Topic	Ramp Follower Robot- IV	
Class Description	**Students will be introduced to Robots controller and how controllers integrate with Robots.**	
Class	**PRO C283**	
Class time	**45 mins**	
Goal	• Introduction to Controllers • Integration of Controller with Webots • Understanding of Robot Language	

Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Smartphone
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen	

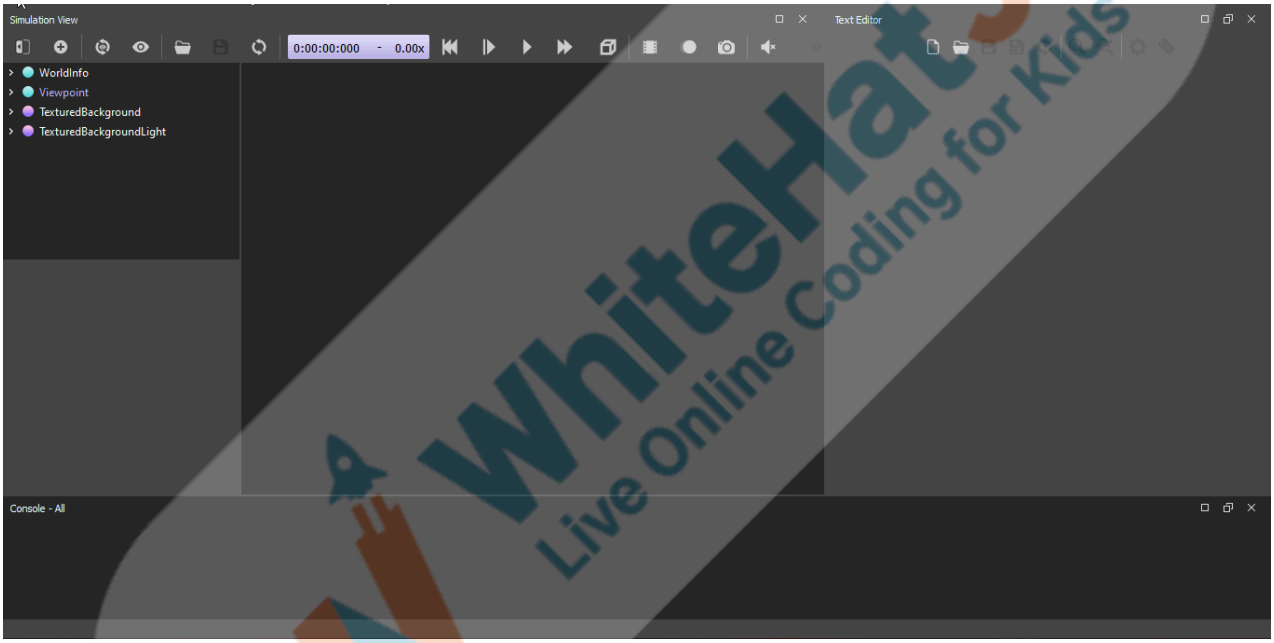
Class structure	**Warm-Up** **Teacher-Led Activity** **Student-Led Activity** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **05 mins**
Credit & Permissions:	**This project uses Webots, an open-source mobile robot simulation software developed by Cyberbotics Ltd.** **License**	
WARM-UP SESSION - 10 mins		
Teacher Action		**Student Action**

Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
• **Introduction to Robot Controller**	
Teacher Action	**Student Action**
Any doubts from the last class! *The teacher will clarify if there are any doubts!* So Let's deep dive into the Robots World! Robots can do anything, we just need to train them accordingly. You must know this as we have seen so many Robo-centric movies.	**ESR:** Varied!

Today we will learn about programming robot controllers. We will design a controller for our Ramp Follower which we did in the last class. We will learn the basics of robot programming in Webots. We should understand what is the link between the design and the controller, how the robot controller has to be initialized and cleaned up, how to initialize the robot devices, how to get the sensor values and program them to train and command the robot.	
Download the file from Teacher Activity 1	Student will download the file from Student Activity 1
Open the file using Webot. 1. Open the **webots** 2. Go to the **Open World** 3. Upload the previous webots file	
Let;s create a new Controller and integrate with the old one.	
Create a New Controller: We will now program a simple controller that will just make the robot move forwards. As there is no obstacle, the robot will move forwards forever. Firstly we will create and edit the Python controller, then we will link it to the robot. A controller is a program that defines the behavior of a robot. Create a new Python controller using the Wizards / New Robot Controller.	

1. Go to Wizards from the menu bar.
2. Select New Robot Controller
3. Name the Controller file and select the language as Python.
4. Click on Finish

Note: Controller name is very important, as this will be used in the Robot scene tree under the controller name.(Check the error section at the bottom if there comes any error while running the program)



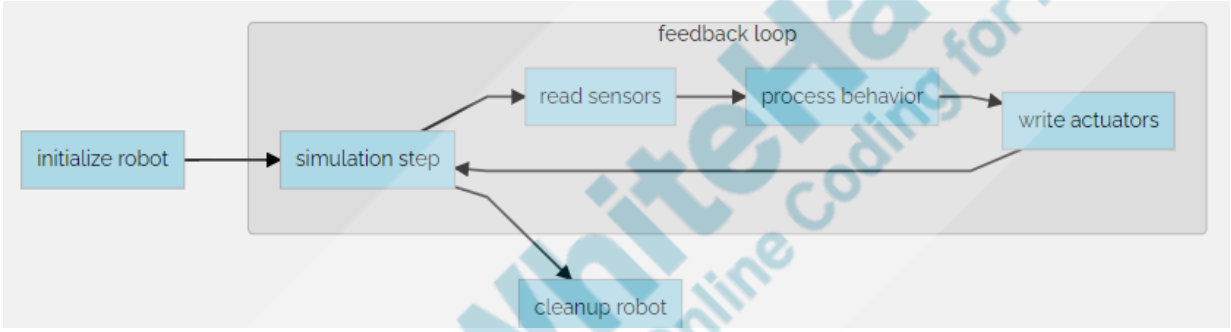
https://s3-whjr-curriculum-uploads.whjr.online/0424480f-a401-4052-9cc8-dbb7ce737a4c.gif

Default controller file would be like this:

Delete the entire program and start writing from scratch. Now let's understand the procedure of Robot Controller	
Program a controller We would like to program a very simple behavior. You will program the robot to go forwards until the end of the ramp is detected by the front distance sensors, and then to turn back towards the ascent side of the Ramp.In order to do that, we will use the simple feedback loop.	

Let's understand how this feedback loop works:

- Feedback Loop: A feedback loop is a process in which the outputs of a system are circled back and used as inputs.
- Our simulation will take the input from the sensors and motors which acts as an actuator on receiving the signal from the sensor will initialize the robot.



At the beginning of the controller file, we must import the libraries corresponding to the Robot.

For example to use distance sensor we will use sensor libraries, motors will use motor libraries.
As our Robot is a Ramp follower and we are using altimeter we have to use altimeter libraries.

Reference Code:

```
from controller import Robot
from controller import Motor
from controller import Altimeter
from controller import LED
import math
```

Main function:

1. The main function is where the controller program starts execution. We have created our own class **MyController** and created an object of it and used its run method where we will implement the code to start the execution.

Reference Code:

```
controller = MyController()
controller.run()
```

2. The Webots API has to be initialized using the **__init__** function

Reference Code:

```
super(MyController, self).__init__()
```

3. The basic time step is the time step increment used by Webots to advance the virtual time and perform simulation. This duration is specified in milliseconds.

Reference Code:

```
self.timeStep = 32  # set the control time step
```

We also define the basic structure of our class MyController as shown below:

Reference Code:

```
from controller import Robot
from controller import Motor
from controller import Altimeter
from controller import LED
import math


class MyController(Robot):
    def __init__(self):


    def run(self):



# main Python program
controller = MyController()
controller.run()
```

Next, we create and define the variables that will store information about devices used by our robot. Let's first see the code to work with an altimeter. **Reference Code:** self.altimeter=self.getDevice("altimeter") self.altimeter.enable(self.timeStep) 1. The **getDevice function of Robot** will look for the actual device name. 2. Each sensor must be enabled before it can be used. If a sensor is not enabled it returns undefined values. Enabling a sensor is achieved by using the corresponding *.enable function, where the star (*) stands for the sensor type.	
Teacher Stops Screen Share	
So now it's your turn. Please share your screen with me.	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
STUDENT-LED ACTIVITY - 20 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Full Screen.**	
Student Initiates Screen Share	
ACTIVITY • **Robot Ascent/Descent Direction**	
Teacher Action	**Student Action**

<table>
<tr><td>Teacher helps student to download boilerplate code

Open text file using option Open Text File under File menu.</td><td>Student downloads <u>student Activity 2</u></td></tr>
<tr><td>Now it's time to do wheel rotation .

1. The sensor value is updated during the call to the step() function. The call to the getDevice() function retrieves the latest value of sensors/motors.

2. Define the left_motor and right_motor using the getDevice() as shown below.
Reference Code:
self.left_motor = self.getDevice("left wheel motor")
self.right_motor = self.getDevice("right wheel motor")

3. To control a motion, it is generally useful to decompose that motion in discrete steps that correspond to the control step. As before, an infinite loop is used here: at each iteration a new target position is computed according to a sine equation.

4. The setPosition function stores a new position request for the corresponding rotational motor. Note that the setPosition function stores the new position, but it does not immediately actuate the motor.

5. The effective actuation starts on the next line, in the call to the step() function. The step() function sends the actuation command to the RotationalMotor but it does not wait for the RotationalMotor to complete the motion (i.e. reach the specified target position); it just simulates the motor's motion for the specified number of milliseconds.</td><td></td></tr>
<tr><td colspan="2">Reference Code:
self.left_motor = self.getDevice("left wheel motor")
self.right_motor = self.getDevice("right wheel motor")</td></tr>
</table>

self.left_motor.setPosition(math.inf) self.right_motor.setPosition(math.inf)	
This function boolean value corresponds to the direction_Switch field of the Robot node. This function can be used to determine rotation, whether it is true or false.	
Reference Code: self.direction_switch = False	
• Each controller process exchanges sensors and actuators data with the Webots process during the **step()** function calls. So for example, the **setPosition()** function does not immediately send the data to Webots. Instead it stores the data locally and the data is effectively sent when the **step()** function is called. • When the **step()** function returns, the motor has moved by a certain (linear or rotational) amount which depends on the target position, the duration of the control step, the velocity, acceleration, force, and other parameters specified in the ".wbt" description of the Motor. • For example, if a very small control step or a low motor velocity is specified, the motor will not have moved much when the **step()**function returns. In this case several control steps are required for the RotationalMotor to reach the target position. If a longer duration or a higher velocity is specified,	

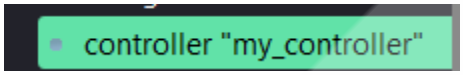
then the motor may have fully completed the motion when the **step()** function returns.

- Note that the **setVelocity()** function only specifies the desired target velocity
- If we want to control the velocity of several rotating Motors simultaneously, then you need to specify the desired velocity for each.
- RotationalMotor separately, using the **setVelocity()** function. Then you need to call the **step()** function one to actuate all the RotationalMotors simultaneously.

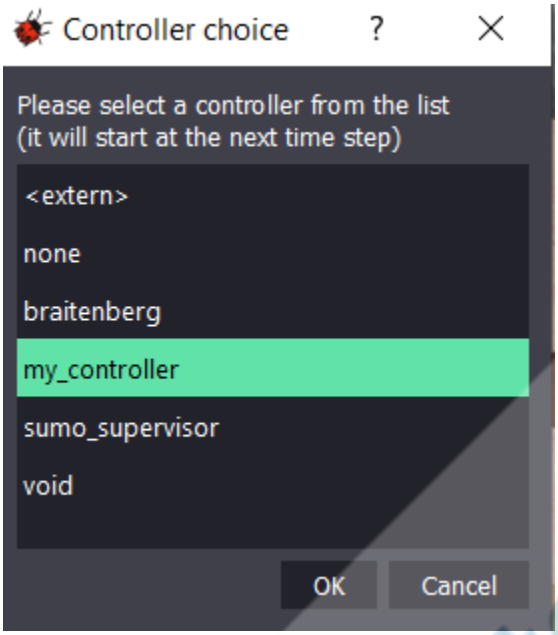
Reference Code:

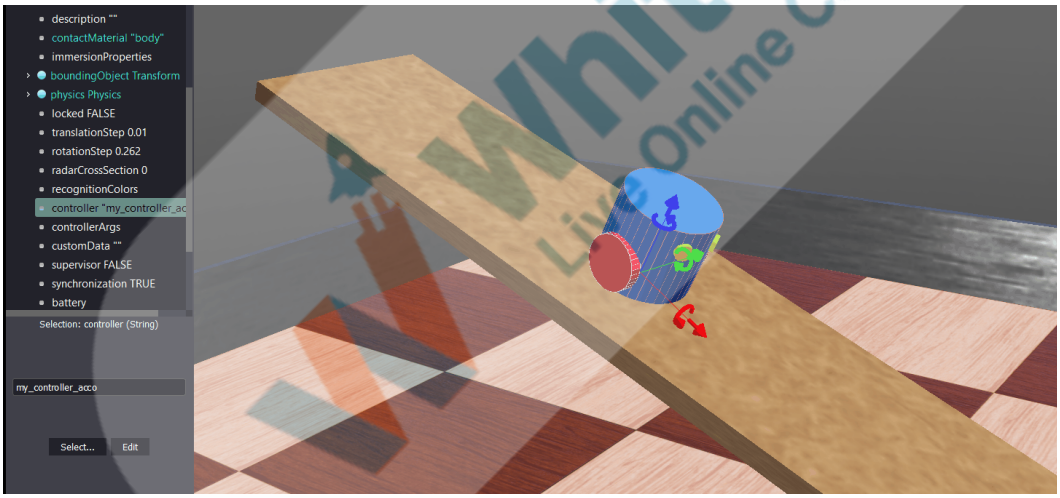
```
def run(self):

    while self.step(self.timeStep) != -1:
        # get the time step of the current world.

        altitude = self.altimeter.getValue()
        # print(altitude)
        if (not self.direction_switch):
            self.left_motor.setVelocity(2.0)
            self.right_motor.setVelocity(2.0)
            if (altitude <= 0.05):
                self.direction_switch = True
        else:
            self.left_motor.setVelocity(-2.0)
            self.right_motor.setVelocity(-2.0)
            if (altitude >= 0.25):
                self.direction_switch = False
```

Run the Program

Click on Save option  **(Save current file)** Try to save your controller and world file in the same location.	
Now we need to add controller name under scene tree too, ● Go to your Scene tree ● Click on Robot ● Check Controller name  ● Click on controller  ● Click on **Select** ● Select the name of your controller file which we mentioned while creating the controller. ● In this case the controller name is **my_controller.**	

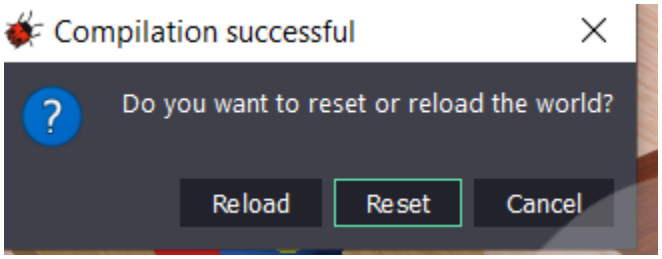
Note: Student can select their controller name

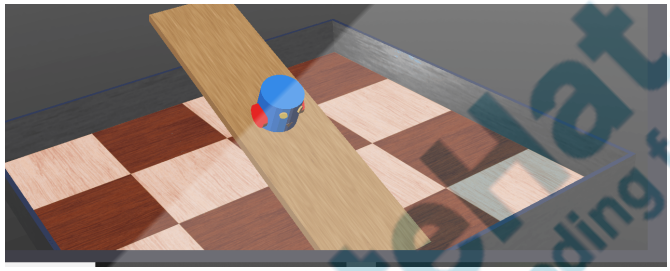




https://s3-whjr-curriculum-uploads.whjr.online/670292b9-fa0f-494a-9fda-6d458f9d8d25.gif

Note: Now if you run the simulation, the robot: maybe don't move at first .Please pause and reload the world.



Output will be look like this:	
 https://s3-whjr-curriculum-uploads.whjr.online/63ada1e8-797a-4697-afd3-df67adf96c5a.gif	
So basically every Robot needs a controller and we must write a program to carry out Robot Operations. So we're done with our full fledged Robot.	
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 05 mins	
Activity details **Following are the WRAP-UP session deliverables:** ● Appreciate the student. ● Revise the current class activities. ● Discuss the quizzes.	
WRAP-UP QUIZ	

Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** • Explain the facts and trivia • Next class challenge • Project for the day • Additional Activity (Optional)	
FEEDBACK • **Appreciate and compliment the student for trying to learn a difficult concept.** • **Get to know how they are feeling after the session.** • **Review and check their understanding.**	
Teacher Action	**Student Action**
You get “hats-off” for your excellent work! In the next class, we will learn about obstacle detection	*Make sure you have given at least 2 hats-off during the class for:* 
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks 	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Previous Class Code	https://github.com/procodingclass/PRO-C282-Reference-Code
Teacher Activity 2	Reference Code	https://github.com/procodingclass/PRO-C283-Ramp-Follower-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/a2991440-2b12-40fd-be7b-0b80ab7da96d.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C283-Project-Solution
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/c50c90ce-82e3-469f-9008-63e7689f0a91.pdf
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C282-Reference-Code
Student Activity 2	Controller file	https://github.com/procodingclass/Pro-C283_Student_BoilerPlateCode