Control of Mobile Robotics

CDA4621

Fall 2017

Lab 2

Kinematics

Total: 100 points

Due Date: 10-30-17 by 8am

The assignment is organized according to the following sections: (A) Lab Requirements, (B) Task Description, and (C) Task Evaluation. Note that only the questions referred to in section (C) need to be answered. Nonetheless, you should be able to answer all questions in section (B) since some of them will come up in a quiz and possibly during a theory exam.

A. Lab Requirements

The lab requires use of the "Robobulls-2017" 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 two students. 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 submitting late assignments (see syllabus). All documentation needs to be in PDF.

A.1 Hardware Requirements

The "Robobulls-2017" (Figure 1) is the main robot hardware used for the course.

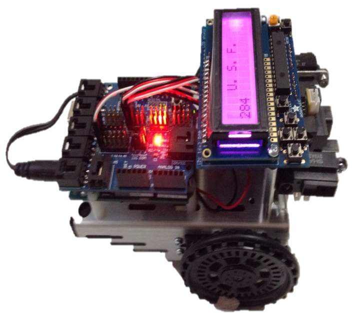

Figure 1: Robobulls-2017

A.2 Software Requirements

Arduino Software (Version 1.8.3 or later): https://www.arduino.cc/en/Main/Software

B. Task Description

The lab is divided into two parts: (1) "Robot Basics" where you will learn how to use the robot hardware, and (2) proportional control and wall following.

B.1 Robot Basics

This section will cover basic usage of the hardware to gain an insight into what type of information can be found on a datasheet, how to properly use the robot hardware, and how to avoid typical errors made while working with the robot hardware.

B.1.1 Distance Sensors

B.1.1.1 Distance Sensor Basics

In this section, we will study some theory of how to use the sensors in a correct manner.
Go through the datasheet of the long[1] and short[2] distance sensors and read the documentation of the Arduino's function "analogRead"[3]. Think about the following:

1. What is the distance measuring range on each type of distance sensor?
2. Calculate how much current do the four sensors consume in total? Do this for both the max supplied current and the typical value.
3. How long does one have to wait after powering up a sensor until the first reading is available? Note that different types of distance sensors will have different values.
4. After the first measurement is available, how long does it take for the sensors to produce a new value?
5. What information do you need to convert raw sensor measurements (the voltage returned by the sensors) to distance measurements (their corresponding conversion of the raw measurement)?
6. The datasheet shows two methods for converting raw values to distance measurements. Which is better (theoretically) and why?
7. Is there a one-to-one correspondence between raw measurements and distance measurements? In other words, is there a unique distance measurement for each raw value?
8. Think of methods that would allow you to make sure that a distance sensor always works in its specified distance range.
9. Look at the plot distance vs voltage for the short distance sensor. Is the precision constant across all distances? If not, do you expect the sensor to be most precise at close or long ranges? Why? (Hint: think how distance versus voltage produce a range of errors for different values.)
10. What does the function "analogRead" return?
11. How many bits does Arduino's analog to digital converter use? How many possible values does the function "analogRead" return?
12. What resolution does the "analogRead" function yield?
13. How long (in milliseconds) does it take to read an analog input?

B.1.1.2 Converting raw values to distance measures

Implement the first two functions of the header file "MySharpSensor.h":

1. float shortToInches(int value)
2. float longToInches(int value)

Each function should return the distance associated to the raw measurement "value" for the corresponding type of distance sensor.
Use the implemented functions to make a plot of "real distance" vs "measured distance" for the long-distance sensor and the front short distance sensor. Make sure the "real distance" should be on the x-axis and "measured distance" should be on the y-axis.

[1] https://www.sparkfun.com/datasheets/Sensors/Infrared/gp2y0a02yk_e.pdf
[2] http://www.sharp-world.com/products/device/lineup/data/pdf/datasheet/gp2y0a41sk_e.pdf
[3] https://www.arduino.cc/en/Reference/AnalogRead

- For the short distance sensor include measurements from 2" to 12" in steps of 1".
- For the long distance sensor include measurements from 5" to 60" in steps of 5".

For each distance, take at least 5 measurements (add them all to the plot) to see how the error varies. On each plot, include the line $y = x$. In an ideal world, all measurements should fall on this line, but in practice this won't be the case. When taking measurements, do not average multiple values (you will be required to do that on the next section).

B.1.1.3 Improving results by taking multiple measurements

As you should have noticed on the previous section, the sensors are very noisy. Standing still, and taking successive measurements will give multiple results. In this section, we will analyze and implement techniques to combine multiple measurements in order to reduce the error.
We will look at the two most basic methods for doing this: averaging and taking the median[4].

1. Analyze what advantages the median has over the average and vice versa.
2. Do we get similar results by "averaging the raw values and then converting the result to inches" and "converting the raw values to inches and then averaging"? Does your answer change when using the median?
3. What problem does the following code have (assume a short distance sensor is connected to the analog port 0)?

```
int raw[11];
for(int i=0;  i< 11; i++) {
     raw[i] = analogRead(0);
}
int result = combine(raw);  // apply whatever combination method such as averaging
```

4. When a robot is performing a task, sensing is one among many tasks the robot has to do. On many occasions, a robot must perform real time tasks for which it must guarantee responsiveness against certain events, e.g., detect a cliff before the robot falls, or just detecting a line in the floor – as will be the case in assignment 3. Taking this info into account, point out what problem does the following code have:

```
int raw[21];
for(int i=0;  i< 21; i++) {
     raw[i] = analogRead(0);
     delay(20);
}
int result = combine(raw);  // apply whatever combination method such as averaging
```

5. Consider the following question: How many measurements should I combine? If the robot is immobile we can expect the more measurements we combine the smaller the error. On the other hand, consider the case in which the robot is moving. Consider the 2 situations shown in Figure 2:

[4] https://en.wikipedia.org/wiki/Median

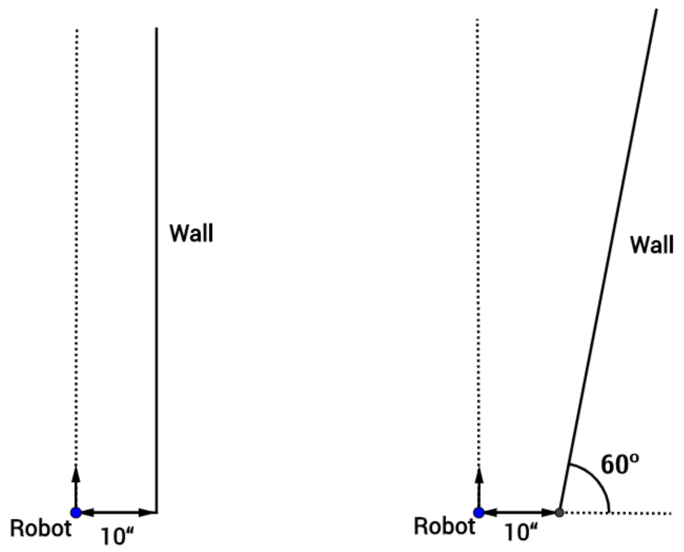

Figure 2. Robot moving along a wall.

In both cases shown in Figure 2, the robot is moving vertically along the dotted line at a constant speed of 10 inch/sec. In the left graph, the wall is parallel to the path of the robot, while in the right graph the wall makes an angle of 60 deg with the horizontal line. Assume that, in order to find out the distance to the wall, the robot will average 26 measurements, the first one taken at the starting position, and afterward taking one new measurement every 20 ms. What will be the average in both cases? In the right graph, does the average correspond to the current distance of the robot to the wall? In the right graph, how much error does the average have with the current distance to the wall? In other words calculate "abs(last measurement - average)". Taking this information into consideration, think on the advantages and disadvantages of averaging few samples vs many samples.

6. Implement the rest of the functions in the header file "MySharpSensor.h":

```
void takeNewMaeasurement(int sensor)
float getCombinedDistace(int sensor)
```

In the functions above, the input "sensor" is an identifier of the distance sensor being used. 0 should represent the left sensor, 1 the front short sensor, 2 the right sensor, and 3 the long front sensor. The first function should make a new measurement and store it in a buffer. The second function should grab the last N distances from 'sensor' and return the combined value. You are free to use any combination algorithm you see fit (mean, median, etc. - choose the one you think is the best), and you are also free to choose any value of the parameter N.

After implementing the functions, redo the plot on section B.1.1.2 but this time by combining multiple measurements to reduce the error for each data point. Again, do not expect the points to fall precisely on the line $y = x$. For each distance take at least 5 combined measurements.

B.2 Proportional control and Wall Following

The goal in this section is to perform close loop control using distance sensors and robot motor actuators. The section consists of 2 tasks, the second building upon the first. Figure 3 presents a diagram for close loop control, with robot velocity proportional to its desired distance to a goal, *r*(*t*), where the goal may be any object such as a wall or an obstacle. As the robot gets closer to the desired distance to the goal, the error *e*(*t*) will modify the control signal *u*(*t*) until such error becomes 0. In our case the control signal corresponds to robot motor velocity until it becomes 0 when the robot reaches a specific distance to the goal.

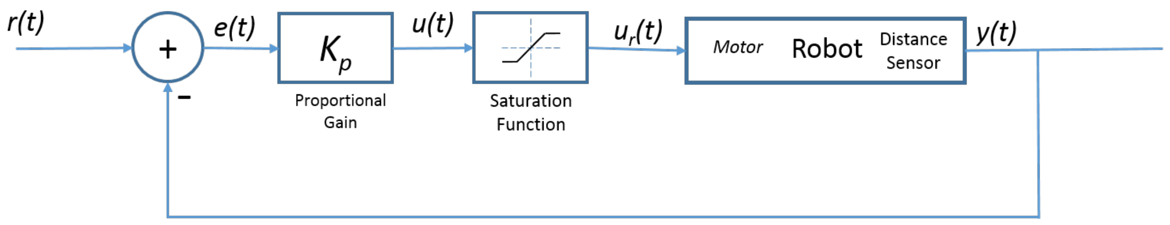

Figure 3: Close loop control of the robot velocity proportional to its distance to a goal.

Eq. 1-5 summarizes the diagram in Figure 3:

$$u(t) = K_p * e(t) \qquad \text{(Eq. 1)}$$
$$u_r(t) = f_{Sat}(\,u(t)) \qquad \text{(Eq. 2)}$$
$$u_r(t) = f_{Sat}(\,K_p * e(t)\,) \qquad \text{(Eq. 3)}$$
$$e(t) = r(t) - y(t) \qquad \text{(Eq. 4)}$$
$$u_r(t) = f_{Sat}(\,K_p\,(r(t) - y(t))\,) \qquad \text{(Eq. 5)}$$

where:

$r(t) = desired\ distance\ to\ the\ goal$
$y(t) = distance\ from\ robot\ to\ the\ goal$
$e(t) = distance\ error$
$K_p = proportional\ gain\ or\ correction\ error\ gain$
$u(t) = control\ signal\ corresponding\ to\ robot\ velocity$
$f_{sat} = Saturation\ Function\ (See\ explanation\ below)$
$u_r(t) = control\ signal\ corresponding\ to\ saturated\ robot\ velocity$
$u_r(t) \rightarrow$ setSpeedsIPS(u_r, u_r) //function implemented in the first assignment

On the formulas above, f_{sat} is a function that limits the inputs to the function "setSpeedsIPS".

$$f_{sat}(x) = f(x) = \begin{cases} -6 & if \quad x < -6 \\ x & if \ -6 \le x \le 6 \\ 6 & if \quad 6 < x \end{cases}$$

B.2.1 Wall Distance

You need to implement a program called "wallDistance.ino". The program should implement the above equations (1-5) in the close loop control program where the control signal *u*(*t*), corresponding to robot motor velocity, will move the robot either forward or backward from a starting position until it reaches a distance of 5 inches from the wall, i.e. *r(t)=5*, as shown in Figure 4. The distance should be measured using the front short distance sensor.

The day of the presentation, the robot will be placed at different distances from the wall (either closer than 5 inches or further away). The robot should react accordingly. Also, after reaching the goal, the robot will be moved both forward and backward. The program should automatically detect this change and go back to the goal position.

For the report, you will be required to test 6 different values of K_p(0.2, 0.6, 0.9, 1.2, 1.5, 5.0). For each value you will need to provide a graph showing Distance from Goal vs. Time. On the day of the presentation, you will have to choose the value that you think is best.

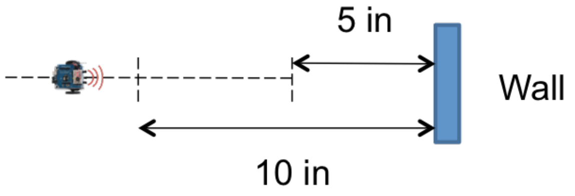

Figure 4: Wall Distance Task – starting at 10 inches

B.2.2 Wall Following

Implement a program called "wallFollowing.ino". The program should make the robot move around a wall while keeping a distance of 5 inches from the wall to the side (as shown in Figure 5). Use the same close loop control implementation in Task 1 with appropriate proportional gains to control front distance to the wall where the robot needs to turn. That is, as long as there is nothing in its front, the robot should move at full speed forwards (while following the wall in parallel). When there is a wall in its front, the robot should reduce its linear speed and it should not approach closer than 5 inches. On the day of the presentation, the program will be tested using multiple types of walls (both straight and curved walls).

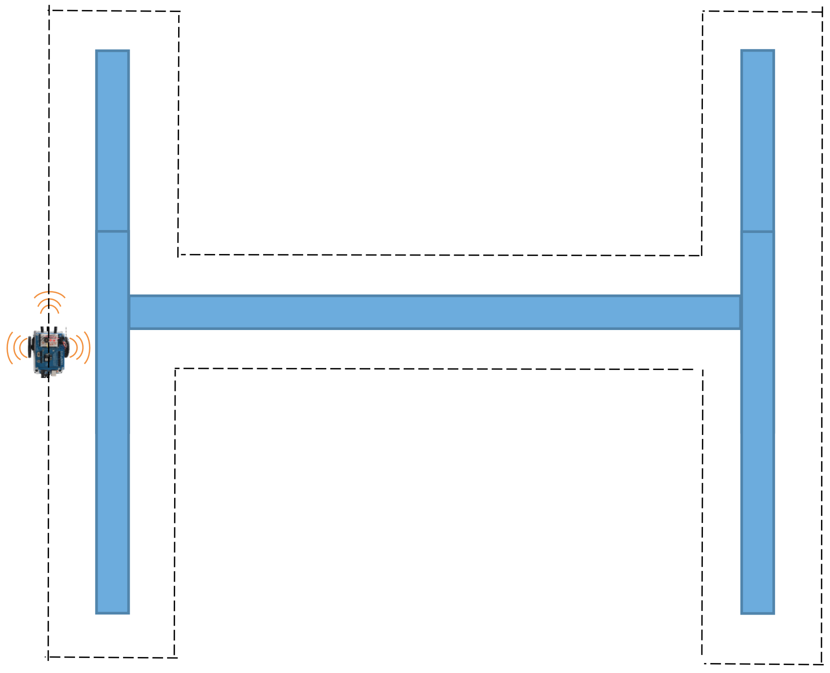

Figure 5: Wall following task

C. Task Evaluation

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

C.1 Task Presentation (70 points)

The task presentation 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. To do so, the submitted code will be downloaded from Canvas and uploaded to the robot.

C.2 Task Report (20 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 in the list. A one-line description for each file to which task it relates to must be added.
2. The plots for sections B.1.1.2 ("real distance vs measured distance"), B.1.1.3.6 ("real distance vs measured distance"), and B.2.1 ("distance to the wall vs. time", one for each k_p). 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 6.

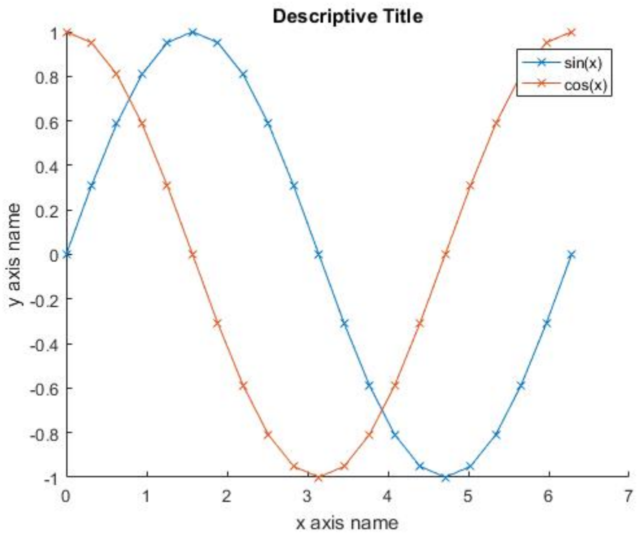

Figure 6. Sample Plot

3. Brief explanation of the equations described in section B.2.1
4. Conclusions 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.

C.3 Online Quiz (10 Points)

An online quiz based on section "Robot Basics" will be due 2 weeks after the assignment presentation.

C.4 Suggested Submission Schedule

Week 2 – Turn in code and plots of section B.1
Week 4 – Turn in the code for section B.2 and the full report.