Control of Mobile Robotics

CDA4621

Spring 2019

Lab 1

Kinematics

Total: 100 points

Due Date: 2-7-19 by 8am

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

A. Lab Requirements

The lab requires use of the "Robobulls-2018" robot hardware provided at no charge for the duration of the semester. Required software can be downloaded free of charge from the web. All labs are to be done by teams of one or two students depending on robot availability. Note that no diagrams or descriptions by hand will be accepted. Each group is required to submit its report through Canvas. Penalties will be applied for late submission (see syllabus). All documents need to be in PDF, while a single "zip" file must be uploaded to Canvas containing all requested files.

A.1 Hardware Requirements

The "Robobulls-2018" is the main robot hardware used for the course.

A.2 Software Requirements

Python (or optionally C++).
Tutorial to python tuples: https://www.w3schools.com/python/python_tuples.asp
Tutorial to python signals: https://docs.python.org/2/library/signal.html

B. Servos and Encoders Description

The robot components for this lab consist primarily of "Servos" and "Encoders". The main task is to implement the "Kinematics" for "Differential Navigation".

B.1 Servos Basics

Read the section "Communication Protocol" on the servo datasheet[1]. Make sure you are able to answer the following questions:

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

```
while true:
        servo.writePWMMicroseconds(1700)
        delay(20)
        servo.writePWMMicroseconds(1300)
        delay(20)
```

[1] http://www.mantech.co.za/Datasheets/Products/900-00008-65S.pdf

B.2 Encoders

Encoders allow the robot to get feedback on motor rotations by counting the number of holes encountered by the encoders. Encoders are needed since a servo's response to the control signal varies due to several factors including floor friction, load on the motors, noise, etc.

In this section you will write an encoder library that will allow you to "measure distances" (count holes) and to measure the instantaneous speed of the wheels. To do so you are required to implement the following 4 functions. It is suggested you use some of the content in "python\4_ServoSetup\servos.py":

- resetCounts() - should reset the tick count (number of holes 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 counts as a tuple. Link to tutorial.
- void 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. Link to tutorial.
- void initEncoders() - should contain whatever code is necessary for initialization.

When implementing the encoder library take into account the following:

- Each encoder has 32 equidistant holes separated by 1/32 rotations.
- The maximum speed of the servos 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 functions getCounts and getSpeeds should not block waiting to see a new hole. To solve this issue use IO interrupts to collect the required information and use the functions getCounts and getSpeeds to process the information and return the requested values. To learn to use interrupts, use this Link. Additionally, the provided python file in "python\3_EncoderInterrupt\encoders.py" shows a correct use of interrupts.

Consider the following questions: If you haven't seen any new ticks since the last time you called *getSpeeds*, and you call the function *getSpeeds* 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 our encoders provide information on the direction that the wheels move (forward / backward)?

B.3 Servos Control

Use "python\4_ServoSetup\servos.py" to implement the following 4 functions. Consider that the wheels' diameter is 2.61" and that the wheels are separated by 3.95" approximately, although this may vary from robot to robot since some may be bent.

- void calibrateSpeeds() - should use the encoder functions implemented on section B.1.2 to create a mapping from the servos input (micro seconds) to the servos output (wheel speed in revolutions per second). In other words, measure the speeds of the wheels for different input values and store the results. You should do this for both wheels since the servo's output won't necessarily be the same for both wheels. This information will be used by the following functions.

- setSpeedsRPS (rpsLeft, rpsRight) - should set the speed of the motors as in servos (part 1), but this time, the input parameters indicate the speeds at which each wheel should spin.
- 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' (given in radians per second). Positive angular velocities should make the robot spin counterclockwise.

C. Tasks Descriptions

The robot components for this lab consist primarily of "Servos" and "Encoders". The main task is to implement the "Kinematics" for "Differential Navigation".

C.1 Task 1

Speeds - Run the function *setSpeeds(0,100)*. Use the encoders to plot the *instantaneous speed*[2] of the right servo (in revolutions per second) vs time. Take measurements every 30 ms for approximately 10 seconds. Before taking any measurements wait 1 second to allow the motor to reach the desired speed. In the y axis, only include the effective range, that is, for example, if you only get values from 0.6 to 0.7, the range in the y axis should be the same, it should NOT be from 0 to 0.7.

Aspects to think about:

- Is this value constant?
- If not, does the plot look random or does it have a pattern? Think of reasons that might explain the behavior.

C.2 Task 2

Encoders - In a single graph use the encoders to plot the speed of both wheels (in revolutions per second) vs the input to the implemented function *setSpeeds*. Measure the speeds of the wheels from 1.3 ms to 1.7 ms in increments of 0.01. Provide the plot in the report. Things to think about:

- How similar do both servos react to the same input? Does the difference look like random noise? If not, what could be causing the difference? Why is it a good idea to use software calibration?
- Before measuring the speed, you should allow some time for the servo to stabilize its speed. Also, you should use the average speed over a time window instead of the instantaneous speed.

C.3 Task 3

Move Straight - Implement the program 'Forward.py'. The program should make the robot stay still until the select button is pressed. After pressing select, the robot should move forward on a straight line for "X" inches. The robot should complete the movement in "Y" seconds. The robot should stop based on encoder readings (distance traversed), not on the time past. "X" and "Y" are parameters provided through terminal interfacing. The program should work for any values "X" and "Y" for which it is possible to complete the movement on the given time. If the robot

[2] https://en.wikipedia.org/wiki/Speed#Instantaneous_speed

can't complete the movement in the given amount of time, instead of moving forward, the program should display an appropriate message.

During the task presentation, the TA will assign different values for "X" and "Y".

C.4 Task 4

Follow an "S" shaped path - Implement the program 'SShape.py. The program should make the robot move in two semicircles of radius "R1" and "R2" respectively, as shown in Figure 1. The robot should stay still until given a command on the terminal. When the terminal command is issued, the robot should perform the first semicircle in a clockwise movement. After completing the first semicircle, the robot should wait for another command, and then perform the second semicircle in a counterclockwise manner. The whole movement must be completed in "Y" seconds moving at the same constant speed for both halves. The robot should stop based on encoder readings (distance traversed), and not based on a given time.

"R1", "R2" and "Y" will be provided through user input. The program must work for any values of "R1", "R2" and "Y". If the motion cannot be completed in the given time, after issuing the command for the first time, the robot should refuse and display the appropriate message.

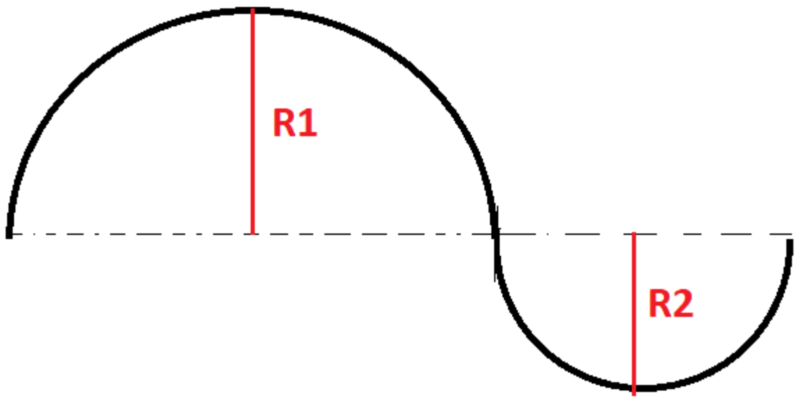

Figure 1. "S" shape path to be followed by the robot.

- Observe that, independently of the trajectory, the robot moves by fixing the speeds of the wheels for a given amount of time (as in the "S" shape shown in Figure 1). Taking this into consideration, can the robot move in any kind of trajectory? Consider an ellipse as an example.

During the task presentation, the TA will assign different values for "R1", "R2" and "Y".

D. Task Evaluation

Task evaluation involves: (1) a task presentation of robot navigation with the TA, (2) the accompanying code to be uploaded to Canvas, and (3) task report to be uploaded to Canvas. All uploaded documents will be scanned through copy detection software.

In what follows note that while tasks 1 and 2 are not evaluated in the presentation, they are evaluated as part of the report (see section D.2). Also, the functions that need to be implemented in sections B.1, B.2 and B.3 are required to complete all tasks as well as future labs.

D.1 Task Presentation (70 points)

The task presentation (same week when reports are due) needs to be scheduled with the TA. Close to the project due date, a time table will be made available online from which groups will be able to select a schedule on a first come first serve basis. All team members 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 either or both presenters have clearly not understood a significant portion of the completed work, points will be deducted up to and including full points for the task presentation.

<u>It is important that all members of the team understand the work submitted.</u>

To do so, the submitted code will be downloaded at that time from Canvas and uploaded to the robot. During task presentation the TA will verify that the robot performs as requested, that is that the robot waits for input, uses the encoders to detect when the movement is done, that the robot completes the tasks in the correct timing, that the robot detects impossible movements and shows an appropriate display, that the robot moves the correct distance in task 3 and has an acceptable radius on task 4, and that the robot moves in a straight line for task 3 and in a circle on task 4.

The following table shows the rubric for the tasks presentation:

- Task 1 – 0 points (report only)
- Task 2 – 0 points (report only)
- Task 3 – 35 points
    - Wait for input – 5 points
    - Moves straight – 6 points
    - Moves adequate distance – 6 points
    - Uses encoders to measure distance – 6 points
    - Completes movement in correct time – 6 points
    - Detects impossible movement – 6 points
- Task 4 – 35 points
    - Wait for input – 5 points
    - Moves in a circle – 6 points
    - Acceptable radius – 6 points
    - Uses encoders to measure distance – 6 points
    - Completes movement in correct time – 6 points
    - Detects impossible movement – 6 points

D.2 Task Report (30 Points)

The accompanying task report needs to be uploaded to Canvas as a PDF file together with ALL the files required to run the robot navigation. Upload all files into a single "zip" file. The task report should include ALL of the following (points will be taken off if anything is missing):

1. List of all code files uploaded to Canvas. All requested code must be included as a list in the main report, adding a one-line description to each of the files being uploaded (1 point).
2. Include in the main report all the equations used to calculate the speed of the wheels in the encoder library (4 points), and the equations used in functions *setSpeedIPS* (4 points) and *setSpeedsvw* (4 points). A brief description should be provided for each equation. Note that the encoder library should calculate instantaneous speed, not an average speed.
3. A section showing plots:
    a. The plot for "Task 1" (*instantaneous speed*[3] *of the right servo vs time* – 3 points).
    b. The plot for "Task 2" (*speed of both wheels vs input* – 3 points).

[3] https://en.wikipedia.org/wiki/Speed#Instantaneous_speed

All plots must include title, axis names, units, sufficient tick marks and legend (if plotting more than one graph). Also, each data point should clearly be marked with a symbol. Plots of a variable vs time should always place time on the x-axis. See sample plot in Figure 4. Points will be taken off for not following the specified format.

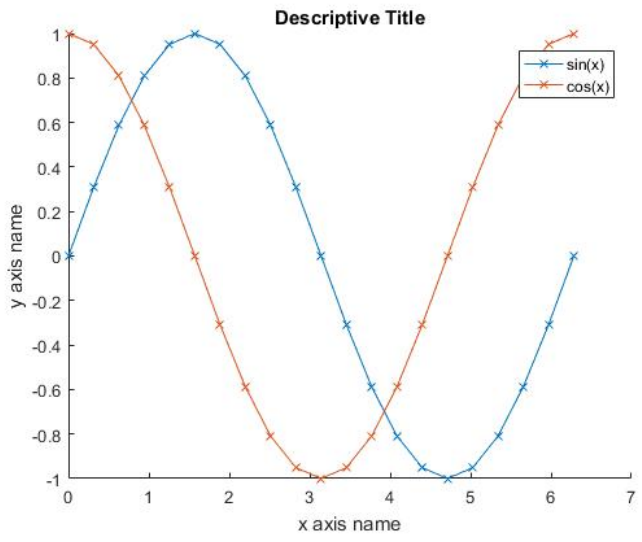

Figure 4. Sample Plot

4. Mathematical solutions for kinematics “Task 3” (3 points) and “Task 4” (3 points). Show how you calculated the speeds of the left and right servos given the input parameters. Also, show how do you decide whether the movement is impossible.
5. Provide a video (2 points) where the robot is shown performing “Task 3” and “Task 4” (Upload to YouTube and provide the link).
6. Conclusions (3 points) where you analyze any issues you encountered when running the tasks and how these could be improved. Conclusions need to 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.