Control of Mobile Robotics

CDA 4621

Fall 2017

Lab 1 - Kinematics

Group 4

Blaine Oakley

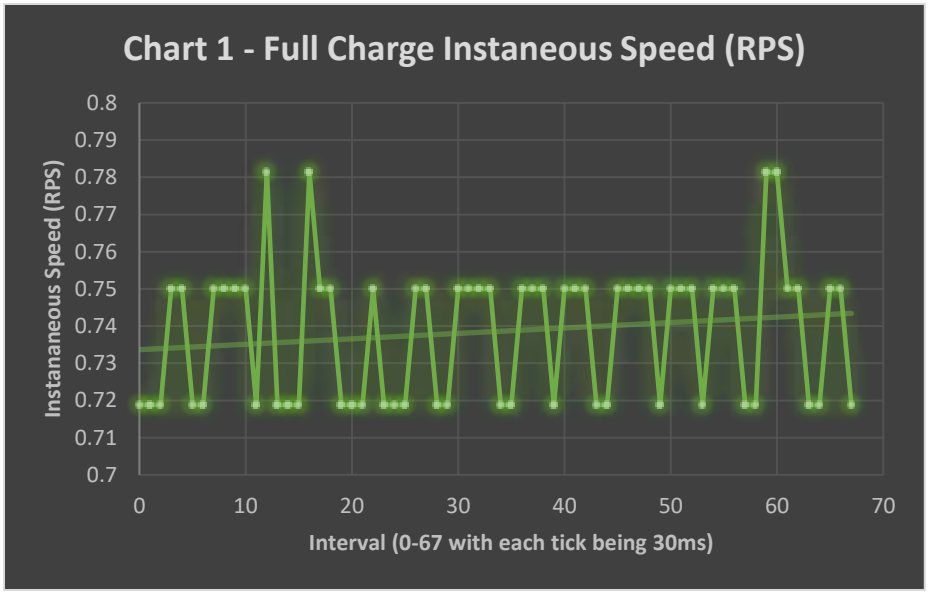
Boyang Wu

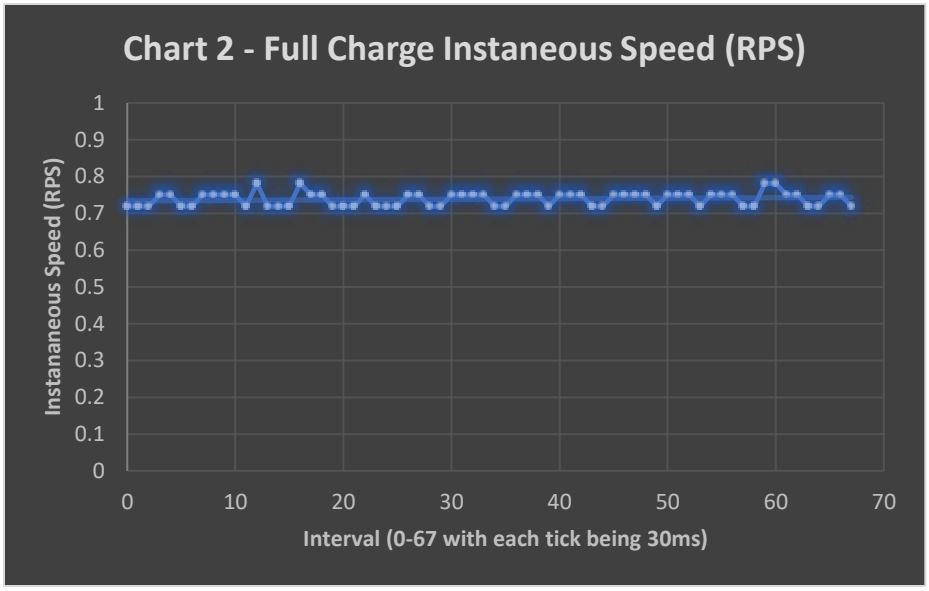
Task 1

Run the function setSpeeds(0,100). Use the encoders to plot the instantaneous speed4 of the right servo (in revolutions per second) vs time. Take measurements every 30ms for approximately 2 seconds (about 67 measurements in total). Before taking any measurements wait 1 second to allow the motor to reach the desired speed. Aspects to think about:

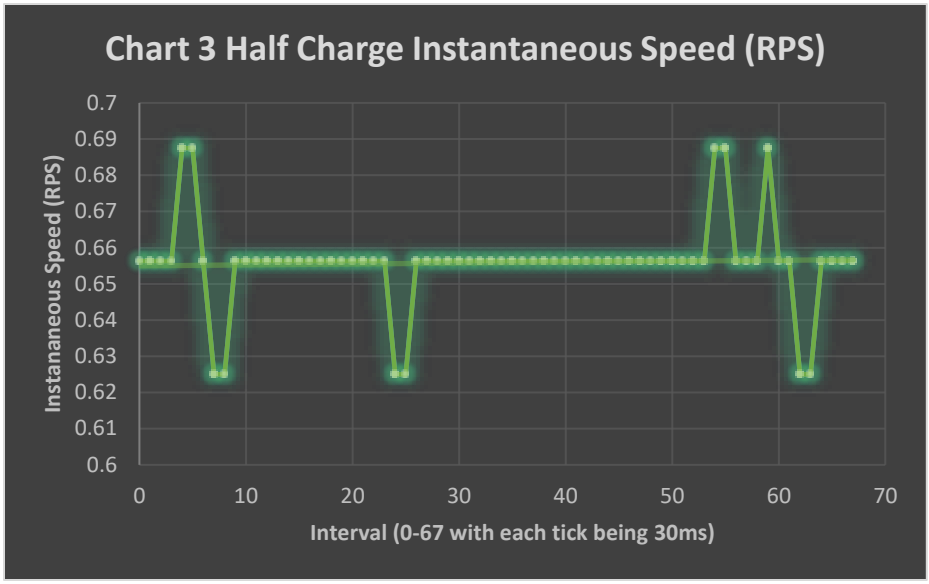
Table 1

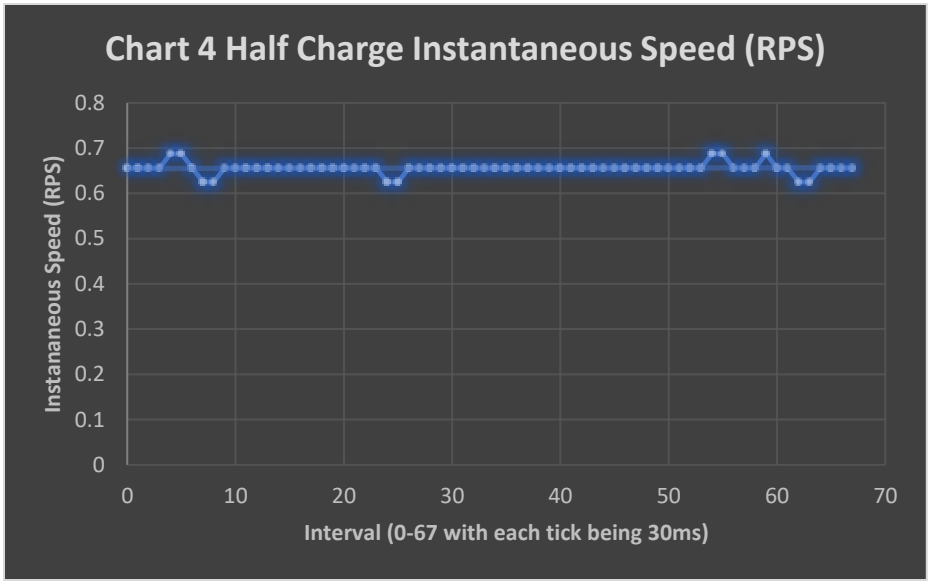
Instantaneous Speed (RPS) every 30ms			
i	Delay	Full Charge	Half Charge
0	0	0.7188	0.6563
1	30	0.7188	0.6563
2	60	0.7188	0.6563
3	90	0.7500	0.6563
4	120	0.7500	0.6875
5	150	0.7188	0.6875
6	180	0.7188	0.6563
7	210	0.7500	0.6250
8	240	0.7500	0.6250
9	270	0.7500	0.6563
10	300	0.7500	0.6563
11	330	0.7188	0.6563
12	360	0.7813	0.6563
13	390	0.7188	0.6563
14	420	0.7188	0.6563
15	450	0.7188	0.6563
16	480	0.7813	0.6563
17	510	0.7500	0.6563
18	540	0.7500	0.6563
19	570	0.7188	0.6563
20	600	0.7188	0.6563
21	630	0.7188	0.6563
22	660	0.7500	0.6563
23	690	0.7188	0.6563
24	720	0.7188	0.6250
25	750	0.7188	0.6250
26	780	0.7500	0.6563
27	810	0.7500	0.6563
28	840	0.7188	0.6563
29	870	0.7188	0.6563
30	900	0.7500	0.6563
31	930	0.7500	0.6563
32	960	0.7500	0.6563
33	990	0.7500	0.6563
34	1020	0.7188	0.6563
35	1050	0.7188	0.6563

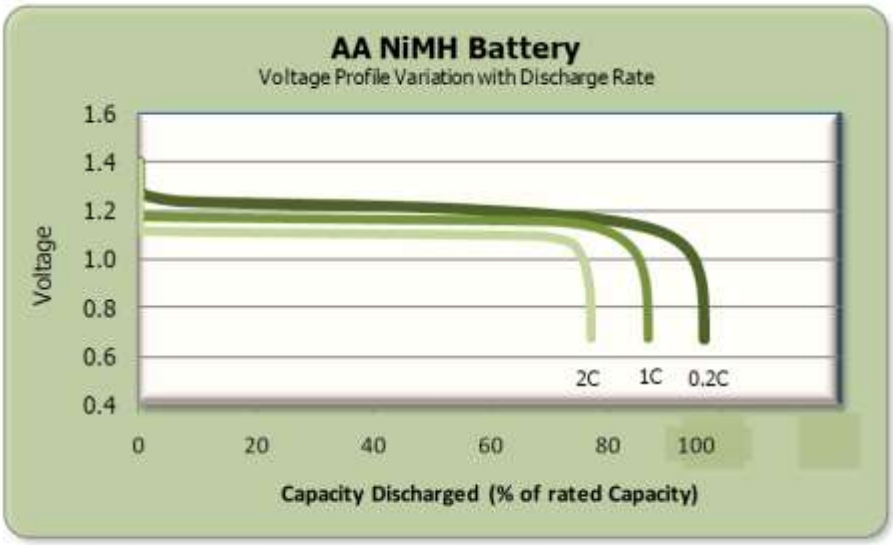




36	1080	0.7500	0.6563
37	1110	0.7500	0.6563
38	1140	0.7500	0.6563
39	1170	0.7188	0.6563
40	1200	0.7500	0.6563
41	1230	0.7500	0.6563
42	1260	0.7500	0.6563
43	1290	0.7188	0.6563
44	1320	0.7188	0.6563
45	1350	0.7500	0.6563
46	1380	0.7500	0.6563
47	1410	0.7500	0.6563
48	1440	0.7500	0.6563
49	1470	0.7188	0.6563
50	1500	0.7500	0.6563
51	1530	0.7500	0.6563
52	1560	0.7500	0.6563
53	1590	0.7188	0.6563
54	1620	0.7500	0.6875
55	1650	0.7500	0.6875
56	1680	0.7500	0.6563
57	1710	0.7188	0.6563
58	1740	0.7188	0.6563
59	1770	0.7813	0.6875
60	1800	0.7813	0.6563
61	1830	0.7500	0.6563
62	1860	0.7500	0.6250
63	1890	0.7188	0.6250
64	1920	0.7188	0.6563
65	1950	0.7500	0.6563
66	1980	0.7500	0.6563
67	2010	0.7188	0.6563







(Fig. 8) Voltage Profile Variation with Discharge Rate

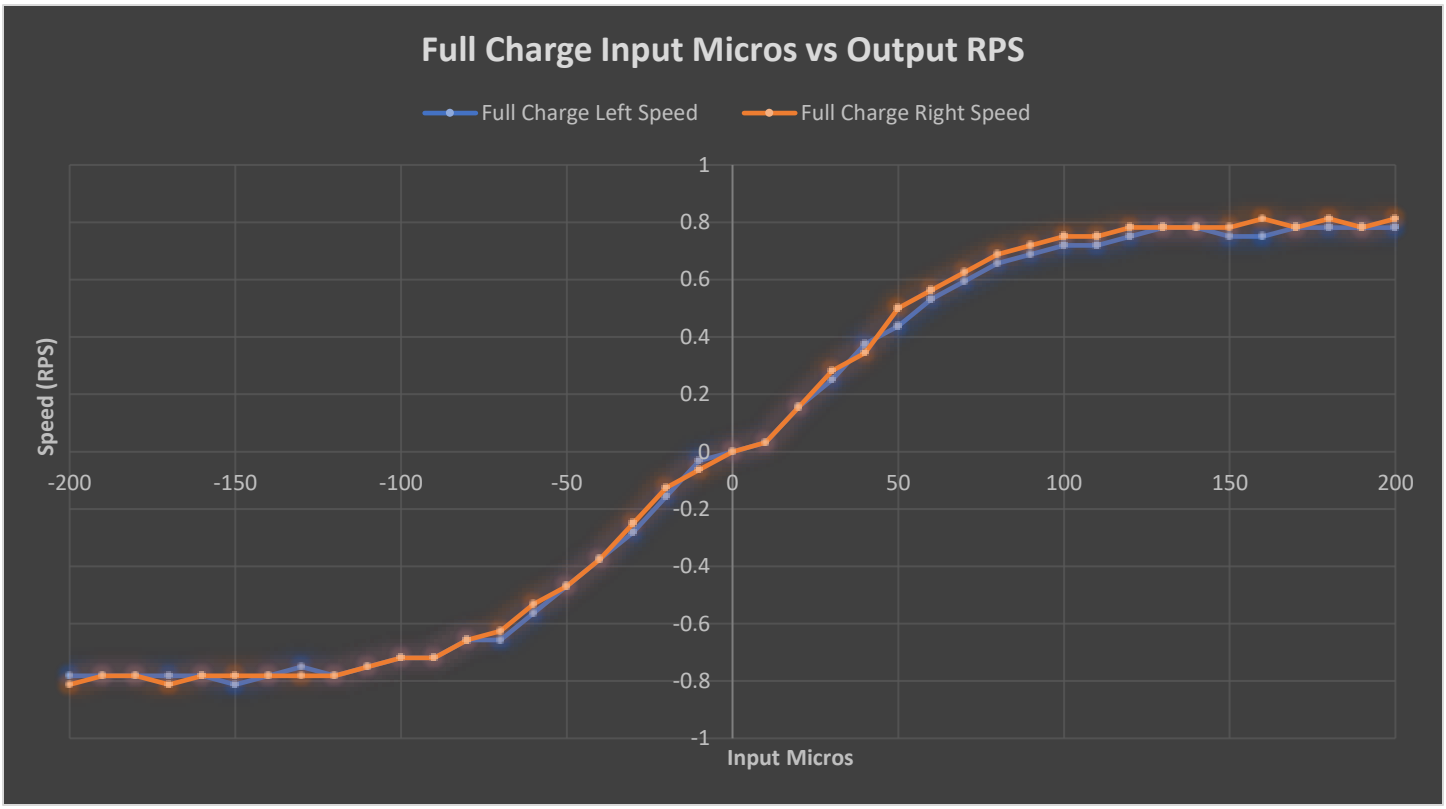
1) Is this value constant?
   a) We did two plots, one with full battery charge and other with roughly half charge (1.18V on multimeter with a 100ohm load).
   b) With a lower RPS range (Charts 1 / 3) we can see that the value changes somewhat, but is generally constant when switching to a larger y-range (Charts 2 / 4).
   c) The values change with battery charge, but the lines are still constant.
   d) Source for NiMH battery voltage graph taken from Energizer site (PDF also included in upload): data.energizer.com/pdfs/nickelmetalhydride_appman.pdf

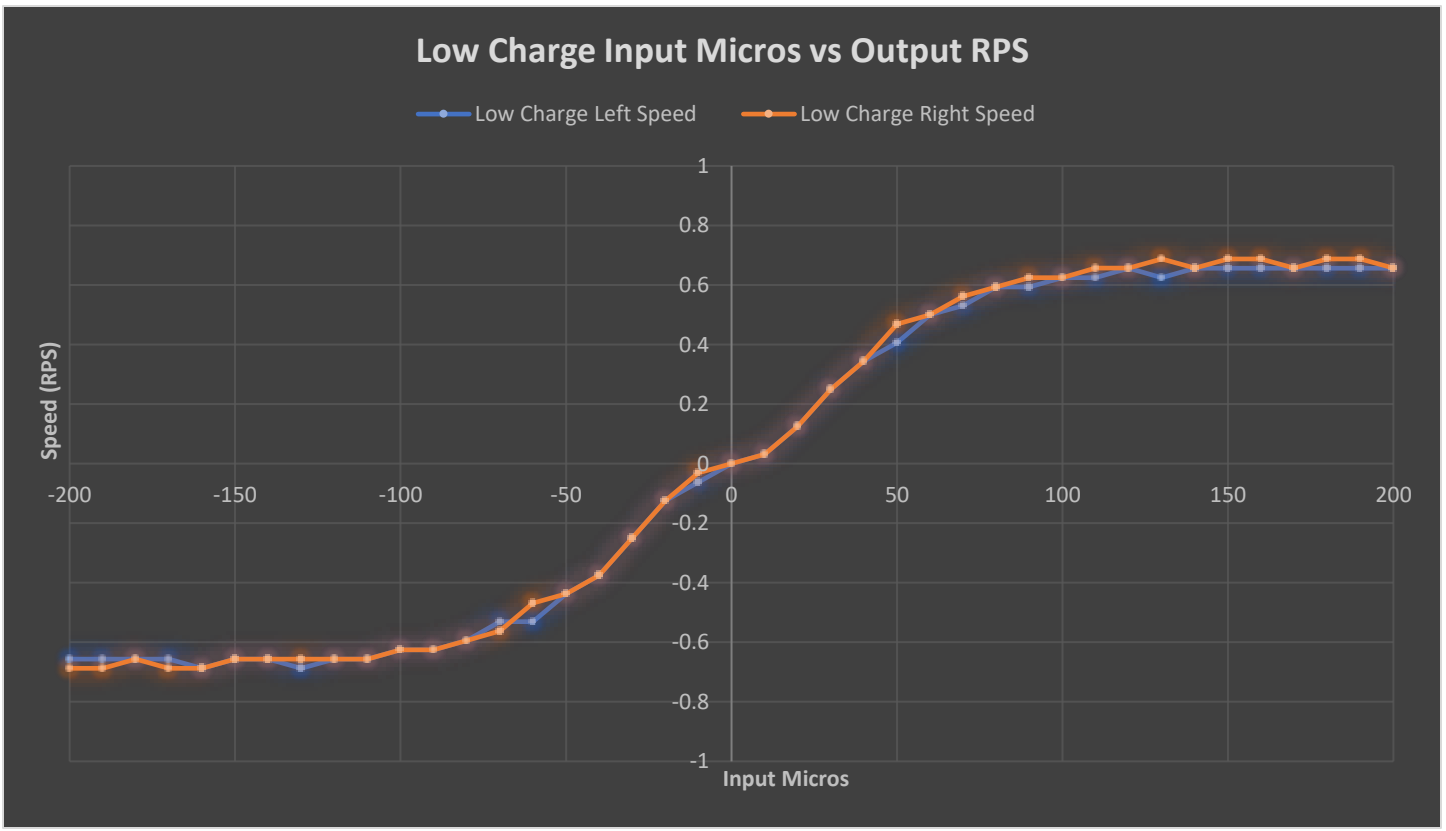
2) If not, does the plot look random or does it have a pattern? Think of reasons that might explain the behavior.
   a) Plot is constant at each RPS value shown in the charts. The RPS value difference between full and half charge could be attributed to the voltage drop and thus a lower max servo speed.
   b) There could be issues with how the values are read in, as for an instantaneous speed, the encode will not be able to read enough data values to get a more consistent RPS value, and thus such as in Chart 1, the RPS could oscillate between two values.
   c) A few spikes to even higher or lower values could be the result of encoder error in general.

*** Note that the charts are also uploaded in .xlsx files if bigger ones are needed. ***

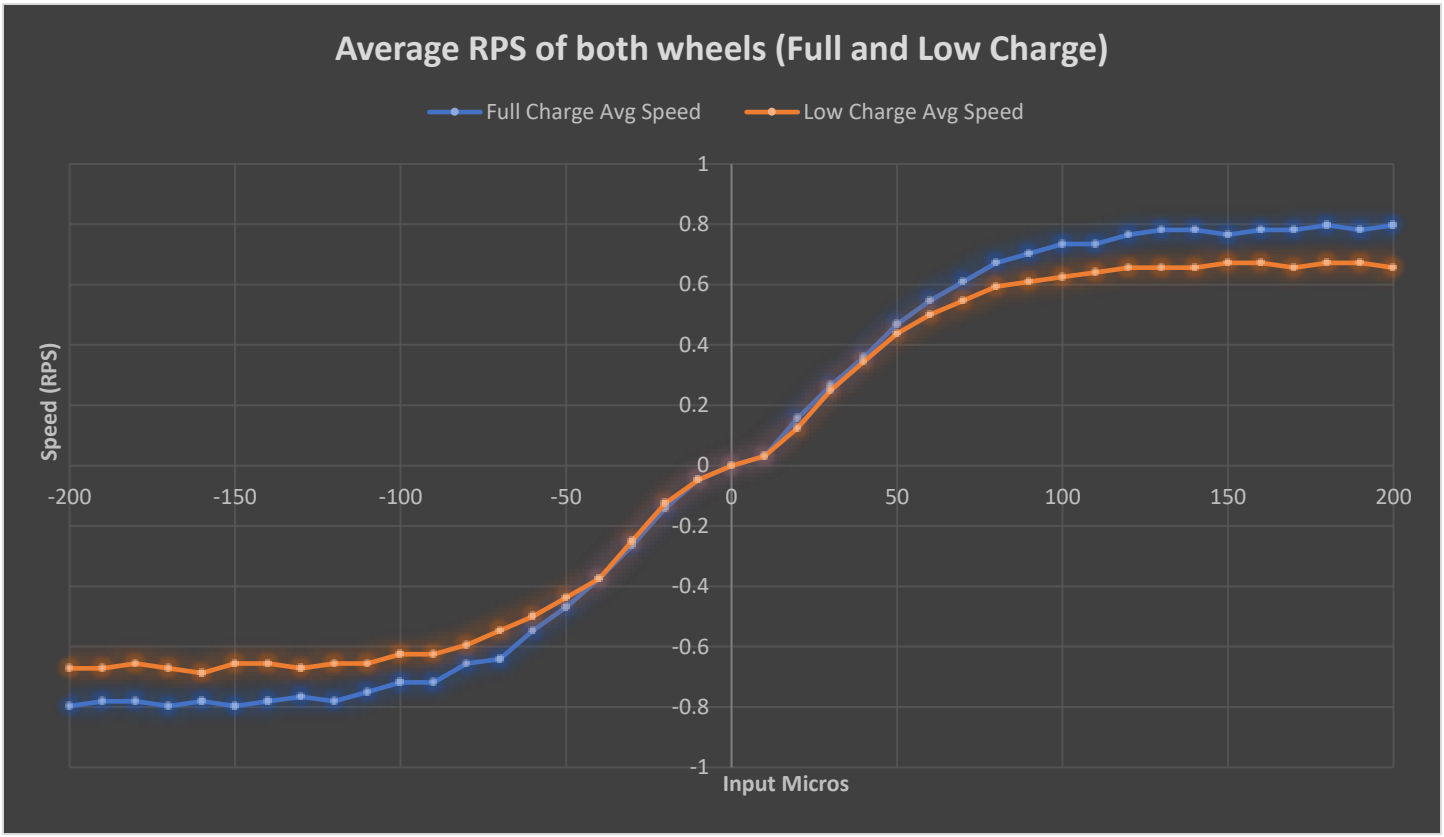
Task 2

With a fully charged set of batteries, in a single graph use the encoders to plot the speed of both wheels (in revolutions per second) vs input to the implemented function setSpeeds. Measure the speeds of the wheels from -200 micros to 200 micros in increments of 10. Repeat the experiment with low batteries. Provide both plots in the report. Things to think about:





Speed Table		Fully Charged (RPS)			10 min charge from 0% (RPS)		
i	Input Micros	Left Speed	Right Speed	Avg Speed	Left Speed	Right Speed	Avg Speed
0	-200	-0.7813	-0.8125	-0.7969	-0.6563	-0.6875	-0.6719
1	-190	-0.7813	-0.7813	-0.7813	-0.6563	-0.6875	-0.6719
2	-180	-0.7813	-0.7813	-0.7813	-0.6563	-0.6563	-0.6563
3	-170	-0.7813	-0.8125	-0.7969	-0.6563	-0.6875	-0.6719
4	-160	-0.7813	-0.7813	-0.7813	-0.6875	-0.6875	-0.6875
5	-150	-0.8125	-0.7813	-0.7969	-0.6563	-0.6563	-0.6563
6	-140	-0.7813	-0.7813	-0.7813	-0.6563	-0.6563	-0.6563
7	-130	-0.7500	-0.7813	-0.7657	-0.6875	-0.6563	-0.6719
8	-120	-0.7813	-0.7813	-0.7813	-0.6563	-0.6563	-0.6563
9	-110	-0.7500	-0.7500	-0.7500	-0.6563	-0.6563	-0.6563
10	-100	-0.7188	-0.7188	-0.7188	-0.6250	-0.6250	-0.6250
11	-90	-0.7188	-0.7188	-0.7188	-0.6250	-0.6250	-0.6250
12	-80	-0.6563	-0.6563	-0.6563	-0.5938	-0.5938	-0.5938
13	-70	-0.6563	-0.6250	-0.6407	-0.5313	-0.5625	-0.5469
14	-60	-0.5625	-0.5313	-0.5469	-0.5313	-0.4688	-0.5001
15	-50	-0.4688	-0.4688	-0.4688	-0.4375	-0.4375	-0.4375
16	-40	-0.3750	-0.3750	-0.3750	-0.3750	-0.3750	-0.3750
17	-30	-0.2813	-0.2500	-0.2657	-0.2500	-0.2500	-0.2500
18	-20	-0.1562	-0.1250	-0.1406	-0.1250	-0.1250	-0.1250
19	-10	-0.0313	-0.0625	-0.0469	-0.0625	-0.0313	-0.0469
20	0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
21	10	0.0313	0.0313	0.0313	0.0313	0.0313	0.0313
22	20	0.1562	0.1562	0.1562	0.1250	0.1250	0.1250
23	30	0.2500	0.2813	0.2657	0.2500	0.2500	0.2500
24	40	0.3750	0.3438	0.3594	0.3438	0.3438	0.3438
25	50	0.4375	0.5000	0.4688	0.4063	0.4688	0.4376
26	60	0.5313	0.5625	0.5469	0.5000	0.5000	0.5000
27	70	0.5938	0.6250	0.6094	0.5313	0.5625	0.5469
28	80	0.6563	0.6875	0.6719	0.5938	0.5938	0.5938
29	90	0.6875	0.7188	0.7032	0.5938	0.6250	0.6094
30	100	0.7188	0.7500	0.7344	0.6250	0.6250	0.6250
31	110	0.7188	0.7500	0.7344	0.6250	0.6563	0.6407
32	120	0.7500	0.7813	0.7657	0.6563	0.6563	0.6563
33	130	0.7813	0.7813	0.7813	0.6250	0.6875	0.6563
34	140	0.7813	0.7813	0.7813	0.6563	0.6563	0.6563
35	150	0.7500	0.7813	0.7657	0.6563	0.6875	0.6719
36	160	0.7500	0.8125	0.7813	0.6563	0.6875	0.6719
37	170	0.7813	0.7813	0.7813	0.6563	0.6563	0.6563
38	180	0.7813	0.8125	0.7969	0.6563	0.6875	0.6719
39	190	0.7813	0.7813	0.7813	0.6563	0.6875	0.6719
40	200	0.7813	0.8125	0.7969	0.6563	0.6563	0.6563



1) 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?
   a) At each charge, the left and right servos generally have very similar outputs, but not that this is because the test is done when hovering. Even after calibration and once on the ground, our robot seems to move properly in the forward direction while it veers right in reverse.
   b) There are some points at the max RPS in which the RPS values seems to oscillate between two numbers. As stated earlier, this is likely due to error in encoder due to similar values.
   c) Our test was not done while the robot was moving on the ground, but possible differences (while moving) between left and right wheel RPS could be the result of inaccurate encoder, servos, and how the servos may react differently due to the Servo.writeMicroseconds() command.
   d) The wheels on our robot (and the metal frame itself) look slightly bent which would also cause differences in output encoder/speed values.
   e) Software calibration can help immensely with making sure that servos and encoders match up, but in our case due to lack of both precision and accuracy in the servos it becomes increasingly hard to make the robot do what we want it to do.
   f) Say for instance, you may get a wheel alignment for your car. Even if the alignment is within .1 degrees, there could be multiple reasons as to why the car won't drive straight when you aren't holding the steering wheel (don't do this obviously). The road could be sloped for drainage, tires may have different wear and thus slightly different circumference, etc.

Task 3

Implement the program 'Forward.ino'. 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 on "ForwardParams.h". 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 can't complete the movement in the given amount of time, when pressing select, instead of moving forward, the robot should display an appropriate message on the LCD. During the task presentation, the TA will assign different values for "X" and "Y".

Task 3 Calculations:

Given PARAM_X (inches) and PARAM_Y (seconds):

Inches per second (IPS) at which robot should travel at = PARAM_X/PARAM_Y

Maximum IPS = Maximum RPS * CIRCUMFERENCE =

= 0.8 * 8.194 = 6.55 inches/sec

Since 32 ticks = 1 revolutions

= 2 * PI * R = 8.2"

Amount of ticks required for any PARAM_X = 32 * PARAM_X / 8.2

Task 4

Implement the program 'SShape.ino'. The program should make the robot move in two semicircles of radius "R1" and "R2" respectively, as shown in Figure 2. The robot should stay still until the "select" button is pressed. When "select" is pressed, the robot should perform the first semicircle in a clockwise movement. After completing the first semicircle, the robot should wait for the "select" button to be pressed again, 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 on "SShapeParams.h". The program must work for any values of "R1", "R2" and "Y". If the motion cannot be completed in the given time, after pressing "select" for the first time, the robot should display an appropriate message on the LCD.

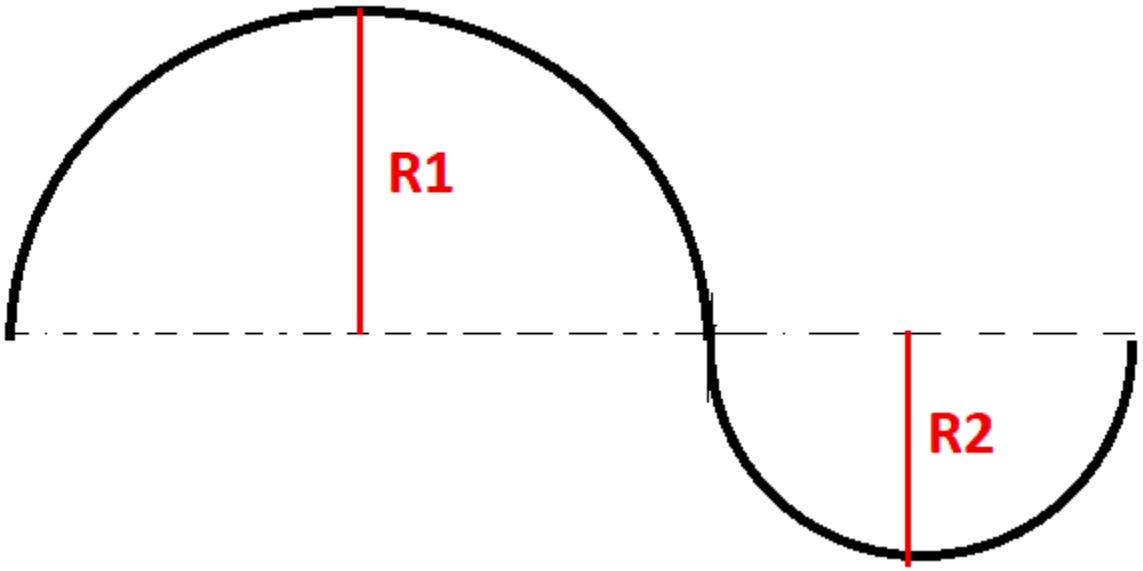


Figure 2. Path to be followed by the robot.

We are lowering some of the requirements for the s shape. Both radii will have the same length. Also, since the basic method of calibration (what we expect you to do) has some performance issues, even after calibration you won't be able to perform all possible circles. Thus, I'm giving some values for which your robot should perform well:

Radius	Time
5	7.64
6	8.73
7	9.82
8	10.92
9	12.01
10	13.11

Task 4 Calculations:

Given PARAM_R1 (inches) and PARAM_Y (seconds):

d (Distance between wheels) = 3.95 inches

Angular velocity = 2π / PARAM_Y

Linear speed = v

= Angular speed * PARAM_R1

= (2π / PARAM_Y) * PARAM_R1

To obtain angular movement, two different velocities for each wheel are required: V_L and V_R, where:

V_L = Angular Speed * (PARAM_1 + d/2)

= (2π / PARAM_Y) * (PARAM_1 + d/2)

V_R = Angular Speed * (PARAM_1 - d/2)

= (2π / PARAM_Y) * (PARAM_1 - d/2)

1) 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 2). Taking this into consideration, can the robot move in any kind of trajectory? Consider an ellipse as an example.
   a) Yes, the robot can move in any kind of trajectory. Based on the value of k, which indicates how much the curve is "bending", the robot can move by fixing the speeds of the wheels for a given amount of time in order to follow the curve. Looking at an ellipse, given the formula, the curvature can indeed be found (See Task 5 Mathematical equation). Plugging in any value for a and b, the curvature at any point of the ellipse (x,y) can be found. However, the exact trajectory may not be covered due to imperfect servo motors and other variables such as floor friction, which can result in slippage. Given the right conditions, the robot should able to maneuver certain trajectories.

Task 5

Implement the program “Ellipse.ino”. The program should make the robot move in an ellipse given by the formula

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

(as seen in Figure 3). The robot should stay still until the “select” button is pressed. When the button is pressed, the robot should move on an ellipse at constant linear speed “V”. The parameters “a”, “b”, and “V” will be provided on “EllipseParams.h”. The program should work for any assigned values. On this part, you do not need to worry on whether the robot can traverse the ellipse at the given speed, but you must include in the report what limits the possible values of “V”. The robot should start on the leftmost point of the ellipse, corresponding to position (-a, 0) facing the positive y-axis direction. The robot should move in clockwise manner.

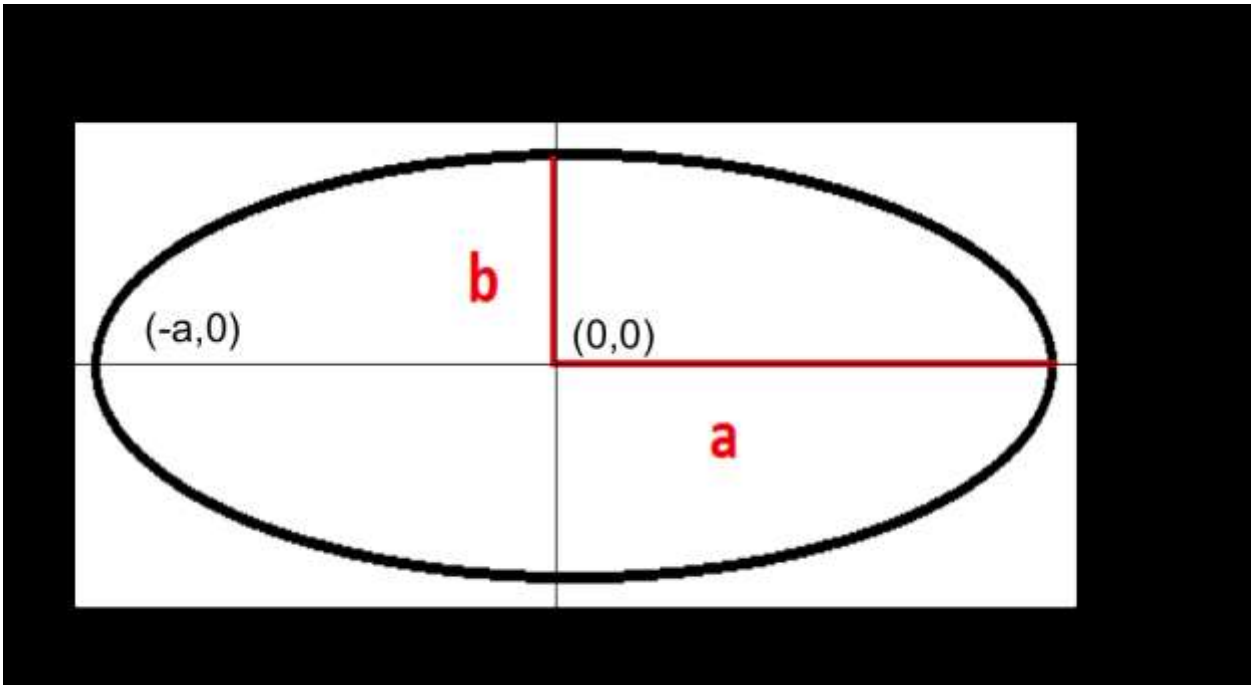


Figure 3. Path to be followed by the robot.

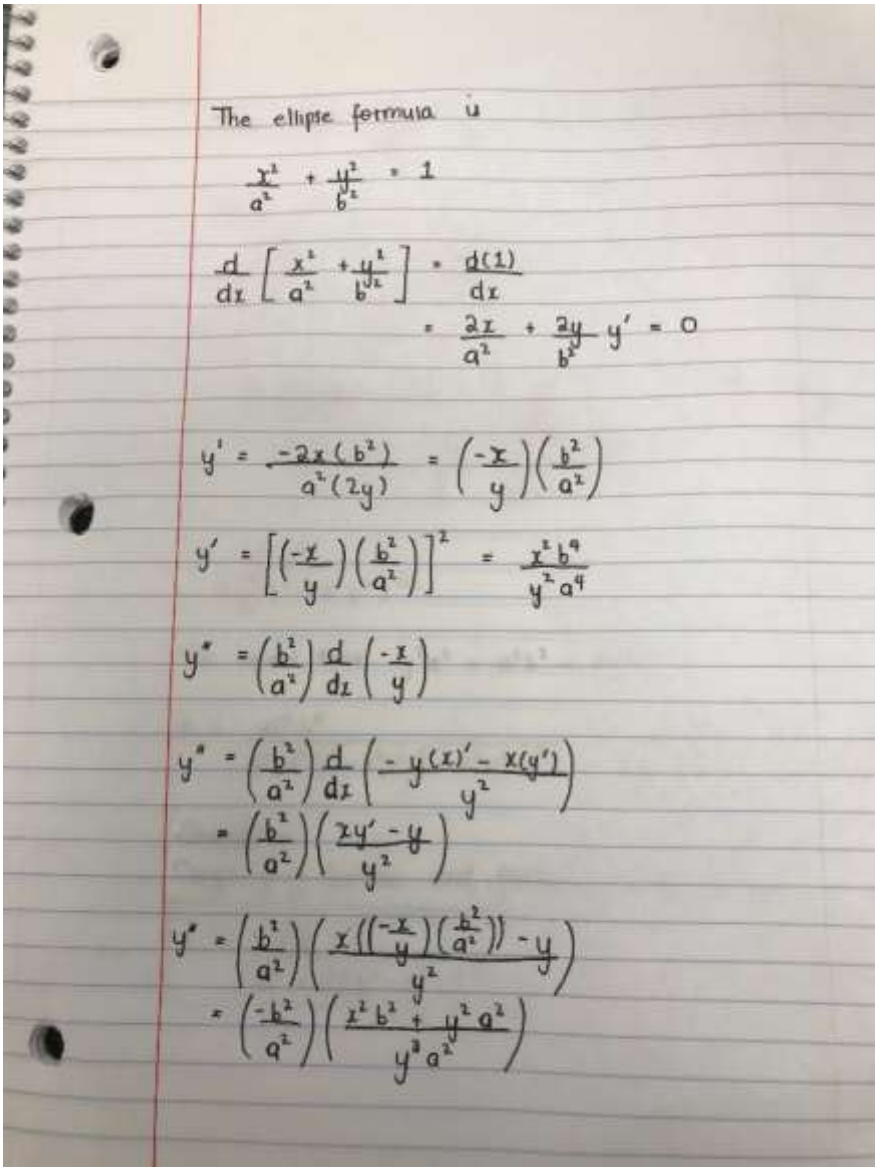
During the task presentation, the TA will assign different values to “a”, “b” and “v”. Hints and notes:

• The objective of this task is for you to understand the difficulties of doing a non-trivial curve. You should strive to draw the ellipse the best you can but don’t expect results to be perfect.

• Use the material seen in class to calculate curvature of an ellipse as a function of the position. Use the formula $\omega = kv$.

Task 5 Calculations:

Curvature of an Ellipse:



The ellipse formula is

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

$$\frac{d}{dx}\left[\frac{x^2}{a^2} + \frac{y^2}{b^2}\right] = \frac{d(1)}{dx}$$

$$= \frac{2x}{a^2} + \frac{2y}{b^2} y' = 0$$

$$y' = \frac{-2x(b^2)}{a^2(2y)} = \left(\frac{-x}{y}\right)\left(\frac{b^2}{a^2}\right)$$

$$y' = \left[\left(\frac{-x}{y}\right)\left(\frac{b^2}{a^2}\right)\right]^2 = \frac{x^2 b^4}{y^2 a^4}$$

$$y'' = \left(\frac{b^2}{a^2}\right)\frac{d}{dx}\left(\frac{-x}{y}\right)$$

$$y'' = \left(\frac{b^2}{a^2}\right)\frac{d}{dx}\left(\frac{-y(x)' - x(y')}{y^2}\right)$$

$$= \left(\frac{b^2}{a^2}\right)\left(\frac{xy' - y}{y^2}\right)$$

$$y'' = \left(\frac{b^2}{a^2}\right)\left(\frac{x\left(\left(\frac{-x}{y}\right)\left(\frac{b^2}{a^2}\right)\right) - y}{y^2}\right)$$

$$= \left(\frac{-b^2}{a^2}\right)\left(\frac{x^2 b^2 + y^2 a^2}{y^3 a^2}\right)$$

Given that $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 = x^2b^2 + y^2a^2 = a^2b^2$

Then

$$y'' = \left(\frac{-b^2}{a^2}\right)\left(\frac{a^2b^2}{y^3a^2}\right)$$

$$= \frac{-b^4}{y^3a^2}$$

$\therefore$ Curvature value $= \dfrac{\left|\frac{-b^4}{y^3a^2}\right|}{\left[1 + \frac{x^2b^4}{y^2a^4}\right]^{3/2}} = \frac{b^4}{y^3a^2} \cdot \frac{y^3a^6}{(y^2a^4 + x^2b^4)^{3/2}}$

$$= \frac{a^4b^4}{(y^2a^4 + x^2b^4)^{3/2}}$$

We know that $y^2a^2 = a^2b^2 - x^2b^2$

$$K = \frac{a^4b^4}{a^2(a^2b^2 - x^2b^2) + x^2b^4)^{3/2}} = \frac{a^4b^4}{(a^4b^2 - x^2a^2b^2 + x^2b^4)^{3/2}}$$

Simplifying further and factoring

The curvature of an ellipse is :

$$K = \frac{a^4b}{(a^4 + x^2(b^2 - a^2))^{3/2}}$$

C.2 Task Report (25 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 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.
    a) Forward.ino – Main function for task 3
    b) ForwardParams.h – Values for task 3
    c) SShape.ino – Main function for task 4
    d) SShapeParams.h – Values for task 4
    e) Ellipse.ino – Main function for task 5
    f) ***Files are all sorted into folder for easy reading***
2) Equations used to calculate the speed of the wheels in the encoder library, and the equations used in functions setSpeedIPS and setSpeedsvw. A brief description should be provided for each equation.
    a) getSpeeds() from MyEncoders.h
        i) speeds[0] = (1.0 / 32.0) * (1000 / lDelta)
        ii) Divide by number of holes, then multiplied by elapsed time over change in time (delta T)
    b) setSpeedIPS()
        i) float tempLeft = ipsLeft / 8.2;
        ii) circumference = Pi*2.61(diameter) = 8.2;
        iii) Then goes back to call setSpeedRPS for calibrated values
    c) setSpeedsvw()
        i) float R = V / W;
        ii) float d = 3.95;
        iii) float vR = W * (R + (d / 2));
        iv) Obtained from lecture slides, used to get velocity of each wheel (right wheel in this case)

Conclusion

At the start of the project, the main confusions came from how to implement both MyEncoders.h and MyServos.h, since the descriptions for the files were still somewhat vague and working out how to create the functions was a much harder task than anticipated, which led to multiple meetings with our TA Pablo to achieve clarity. Our group worked many hours stuck on single functions such as calibrateSpeeds() and setSpeedsRPS() because overall they were a bit tricky to get working. Looking back, calibrateSpeeds() was not that difficult of a function overall, but we also originally did not know that the function was supposed to automatically calibrate in any situation, and so we spent a good deal of time with manual calibration values and ratios, all of which were eventually discarded. Even after finishing the function

and testing it to see that it works as intended, the bot would still behave erratically at times, with forward movement being fairly aligned, while backwards movement always made the robot veer off to the right. After checking through our calibration again and looking at encoder values, we concluded that the encoders and servos were not accurate or precise enough to get good data, and that even slight differences in those values would cause the bot to move in a completely different direction than intended.

After the header files, Task 1 and 2 were fairly straightforward, and it did not take long to finish those, as Task 2 was mostly what we had in our calibrateSpeeds() function, so we were able to reuse most of that code. However, with Task 3 Forward.ino, we once again faced some pitfalls that made us work on the code for far longer than we intended to. First, we did not realize that the input values of PARAM_X and PARAM_Y were ints, and in trying to find the inches per second values (IPS) we did division but failed to first cast those parameters to float values. That, along with casting the encoder values to floats (for division) forced us to change and debug the code multiple times, only to realize that small errors ended up causing big problems. Additionally, we originally also did not figure out how use the buttons to select the functions, so we had to look through the example codes and do trial and error for that. We did start by stripping the example TotalTest3 code for the chooser, but that specific function chooser ended up being far to complicated and different to what we needed.

SShape didn't work out and we ran out of time.