.

Partial solutions for this exercise, i.e. world files containing a finished robot are included in the archive lab1.tar.gz that you find on the course website.

Deliverables: Demonstration of a working robot and controller

0. Setting up your user account
A local copy of Webots is installed in /root/usr/local/webots. As this directory is write-protected, you will need to create your own Webots directory.

1. Create a webots directory somewhere in your home directory, e.g. by typing

   cd ~
   mkdir webots

2. Tell Webots where it can find local files by setting the WEBOTS_HOME environment variable

   export WEBOTS_HOME=~/webots

   you can make this permanent by adding this command to your bashrc file (If you use bash as follows

   echo export WEBOTS_HOME=~/webots >> ~/.bashrc

   Department of Computer Science, University of Colorado at Boulder
1.0 Guided tour

Webots is a “physics-based” simulator, that faithfully simulates dynamical systems using first principles and the probabilistic natures of sensors and actuators using random number generators. On the CSEL computers, Webots should start simply after typing `webots` in a terminal window. Choose “Guided Tour” on the welcome screen and observe how Webots is able to simulate different mechanisms and sensors.

1.1 My first world: mybot.wbt

As a first introduction, we are going to simulate a very simple robot made up of a cylinder, two wheels and two infra-red sensors (see figure 1.1). The robot will be controlled by a program performing obstacle avoidance and move in a simple environment which contains some obstacles to avoid, surrounded by a wall.

![Figure 1.1: The MyBot robot](image)

1.1.1 Setup files

After starting a new project, you must set up a working directory that will contain the files your will create in this tutorial. To do so, create a project called `my_webots` somewhere in your local `webots` directory by using the corresponding wizard (by selecting the **Wizard | New Project Directory...** menu item). This operation creates a directory called `my_webots` and notably four subdirectories called `worlds`, `controllers`, `protos`, and `plugins`. The first one will contain the simulation worlds you create, the second one will contain your programs for controlling the simulated robots. Don’t worry about `protos` and `plugins` for now.

1.1.2 Environment

This first simulated world is as simple as possible. It includes a floor, 4 obstacles and a surrounding wall to prevent the robot from escaping. This wall is modeled using an Extrusion node. The coordinates of the wall are shown in [figure 1.2](image).
First, launch Webots and stop the current running simulation by pressing the Stop button. Go to the File menu, New World item to create a new world. This can also by achieved through the New button of the 3D windows, or the keyboard shortcut indicated in the File menu. Then open the scene tree window from the Scene Tree... item in the Tools menu. This can also be achieved by double-clicking in the 3D world. Let us start by changing the lighting of the scene:

1. Select the PointLight node, and click on the + just in front of it. You can now see the different fields of the PointLight node. Select ambientIntensity and enter 0.6 as a value, then select intensity and enter 0.3. Finally, select location and enter [0.75 0.5 0.5] as values.
2. Select the PointLight node, copy and paste it by using the corresponding buttons at the top of the scene tree. Open this new PointLight node and enter [-0.5 0.5 0.35] in the location field.
3. Repeat this paste operation twice more with [-0.45 0.5 -0.5] in the location field of the third PointLight node, and [-0.49 0.49] in the location field of the fourth and last PointLight node.
4. The scene is now better lit. Click on View -> Optional Rendering -> Show Light Positions and check that the light sources are now visible in the scene. Try the different mouse buttons, including the mouse wheel if any, and drag the mouse in the scene to navigate and observe the locations of the light sources. If you need further explanation about 3D navigation in the world, go to the Help menu and select the How do I navigate in 3D? item.

Secondly, let us create the wall:
1. Select the last Solid node in the scene tree window (which is the floor) and click on Add New -> New node button. Depending on your Webots version, the Add New button might also be called Insert after.
2. Choose a Solid node.
3. Open this newly created Solid node from the + sign and type "wall" in its name field.
4. Select the children field and Add New a Shape node.
5. Open this Shape, select its appearance field and create an Appearance node from the New node button. Use the same technique to create a Material node in the material field of the Appearance node. Select the diffuseColor field of the Material node and choose a color to define the color of the wall. Let us make it light brown. In order to make your object change its color depending on its illumination, select the specularColor field of the Material node and choose a color to define the color of the illuminated wall. Let us use an even lighter brown to reflect the effect of the light.
6. Similarly, it also is possible to modify the colors of the ground. To do so you will have to modify the two color fields of the last Transform node (the one corresponding to the ground), which are located in the DEF FLOOR Solid node. child / Shape / geometry ElevationGrid/ Color color node. In our examples we have changed it to a black and white grid.
7. Now create an Extrusion node in the geometry field of the Shape.
8. Set the convex field to FALSE. Then, set the wall corner coordinates in the crossSection field as shown in figure 1.2. You will have to re-enter the first point (0) at the last position (10) to complete the last face of the extrusion.
9. In the spine field, write that the wall ranges between 0 and 0.1 along the Y axis (instead of the 0 and 1 default values).
10. As we want to prevent our robot from passing through the walls, we have to define the boundingObject field of the wall. Bounding objects cannot use complex geometry objects. They are limited to box, cylinder, sphere and indexed facetset primitives. Hence, we will have to create four boxes (representing the four walls) to define the bounding object of the surrounding wall. Select the boundingObject field of the wall and create a Group node that will contain the four walls. In this Group, insert a Transform node in the children field. Add a Shape to the unique children of the Transform. Create a Material in the node Appearance and set its diffuseColor and specularColor to white. This will be useful later when the robot will need to detect obstacles, because sensor detection is based on color. Now create a Box as a geometry for this Shape node. Set the size of the Box to [ 0.01 0.1 1 ], so that it matches the size of a wall. Set the translation field of the Transform node to [ 0.495 0.05 0 ], so that it matches the position of the first wall.
11. You will now need to create three copies of the Box bounding object so that the entire wall is enclosed, but we will skip this step here.
12. Close the tree editor, save your file as "my_mybot.wbt" and look at the result.

The wall in the tree editor and its result in the world editor are visible in figure 1.3.

Figure 1.3: The wall in the tree editor and in the world editor

Now, let us create the obstacles:

1. Select the last Solid node in the scene tree window (which is the wall) and click on the insert after button.
2. Choose a Solid node.
3. Open this newly created Solid node from the + sign and type "green box" in its name field.
4. Using the same technique as for the wall, add first a Shape, then an Appearance and a Material. For the color, let us make it green with a lighter green for the illuminated parts.
5. Now create a Box node in the geometry field of the Shape and set its size to \([0.23 \ 0.1 \ 0.1]\). Set the DEF name of this geometry to BOX0.
6. To create the boundingObject of this object, create a Shape node and reuse the previous DEF for the geometry. Also, create an Appearance and a Material node and set the two colors to white, like we did for the wall.
7. Finally set the translation field to \([-0.05 \ 0.05 \ -0.25]\), but leave its rotation field at the default value.
8. You can now repeat the above steps to place arbitrary additional obstacles. We will skip this step here. Instead, open the file my_bot1.wbt and continue from there.

Review
- Use the Add New icon to create a Solid object
- Under the solid’s children tab add a new Shape
- Inside the shape node, add a Appearance node and inside that a Material
- Change the color by changing the diffuseColor for the shape’s color and then the specularColor for the illuminated color – click on the color square to change the color
- Under the Geometry section of the Shape field, add a new Box node and set the size
- To make an object “solid” it needs a bounding object. For this, add a new Shape node with appearance and material as above setting the color to white as a child node. Copy the geometry of the Solid and paste it into the geometry of the boundingObject.
- Finally, set the translation and rotation fields to place the object where you want it.

1.1.3 Robot
This subsection describes how to model the MyBot robot as a DifferentialWheels node containing several children: a Transform node for the body, two Solid nodes for the wheels, two DistanceSensor nodes for the infra-red sensors and a Shape node with a texture for the "face".

The origin and the axis of the coordinate system of the robot and its dimensions are shown in figure 1.4.
To model the body of the robot:

1. Open the scene tree window.
2. Select the last Solid node.
3. **Insert after** a DifferentialWheels node, set its name to "mybot".
4. In the children field, first introduce a Transform node that will contain a shape with a cylinder. In the new children field, **Insert after** a Shape node. Choose a color, as described previously. In the geometry field, **insert** a Cylinder node. Set the height field of the cylinder to 0.08 and the radius one to 0.045. Set the DEF name of the geometry to BODY, so that we will be able to reuse it later. Now set the translation values to [0 0.0415 0] in the Transform node (see **figure 1.5**).

![Figure 1.4: Coordinate system and dimensions of the MyBot robot](image)

To model the left wheel of the robot:

1. Select the Transform node corresponding to the body of the robot and **Insert after** a Solid node which will model the left wheel. Type "left wheel" in the name field, so that this Solid node is recognized as the left wheel of the robot and will rotate according to the motor command.
2. The axis of rotation of the wheel is z. The wheel will be made of a Cylinder rotated by $\pi/2$ radians around the z axis. To obtain proper movement of the wheel, you must be careful not to confuse these two rotations. Consequently, you must add a Transform node to the children of the Solid node.
3. After adding this Transform node, introduce inside it a Shape with a Cylinder in its geometry field. Don't forget to set an appearance as explained previously. The dimensions of the cylinder should be 0.01 for the height and 0.025 for the radius. Set the rotation to [0 0 1 1.57]. Pay
attention to the sign of the rotation; if it is wrong, the wheel will turn in the wrong direction.

4. In the Solid node, set the translation to [-0.045 0.025 0] to position the left wheel, and set the rotation of the wheel around the x axis: [1 0 0 0].

5. Give a DEF name to your Transform: WHEEL; notice that you defined the wheel translation at the level of the Solid node, so that you can reuse the WHEEL Transform for the right wheel.

To model the right wheel of the robot:

1. Select the left wheel Solid node and insert after another Solid node. Type "right wheel" in the name field. Set the translation to [0.045 0.025 0] and the rotation to [1 0 0 0].

2. In the children, insert after USE WHEEL.

3. Close the tree window and save the file. You can examine your robot in the world editor, move it and zoom in on it.

The robot and its two wheels are shown in figure 1.6.

The two infra-red sensors are defined as two cylinders on the front of the robot body. Their diameter is 0.016 m and their height is 0.004 m. You must position these sensors properly so that the sensor rays point in the right direction, toward the front of the robot.
1. In the children of the DifferentialWheels node, insert after a DistanceSensor node.
2. Type the name "ir0". This name will be used by the controller program.
3. Attach a cylinder shape to this sensor: In the children list of the DistanceSensor node, insert after a Transform node. Give a DEF name to it: INFRARED, which you will reuse for the second IR sensor.
4. In the children of the Transform node, insert after a Shape node. Define an appearance and insert a Cylinder in the geometry field. Type 0.004 for the height and 0.008 for the radius.
5. Set the rotation for the Transform node to [0 0 1 1.57] to adjust the orientation of the cylinder.
6. In the DistanceSensor node, set the translation to position the sensor and its ray: [-0.02 0.063 -0.042]. In the Preferences dialog, in the Rendering tab, check the Display sensor rays box. In order to have the ray directed toward the front of the robot, you must set the rotation to [0 1 0 2.07].
7. In the DistanceSensor node, you must enter some distance measurement values for the sensors in the lookupTable field, according to figure 1.7. These values are:

```
lookupTable [ 0 1024 0,
              0.05 1024 0,
              0.15 0 0 ]
```

![Graph](image)

Figure 1.7: Distance measurements of the MyBot sensors.

8. To model the second IR sensor, select the DistanceSensor node and insert after a new DistanceSensor node. Type "ir1" as a name. Set its
translation to $[0.02 \ 0.063 \ -0.042]$ and its rotation to $[0 \ 1 \ 0 \ 1.07]$. In the children, insert after USE INFRARED. In the lookupTable field, type the same values as shown above.

9. In order to better detect the obstacles, we will use two rays per DistanceSensor. To do so, open both DistanceSensor nodes and set the value of the numberOfRay field to 2 and the aperture field of each to 1.

The robot and its two sensors are shown in figure 1.8.

![Figure 1.8: The DistanceSensor nodes of the MyBot robot](image)

We will now add some textures to the robot. The following steps have already been implemented in the world `my_bot2.wbt` in order to save you the work of entering the values manully.

**Note:** A texture can only be mapped on an IndexedFaceSet shape. The texCoord and texCoordIndex entries must be filled. The image used as a texture must be a `.png` or a `.jpg` file, and its size must be $(2^n) \times (2^n)$ pixels (for example $8 \times 8$, $16 \times 16$, $32 \times 32$, $64 \times 64$, $128 \times 128$ or $256 \times 256$ pixels). Transparent images are allowed in Webots. Moreover, PNG images should use either the 24 or 32 bit per pixel mode (lower bpp or gray levels are not supported). Beware of limits on texture images imposed by your 3D graphics board: some old 3D graphics boards are limited to $256 \times 256$ texture images, while more powerful ones will accept $2048 \times 2048$ texture images.

To paste a texture on the face of the robot:
1. Select the last DistanceSensor node and **Insert after** a Shape node.
2. Create an Appearance node in the appearance field. Create an ImageTexture node in the texture field of this node. In the ImageTexture node select the url field and hit **Insert after** button. Fill the empty url string ("") with textures/mybot.png. This URL path is relative to the worlds directory.
3. In the geometry field, create an IndexedFaceSet node, with a Coordinate node in the coord field. In the Coordinate node select the point field and hit the **Insert after** button. Then enter the coordinates of the points in the point field (you need to hit **Insert after** for each line):

   ```
   [ 0.015 0.038 -0.041 ,
     0.015 0.023 -0.041 ,
     0 0.023 -0.0455 ,
     -0.015 0.023 -0.041 ,
     -0.015 0.038 -0.041 ,
     0 0.038 -0.0455 ]
   ```

4. and **Insert after** in the coordIndex field the following values: 0, 1, 2, 5, -1, 5, 2, 3, 4, -1. The -1 value is there to mark the end of a face. It is useful when defining several faces for the same IndexedFaceSet node.
5. In the texCoord field, create a TextrueCoordinate node. In the point field, enter the coordinates of the texture (you need to hit **Insert after** for each line):

   ```
   [ 0 0
     0.5 0
     1 0
     1 1
     0.5 1
     0 1 ]
   ```

6. and in the texCoordIndex field, type: 5, 0, 1, 4, -1, 4, 1, 2, 3, -1. This is the standard VRML97 way to explain how the texture should be mapped to the object.
7. In our example, we have also modified the value of the creaseAngle of the IndexedFaceSet. This field modifies the way the transition of illumination between the different faces of the IndexedFaceSet is done. In our example, we have set its value to 0.9 so that the illumination transition is smooth between the two faces.
8. The texture values are shown in figure 1.9.
Figure 1.9: Defining the texture of the MyBot robot

To finish with the DifferentialWheels node, you must fill in a few more fields:

1. In the controller field, select "MybotSimple," which should appear in the popup controller list when you press the file selection button. It is used to determine which controller program controls the robot. Notice that the controllers that show up in the list are actually derived from the name of the directory the controller lives in. Directory and controller name have to be consistent.

2. The boundingObject field can contain a Transform node with a Cylinder, as a cylinder as bounding object for collision detection is sufficient to approximately bound this robot. Create a Transform node in the boundingObject field, with the translation set to [0 0.0415 0]. Reuse the BODY node defined previously, and add it to the children of the transform.

3. In the axleLength field, enter the length of the axle between the two wheels: 0.09 (according to figure 1.4).

4. In the wheelRadius field, enter the radius of the wheels: 0.025.
5. Values for other fields and the finished robot in its world are shown in figure 1.10.

![Figure 1.10: The other fields of the DifferentialWheels node](image)

The `mybot.wbt` file is included in the Webots distribution, in the `worlds` directory.

### 1.1.4 Principles of collision detection

The collision detection engine is able to detect every collision between any two *solid* nodes. By *solid* node, it is meant the Solid node itself and all the derived nodes, and this includes DifferentialWheels, Robot, Servo nodes, etc. The purpose of the collision detection is to prevent any two bounding objects from inter-penetrating. This is achieved by generating contact forces that will push the solids apart whenever necessary.

The collision detection engine calculates the intersection between the *bounding objects* of these solid nodes. The bounding object is specified in `boundingObject` field of these nodes. A bounding object is composed of a geometric shape or a group of geometric shapes that define the bounds of the solid. If the `boundingObject` field is `NULL`, then no collision detection will be performed on this particular solid and therefore nothing will prevent it from passing through other solids.

A solid node may contain other solid nodes in its list of *children*, each of them having its own bounding object. Therefore it is possible to define complex solid hierarchies, that can be used for example to model an articulated Robot.
1.1.5 A simple controller

This first controller is very simple, and therefore appropriately named `MybotSimple`. The controller program simply reads the sensor values and sets the two motors’ speeds in such a way that `MyBot` avoids the obstacles.

Below is the source code for the `MybotSimple.java` controller:

```java
import com.cyberbotics.webots.controller.DifferentialWheels;
import com.cyberbotics.webots.controller.DistanceSensor;

public class MybotSimple extends DifferentialWheels {
    // determines the speed of the bot (in units of the bot's motors)
    public static final int SPEED = 60;
    // determines simulation step (you get one new reading every step)
    public static final int TIME_STEP = 64;

    protected DistanceSensor ds0;
    protected DistanceSensor ds1;

    /**
     * initialization done in constructor
     */
    public MybotSimple() {
        // get your distance sensors by name
        ds0 = getDistanceSensor("ir0");
        ds1 = getDistanceSensor("ir1");

        // enable both of the distance sensors
        ds0.enable(TIME_STEP);
        ds1.enable(TIME_STEP);
    }

    /**
     * the run() method controls the actual robot behavior
     */
    public void run() {
        while (true) {
            // get the values of the sensors
            double ds0Value = ds0.getValue();
            double ds1Value = ds1.getValue();

            double leftSpeed, rightSpeed;
            if (ds1Value > 500) {
                /*
                 * If both distance sensors are detecting something,
                */
            }
        }
    }
}
```
* this means that we are facing a wall. In this case
* we need to move backwards.
*/
if (ds0Value > 500) {
    leftSpeed = -SPEED;
    rightSpeed = -SPEED / 2;
} else {
    /*
    * We turn proportionally to the sensors value
    * because the closer we are from the wall, the
    * more we need to turn.
    */
    leftSpeed = -ds1Value / 10;
    rightSpeed = (ds0Value / 10) + 5;
}
else if (ds0Value > 500) {
    leftSpeed = (ds1Value / 10) + 5;
    rightSpeed = -ds0Value / 10;
} else {
    /*
    * If nothing has been detected, we can move forward
    * at maximal speed.
    */
    leftSpeed = SPEED;
    rightSpeed = SPEED;
}

// adjust the speed
setSpeed(leftSpeed, rightSpeed);

/**
 * perform a simulation step, leave the loop when
 * the controller has been killed
 */
if (step(TIME_STEP) == -1) {
    break;
}
}
```
public static void main() {
    MybotSimple mybot = new MybotSimple();

    mybot.run();
}
```

This controller is found in the MybotSimple subdirectory of the controllers directory.

### 1.2 Adding a camera to the MyBot robot

This section can be considered as an exercise to check if you understood the principles for adding devices to a robot. The camera to be modeled is a color 2D camera, with an image 80 pixels wide by 60 pixels high, and a field of view of 60 degrees (1.047 radians).

We can model the camera shape as a cylinder, on the top of the MyBot robot at the front. The dimensions of the cylinder are 0.01 for the radius and 0.03 for the height. See [figure 1.11](#).

![Figure 1.11: The MyBot robot with a camera](image)

Try modeling this camera on your own. The my_bot3.wbt file is included in the worlds directory of this lab, in case you need help.

Once you have modeled the camera, make sure you fill in camera into the name field of the node. Every node in Webots needs a unique name and allows your
controller to link a class to a simulated device via the `GetDeviceName(string)` routine.

A controller program for this robot, named `mybot_camera`, is also included in the `mybot` project, in the `controllers` directory of your webots installation (`/usr/local/webots`). This camera program does not perform image processing, since it is just a demonstration program, but you could easily extend it to perform actual image processing – and we will do this in subsequent exercises. The robot could, for example, learn to recognize the different objects of the scene and move towards or away from them depending on whether the object is categorized as "good" or "bad."

### 1.3 Adding physics to the MyBot simulation

#### 1.3.1 Overview

The model we have defined for the *MyBot* robot does not include any physics modeling, as we did not specify any mass, etc. Instead it is a simple kinematic model, which can be used nonetheless for many mobile robotics simulation experiments where inertia and friction can be neglected. For example, it is well suited to simulate light desktop robots. Additionally, simulations run faster without physics.

However, as soon as things get more complex, you will need to introduce some physics to your model. For example, if your robot is heavy, you cannot afford to neglect inertia effects on its trajectory. If you want to add moveable objects, like boxes or a ball, physics simulation becomes necessary. Finally, if you want to model a robot architecture that is significantly different from the plain differential wheels model, like an omni-directional robot, a legged robot, a swimming robot or a flying robot, then you need to setup many physics parameters.

This section introduces a simple physics simulation to the *MyBot* world allowing the robot to play with a ball. More complex physics simulations can be implemented with Webots, involving for example different locomotion schemes based on the `Robot` and `Servo` nodes, allowing you to build complex wheeled and legged robots. Other possibilities include flying and swimming robots where hydrodynamics models are needed. These features will not be addressed in this tutorial, however.

#### 1.3.2 Preparing the floor for a physics simulation

Select the floor node, which should be the first `Transform` node in the scene tree, just after the `PointLight` nodes. Turn that `Transform` into a `Solid` node using the `Transform` button.

Now it is possible to define a `boundingObject` for the floor. Create a `Transform` node containing a `Box` as the bounding object. Set the `size` field for the `Box` to `[1 0.02 1]` and the `translation` field of the `Transform` to `[0.05 -0.01 0.05]`. The bounding object we just defined will prevent the robot from falling down through the floor due to gravity.
1.3.3 Adding physics to the MyBot robot

The MyBot robot already has a bounding object defined. However, since it will be moving, it also needs physics parameters that will be defined in its physics field as a Physics node. Create such a node and, as it is recommended to use the mass instead of the density, set its density to -1 and its mass to 0.5. The density is expressed in kilograms per cubic meter, and the mass in kilograms. The mass is ignored when the density is specified.

The wheels of the robot also need some physics properties to define the friction with the floor. But first they need a bounding object. Set the defined WHEEL node as the boundingObject for each wheel Solid. Then, add a Physics node to the first wheel, and enter WHEEL_PHYSICS as a DEF name. Finally, set the density to -1, the mass to 0.05, the coulombFriction to 1 and the forceDependantSlip to 0. Use this WHEEL_PHYSICS definition to define the physics of the second wheel.

We are now done! Save the world as my_mybot_physics.wbt, reload it using the revert button and run the simulation. You will observe that the robot is not moving very steadily (especially if you look at what the robot’s camera sees). That’s physics! Of course you can improve the stability of the movement by adjusting the bounding object of the robot, the speed of the wheels, the friction parameters, etc.

1.3.4 Adding a ball to the MyBot world

Now let us offer a toy to our robot. Instead of creating a ball object from scratch, let's borrow it from another world where such an object already exists. Open the supervisor.wbt world (/usr/local/webots/projects/samples/howto/worlds). Double-click on the soccer ball. This should open the scene tree window and select the BALL solid. Simply copy it using the Copy button, then re-open your mybot_physics.wbt world. Open the scene tree window, select the last object of the scene tree and click on Paste after. Can you see the soccer ball? Read How do I move an object? from the Help menu and place the ball in front of the robot. Save the world and run the simulation. The MyBot robot should be able to kick the ball, making it roll and bounce off the walls.