Lab 2: Physics Simulation
Webots Simulation

ECE183DA/MAE162D

Version 1/11/2023

1 Lab Overview

You **in pair** implement a Webots simulation setup for the Woodbot. You will use the code implemented in the previous lab and merge new codes from the main repository to your forked repository to update. You will conduct design decisions on which state estimators you would choose for the final algorithm design.

2 Requirements

You present TAs your results to get checked-off for the next lab.

- Saved outputs: Graph, video recording, and a corresponding output file for at least one illustrative run

2.1 Submission

Push your changes so that the codes can be verified.

3 Methods/Tools

Each student will be provided the followings:

- A Webots world file with woodbot imported (but no setup)
- Component positions and sensor specifications (in this documents)
- Codes that are necessary to run your woodbot with our system

Additionally, you need your implemented python codes from the previous lab. You should merge the updates from the main repository.

To merge from our main branch, you can add remote branch, and then fetch and merge the difference.

```
git remote add upstream https://git.capstone.uclalemur.com/staff/woodbot
git fetch upstream
git merge upstream/main
```

If you get unrelated history error, you can force merging with

```
git merge upstream/main --allow-unrelated-histories
```

A Appendix: Webots Setup

A.1 Introduction to Webots

In this lab, we use a physics simulator, Webots, to evaluate our robot motions and sensor models. Webots is an open source and free software to simulate dynamics and motions of robots.

A.1.1 Download and Install

You can download and install Webots from here. It is available for Windows 64bit, Mac, and Ubuntu. With the default installation, Webots comes with sample simulations and you can look into them to see what Webots is capable of.

A.1.2 Programming Language Setups

Webots has a built-in basic IDE and We use Python. For other language details and procedures, please refer to their manual language-setup, but you don't need to do it in our case.

You need to setup Pycharm/Visual Studio appropriately by following the instruction here.

A.2 Woodbot Webots Setups

Setting up Webots for our Woodbot robot includes following steps:

- **[Optional, Apex B]**Exporting 3D models from Solidworks
- **[Optional, Apex B]** Importing them in Webots
- Configuring your robot boundary conditions and actuation
- Defining sensors
- Setting up a controller to drive our robot and to read sensor outputs

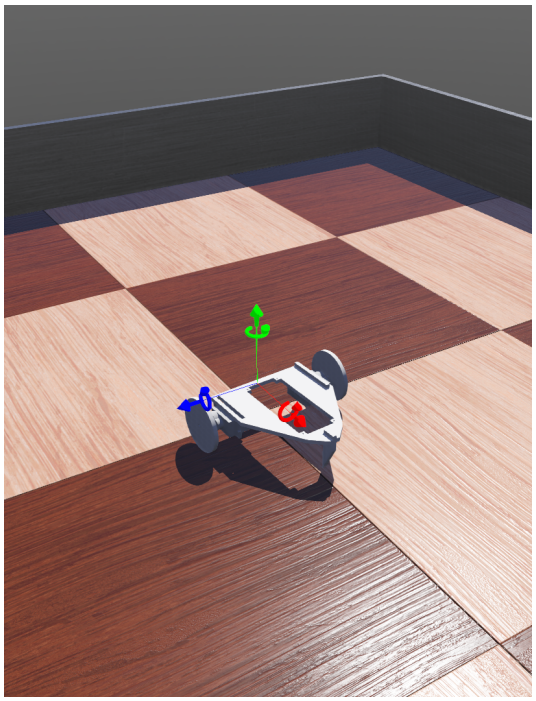

Figure 1: Webots Woodbot setup

A.2.1 Create a new Webots project and environment

We need to create a new project directory.

1. Go to "Wizards → New project directory..."
2. Pick a folder you want to place your Webots project in

Now we have an environment for our robot. In the Scene tree, you can add or modify your environment, robots, and objects.

Copy over Webots world file provided to your project directory and open the file from Webots. You should see the robot geometry is pre-imported and robot node is added.

A.2.2 Defining Your Robot

We need to define the robot and object boundaries so that Webots can compute contact with environments or objects. Our imported CAD models are Solid nodes, which only defines rendering (cosmetic) geometries at this moment.

1. Cut and paste all Woodbot Solid nodes under the Robot's children node one by one. A body Shape node is already included in the robot tree.
2. set Solid DEF name appropriately as you cut and paste (Servo, servo horn). Some of them are already defined.
3. Double click wheel Solid nodes physics node and add physics. Now your solid has mass and inertia based on its shape and density.
4. Set density $600 kg/m^3$
5. Check if **the boundingObject node** in the Robot node is set "USE Body" (Shape)
6. Set Robot node physics weight to 0.15 kg and center of mass to $[-.03, 0, 0]$ m

The boundingObject nodes define your object boundary conditions and they are used to compute contacts between objects. You should not have solids for your body as its boundary conditions are already defined in your robot node. Adding more boundingObjects will increase computation time.

Save your progress occasionally.

A.2.3 Defining Joints and Actuators

We need to define all joints you want to simulate in your robot. SolidWorks joint definitions are not inherited because your file was exported as meshed objects. Alternatively, you can use URDF which has all joint definitions, but that will require significant effort, too. We have 2 revolve joints: two actuated wheels. Necessary joint nodes are already added under the robot tree.

1. Expand one of the hingeJoints in Robot node children
2. Double click jointParameters and add a jointParameter node
3. Double click device and add a rotational motor node
4. Set axis to $[0, 0, -1]$
5. Set anchor position to $[-0.01425, -0.0063, 0]$m under "jointParameters HingeJointParameters → anchor"
6. Cut and paste your corresponding wheel solid into the endPoint node under the hinge joint node
7. Expand the rotationalMotor node and change its name to motor_r or motor_l. Be sure to change name, not only DEF
8. Repeat the same procedure for the other wheel

With motor names we can control motors from our code.

Now, ***Save your world*** before you proceed.

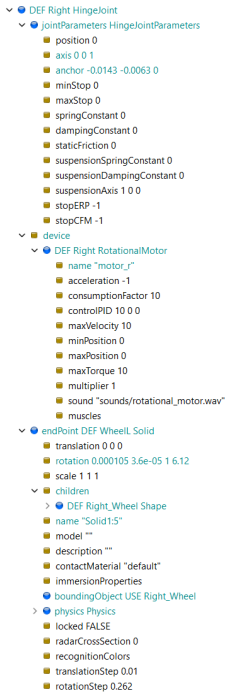

Figure 2: Scene tree after finishing "Defining Joints and Actuators". Your shape names may vary.

Actual [m]	Output	STD [%]
0	0	0.04
1.2	1200	0.04

A.2.4 Defining Sensors and Their Models

Here we set up sensors. They should already be added to your robot tree with a predefined name. There are five sensors: Gyro, Compass x, Compass y, and two Distance sensors. Gyro and compass have three axes, but only necessary axes are enabled.

1. Expand Lidar_l node and change type to "laser"
2. Set the lidar_f translation to $[-0.006, 0, 0]$ m and the lidar_l translation to $[-0.05, 0.0, -0.0]$ m, Compass and gyro to $[-0.015, 0, .015]$.. Note you can see the coordinate system by clicking the body. (Red: x, Green: y, Blue: z). The origin in Webots is the back of Woodbot and the origin in constant.py is at the center of wheel. (offset by the wheel radius)
3. set the lidar_r rotation to face it right by changing the rotation to $[0, 1, 0, -1.57]$. Webots uses unit vector and angle in radian.
4. Open lookup table node in lidar. This will define the relationship between the distance and sensor outputs
5. Set lookup table as shown in Table A.2.4.

Lookup tables define the relationship between the distance and sensor outputs. X column represents a distance [meter], Y column represents a corresponding sensor output [unitless], and Z column represents error standard deviation [percent]. You can add more control points. The values, $[0, 0, 0.04]$ and $[0, 1200, 0.04]$, mean that this distance sensor returns 1200 at 1.2m, 600 at 0.6m, and 0 at 0m with 4% standard deviation noise.

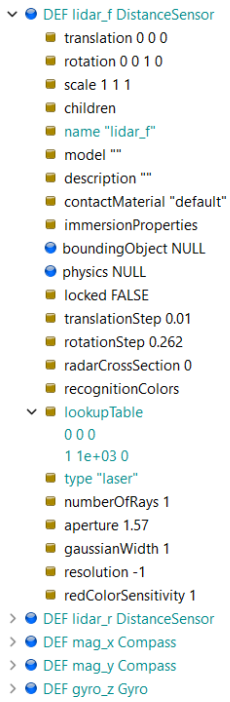

Figure 3: Scene tree with proper sensor definitions. sensor names maybe different depending on your formulation.py

We need to define those sensor positions. For distance sensors, we need to make sure that the lasers are pointing at walls, not the robot itself. Click "View → Optional Rendering → Show DistanceSensor Rays". This will visualize your sensor ray and its intersection with the walls. The lidar_F should point at a wall in front and lidar_R at a wall on the right. IMU and Gyro are at $[0, -0.02, 0]$.

You can apply a shape or a solid with a bounding object on sensors to visualize or to enable contacts, but for this lab those are unnecessary. ***Save your world*** before you proceed.

A.2.5 WorldInfo and Contact definition

Expand WorldInfo tree. Here you can define physics and change simulation settings. Change your simulation simulation time step (basicTimeStep) to 3ms, which will slow down your simulation, but improve simulation accuracy and stability. Later, you can increase time step to larger or reduce if you see some weird behaviors.

1. Double click contactProperties to add a contactProperties node, which defines frictions between objects
2. Change material 1 to floor and material 2 to wheel by typing it out in their properties
3. Set coulombFriction to 0.8 (rubber)
4. Go to two wheel solid nodes (recall that they are in your HingeJoints), then change contactMaterial to wheel, respectively from default (RectangleArena contactMaterial is already set to floor)
5. Create another contact properties for floor and tail and set coulombFriction to 0.01

A.2.6 Defining a Robot Controller

By now, we have setup robot boundary conditions, actuators, and sensors. The last thing to drive your robot is to setup your robot controller. In our case, all you have to do is set the controller under Robot node to $< extern >$. Make sure Supervisor is true in the Robot node, which is necessary to read ground truth postitions of the robot in our code.

Save your world before you proceed. Now you have done all you need to control your robot. Congrats!

A.3 Running Simulations

Before you try to run the simulation, click run or run as fast as possible. This does not move your robot yet, but it allows external controller to communicate with Webots. Add WebotsModel() with your choice of outputs, then run your main.py. You should see Woodbot starts running from the initial states.

If your formulation.py is used to as motors and sensors name in Webots. You can use provided formulation.py, or you can change motors and sensors name (NOT DEF!) in Webots.

If you encounter errors:

- Check if you properly setup your IDE with proper PATH
- Check if you have started your Webots simulation
- Check if your WebotsModel.py is properly merged
- Check if your formulation.py is properly merged (those are the sensor/actuator names)

B Export the CAD design for Webots (Not Required for this lab)

B.1 Create OBJ

Unfortunately, Solidworks cannot convert assembly to obj files. Alternatively, we can use Autodesk Inventor or Fusion 360 to open your assembly and save as obj. Make sure unit is meter in option.

B.2 From URDF files

An alternative method is to create URDF file then convert to a webots node using python urdf2webots. You can refer to how to generate urdf in Solidworks. Note that you cannot change the robot node in Webots tree if you do this way, but all joint axes info are carried from SolidWorks. If you have very complex robot, this could be a better way, but you manually have to modify urdf files to change the definition of bounding objects or to add sensors.

B.3 Simplify Bounding Objects

In physics simulation, it is common practice to use simplified bounding objects instead of the mesh form CAD. For instance, the body can be simplified as a box and the wheels as cylinder.

B.4 Importing Woodbot CAD Files

1. Click "File → Import 3D model" then follow the wizard to locate your file
2. Check use mesh as bounding object option and Click finish

Now be sure to ***Save your world***. You should save your world as you modify, otherwise all changes will be lost or you may be unable to undo your mistakes. Webots Undo does not keep more than one action, and it is not always available. You can reload your file with a refresh button next to the save button.