Control of Mobile Robotics

CDA4621

Fall 2022

Lab 1

Basic Robot Programming

Total: 100 points

Due Date: 9-12-2022 by 11:59pm

The assignment is organized according to the following sections: (A) Requirements, (B) Objective, (C) Task Description, (D) Task Evaluation, and (E) Lab Submission.

A. Requirements

A.1 Physical Robot

Robot: "Robobulls-2018" **Programming**: Python
Basic Files: servos.py, encoders.py ("SamplePrograms.zip")

A.2 Robot Simulator

Simulator: Webots **Robot**: "e-puck". **Programming**: Python
Basic Files: "Lab1_epuck.zip"

B. Objective

This lab is an introduction to the robotic system. It will teach you about motion control and encoder feedback: (1) Motors, and (2) Encoders.

B.1 Physical Robot

"Robobulls-2018" has two wheels, each with approximate diameter 2.6", and approximate distance between wheels 4", as shown in Figure 1.

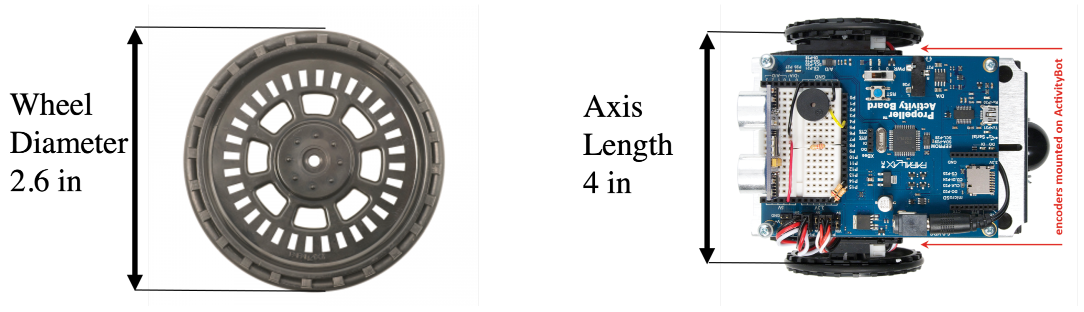

Figure 1. "Robobulls 2018" robot.

B.1.1 Motors

Robot motion is controlled by specifying individual motor (servo) velocities. Read the section "Communication Protocol" on the Parallax servo datasheet[1].
The robot has the following PWM method[2] to control motor velocities ("servos.py"):

[1] http://www.mantech.co.za/Datasheets/Products/900-00008-65S.pdf
[2] https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/library-reference

```
pwm.set_pwm(channel, on, off);
```

where

- channel: Motor channel value (0…15).
- on: Signal transition from low to high (0…4095).
- off: Signal transition from high to low (0...4095).

Make sure you understand and can answer the following questions:

- How often does it make sense to call the PWM function?
- What would happen in the following pseudo-code loop?

```
while true:
    pwm.set_pwm(LSERVO, 0, math.floor(1.6 / 20 * 4096));
    pwm.set_pwm(RSERVO, 0, math.floor(1.6 / 20 * 4096));
    time.sleep(0.5)
    pwm.set_pwm(LSERVO, 0, math.floor(1.4 / 20 * 4096));
    pwm.set_pwm(RSERVO, 0, math.floor(1.4 / 20 * 4096));
    time.sleep(0.5)
```

B.1.2 Encoders

Encoders allow the robot to get feedback on wheel rotations by counting the number of holes ("ticks") encountered by each encoder. Encoders are needed since a motor's response to the control signal varies due to factors such as floor friction, load on motors, wheel alignment, etc. When implementing the encoder library consider the following:

- Each encoder has 32 equidistant holes separated by 1/32 rotations.
- The maximum speed of the motors is approximately 0.80 revolutions per second, so the shortest time interval between two holes is approximately 39ms. At half speed, the interval increases to 78ms, at one fourth it increases to 156ms, and so on. These intervals are very large when compared to the speed at which a processor can run. Thus, the encoder reading functions should not block waiting to see a new hole. To solve this issue use I/O interrupts to collect the required information and use the functions to process the information and return the requested values. See interrupt programming in Python[3].

Consider the following questions:

- If you haven't seen any new holes since last time you called the encoder read function, and you call the function again, what speed should you return?
- In that case, how do you know whether the robot is moving very slow or just standing still?
- Do the encoders provide information about the direction that the wheels move (forward / backward)?

B.2 Robot Simulator

Webot's "e-puck" robot[4] (you can refer to the original "e-puck"[5] robot) has two wheels, each with approximate diameter 1.6" inches, and distance between wheels 2.28" inches, as shown in Figure 2. Note that Webots gives its distance measurements using the metric system.

B.2.1 Motors

Webots' e-puck robot has a left and right motor[6].
The *setVelocity* function specifies the velocity of each motor in radians per second.
Note that the maximum velocity the motor can rotate is $2\pi = 6.28$ rad per sec.

[3] Tutorial on Python signals: https://docs.python.org/2/library/signal.html
[4] https://cyberbotics.com/doc/guide/epuck
[5] http://www.e-puck.org/
6 Tutorial on Webots' Motor Control https://cyberbotics.com/doc/reference/motor

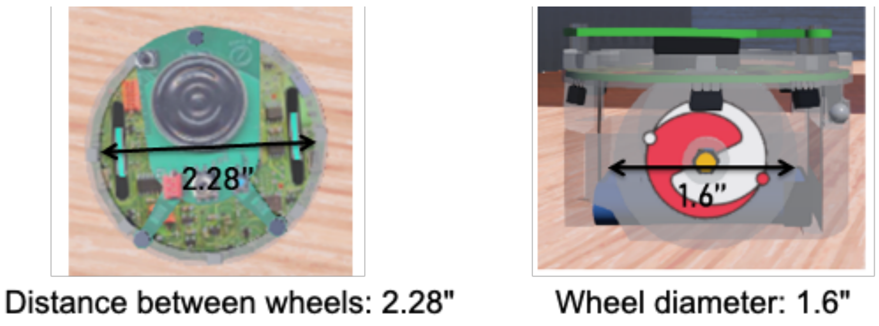

Figure 2. Webot "e-puck" robot.

B.2.2 Encoders

Encoders allow the robot to get feedback from motor rotations. In Webots, encoders are set using *PositionSensors*[7] attached to *RotationalMotors* through *HingeJoints*. The *PositionSensor* detects the distance travelled by the robot between a start and end point. The appropriate *getValue* function will return the *PositionSensor* readings in radians. You should check whether the *setVelocity* for the motor matches the encoder feedback. You can use the *getTime* function returning values in secodns to calculate motor rotations. You will need to write an encoder library to measure distances and instantaneous speed of the wheels based on encoder readings.

B.3 General Functions

Implement the following motor and encoder general functions as part of this lab.

B.3.1 Motor Functions

Motor functions:

- straightlineMotionV(V) - robot moves in a straight line at a linear velocity of "V" inches per second.
- straightlineMotionVX(V, X) - robot moves in a straight line at a linear velocity of "V" inches per second for a distance of "X" inches.
- straightlineMotionVT(V, T) - robot moves in a straight line at a linear velocity of "V" inches per second for "T" seconds.
- circleMotionRV(R, V) - robot moves in a circle of radius "R" at a constant linear velocity of "V" inches per second.
- circleMotionRVX(R, V, X) - robot moves in a circle of radius "R" at a constant linear velocity of "V" inches per second for a distance of "X" inches.
- circleMotionRVT(R, V, T) - robot moves in a circle of radius "R" at a constant linear velocity of "V" inches per second for "T" seconds.
- ~~setSpeedsPWM (pwmLeft, pwmRight) - should set the speed of the motors in PWM (physical robots only).~~
- setSpeedsRPS (rpsLeft, rpsRight) - should set the speed of the motors in RPS.
- setSpeedsIPS(ipsLeft, ipsRight) - this function is the same as the previous one, but this time, the speed is given in inches per second.
- setSpeedsVW(V, W) - should set the speed of the robot, so that the robot will move with a linear speed given by the parameter 'V' (in inches per second), and with an angular velocity 'W' (in radians per second). Positive angular velocities should make the robot spin counterclockwise.

B.3.2 Encoder Functions

Encoder functions:

[7] Tutorial on Webots' PositionSensor https://cyberbotics.com/doc/reference/positionsensor

- resetCounts() - should reset the number of ticks counted to zero.
- getCounts() - should return the left and right tick counts since the last call to *resetCounts*, or since the start of the program (if there were no calls to *resetCounts*). Note that the function should return the left and right tick counts as a tuple[8].
- getSpeeds() - should return the instantaneous left and right wheel speeds (measured in revolutions per second). Note that the function should return the left and right wheel speeds as a tuple[9].
- initEncoders() - should contain all code necessary for initialization.

C. Task Description

The lab consists of the following motion control tasks:

- Task 1: Motor Control – Rectangle ('Lab1_Task1.py')
- Task 2: Motor Control – Circles ('Lab1_Task2.py')

C.1 Task 1 – Motor Control - Rectangle

The robot follows the rectangle with length "H" and width "W", as shown in Figure 3. The robot starts at (-W/2,0) moving clockwise around the rectangle with the same constant speed "V" inch per sec. Print total navigation time "T". The TA will test for different values for "H", "W", and "V". If the motion cannot be completed with the given values, the robot should display an appropriate message.
Implement the following function:

- rectangleMotion(W, H, V) - main task function making the robot follow the rectangle with width "W", height "H", at a constant linear velocity of "V" inches per second.

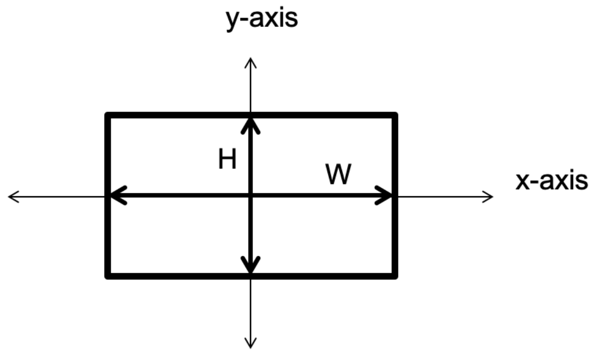

Figure 3. Rectangle shaped path to be followed by the robot.

C.2 Task 2 – Motor Control – Circles

The robot follows the two circles of radius "R1" and "R2", as shown in Figure 4. The robot starts at (0,0) moving clockwise around the circle with radius "R1". When reaching (0,0) again, it moves clockwise without stopping around the circle with radius "R2" at the same constant velocity "V" inch per sec. Print total navigation time "T". The TA will test for different values for "R1", "R2", and "V". If the motion cannot be completed with the given values, the robot should display an appropriate message.
Implement the following function:

- circlesMotion(R1, R2, V) - main task function making the robot follow the circles with radii "R1" and "R2", at a constant linear velocity of "V" inches per second.

[8] Tutorial on Python tuples: https://www.w3schools.com/python/python_tuples.asp
[9] Tutorial on Python tuples: https://www.w3schools.com/python/python_tuples.asp

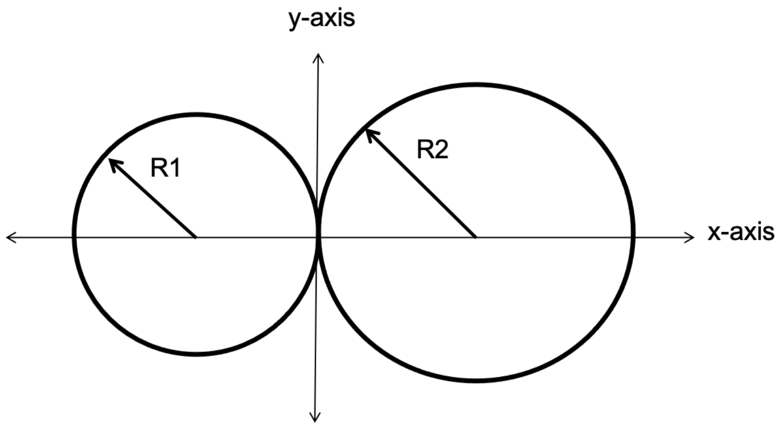

Figure 4. Circle shaped path to be followed by the robot.

D. Task Evaluation

Task evaluation is based on: (1) program code, (2) report, including a link to a video showing each different task, and (3) task presentation of robot navigation with the TA. Note that all functions and tasks to be implemented are required in future labs.

D.1 Programs (90 points)

The following section shows the rubric for the different tasks:

- Task 1 (45 points)
  - Robot travels the rectangle (10 points)
  - Robot travels the rectangle at given height "H" and width "W" values (10 points)
  - Robot travels at the specified "V" velocity (10 points)
  - Program computes and prints correct stopping time "T" (10 points)
  - Output message if robot cannot complete the motion (5 points)
- Task 2 (45 points)
  - Robot travels the circles (10 points)
  - Robot travels the circles at given "R1" and "R2" values (10 points)
  - Robot travels at the specified "V" velocity (10 points)
  - Program computes and prints correct stopping time "T" (10 points)
  - Output message if robot cannot complete the motion (5 points)

D.2 Report (10 Points)

The report is mandatory (without a report the full lab grade will be "0"). The report should include the following (points will be taken off if anything is missing):

1. Mathematical computations for all kinematics. Show how you calculated motions for left and right motors given the input parameters for each task. Also, show how you decide whether the movement is possible or not. (4 points)
2. Video uploaded to Canvas showing the robot executing the different tasks. You should include in the video a spoken description for each task. You can have a single or multiple videos. Note that videos will help assist task evaluations. (3 point)
3. Conclusions where you analyze any issues you encountered when running the tasks and how these could be improved. Conclusions need to state what portions of the work was done by each group members and show an insight of what the group has learnt (if anything) during the project. Phrases such as "everything worked as expected" or "I enjoyed the project" will not count as conclusions. (3 points)

D.3 Task Presentation

Task presentation needs to be scheduled with the TA. Close to the project due date, a timetable will be made available online to select a schedule on a first come first serve basis. Students must be present at their scheduled presentation time. On the presentation day, questions related to the project will be asked, and the robot's task performance will be evaluated. If it is seen that any presenter has not understood a significant portion of the completed work, points will be deducted up to total lab points. Students that do not schedule and attend presentations will get an automatic "0" for the lab.

NOTE for physical robot presentation: During presentations, robot calibration time is limited to 1 min. It is important that all members of the team attend and understand the work submitted.

E. Lab Submission

Each student needs to submit the programs and report through Canvas as a single "zip" file.

- The "zip" file should be named "yourname_studentidnumber_labnumber.zip"
- Videos should be uploaded to the media folder in Canvas. Name your files "yourname_studentidnumber_labnumber_tasknumber".
- The programs should start from a main folder and contain subfolders for each task.
- The report should be in PDF and should be stored in the same "zip" file as the programs.