Sara Savitz – U37713110
Alan Rodriguez – U86831061 CDA4621.001F17

Control of Mobile Robotics – Lab 1 Report

Section I: List of Files

```
MyEncoders.h

MyServos.h

ForwardParams.h

SShapeParams.h

task1.ino     // Calculate Instantaneous Speed of Right Servo

task2.ino     // Measure/Calibrate Speed with Full/Drained Batteries

task3.ino     // Implement Forward Function

task4.ino     // Implement S-Shape Function
```

Section II: Speed Equations and Calculations

For function **getSpeeds()**:

```
                    LdeltaT = now - timeL;
                    RdeltaT = now - timeR;

      speeds[0] = (float)Lticks/(32.0f*LdeltaT) * 1000.0f;
      speeds[1] = (float)Rticks/(32.0f*RdeltaT) * 1000.0f;
```

Using the millis() function, we can record a change in time (deltaT) between subsequent calls of getSpeeds(), which we calculate by subtracting the last recorded time from the current time.

To determine speed, we divide the left and right tick counts by 32 to determine the number of revolutions. We further divide this number by their respectful deltaT to yield a velocity in revolutions per milliseconds. Finally, to convert to revolutions per second, multiply the value by 1000. Store the value to the speeds array.

Note: An interrupt is issued every time the encoders sense an opening in the wheel. Lticks and Rticks will increment when interrupts occurs.

For function **setSpeedsIPS(float ipsLeft, float ipsRight):**

```
                  float ipr = 8.1995f;
                   float tmpL, tmpR;
                  tmpL = ipsLeft/ipr;
```

```
tmpR = ipsRight/ipr;

setSpeedsRPS(tmpL, tmpR);
```

ipr represents the number of inches per revolution of the robot's wheel. By diving the input parameters ipsLeft and ipsRight by ipr, this yields the speed in revolutions per second. Passing these two modified parameters into setSpeedsRPS will use the calibrated data to obtain the average speed in rotations per second. The data is further converted to a range within [-100, 100] and passed into the function setSpeeds.

For function **setSpeedsvw(float V, float w):**

We must first check the direction of the angular velocity to determine if the robot will move clockwise or counterclockwise. If **w** is less than zero, the robot will move clockwise; if it is greater than zero, the robot will move counterclockwise. Since the robot is moving in a circle, the linear velocity of each servo will vary.

If w < 0:

```
R = v / -w;
vL = -w * (R+dr);
vR = -w * (R-dr);
```

We obtain the radius by dividing v by w, being sure to invert w so our radius is not negative. We must adjust the radius so the left wheel is traveling faster than the right wheel. We do this by adding/subtracting the given R to the distance from one servo to the center of the robot. Using these values, we can calculate two distinct velocities for each servo that will allow us to traverse a circle of radius R.

If w > 0:

```
R = v / w;
vL = w * (R-dr);
vR = w * (R+dr);
```

In this case, the robot must travel counterclockwise since w is greater than zero. This time we do not need to account for the sign on w. After calculating R, we must adjust the radius so the left wheel travels slower than the right wheel.

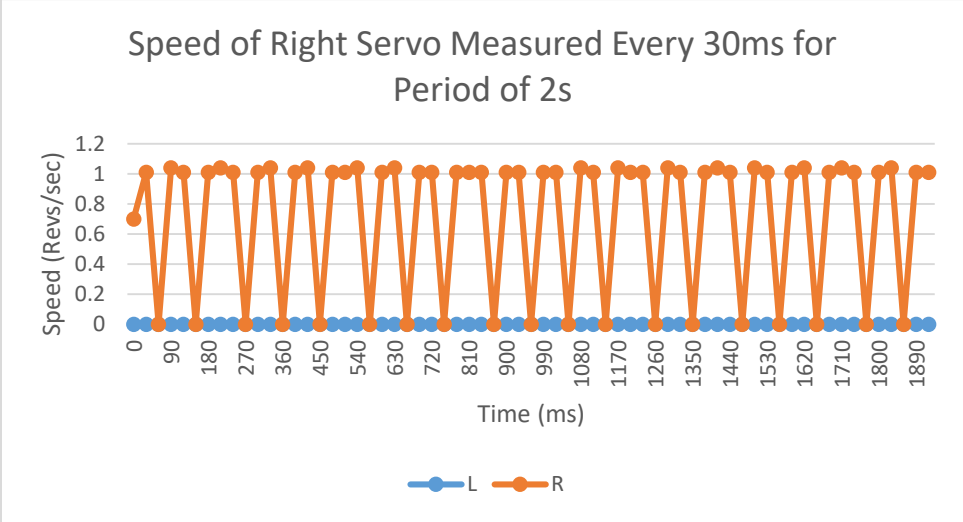
In either case, we obtain two distinct values of vL and vR that are expressed in inches per second. We can pass these two values into the function setSpeedsIPS to obtain the desired speed of the robot.

Sara Savitz – U37713110
Alan Rodriguez – U86831061 CDA4621.001F17

Section III: Tasks 1 and 2

Task 1:

The following chart plots the instantenous speed of the right servo measured every 30ms:

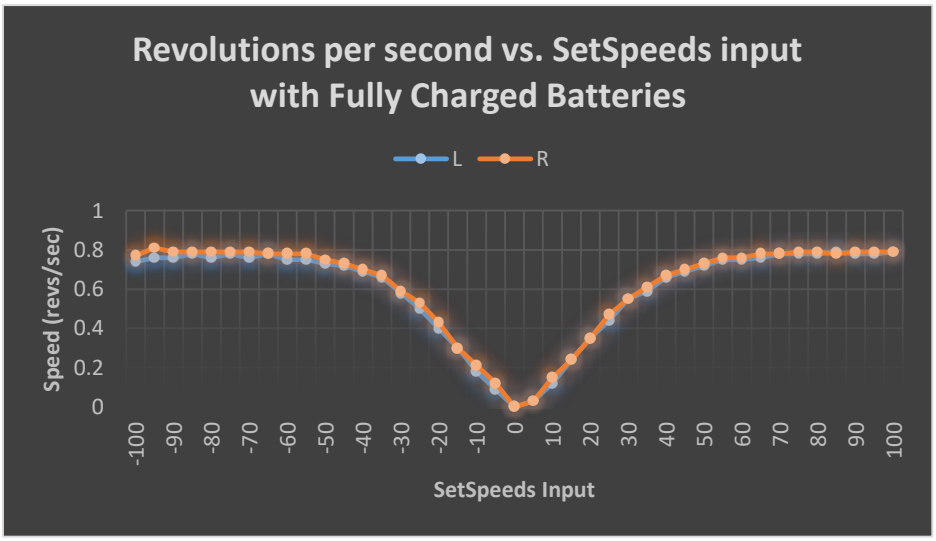


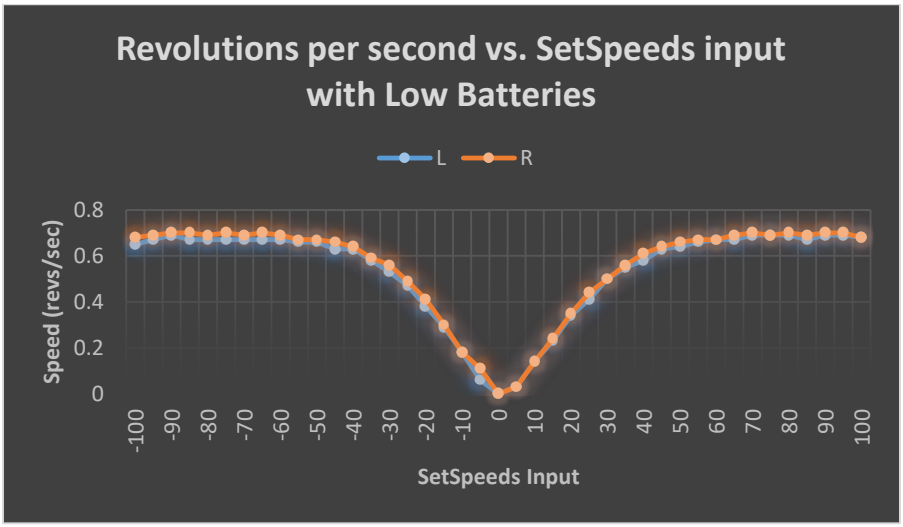
As seen in the chart, the measured values spike between a large speed and zero every 90ms. This is most likely because the speed is being measured too frequently. Since the speed is determined by the number of ticks counted by the encoders over a certain period of time, we must ensure that the time interval between each measurement is large enough for an interrupt to occur. If no interrupt occurs, the time calculated is zero, resulting in an undefined speed.

Due to physical harware limitations, the encoders can only read every 39 ms. Trying to read the speeds every 30ms isn't enough time for an interrupt to occur therefore explains the random zeros plotted in the graph.

Task 2:

The following charts are measurements of speed for the left and right wheel when the batteries are both fully charged and drained:





Section IV: Mathematical Solutions for Task 3 and 4

Task 3 Mathematical Solution – Forward Movement:

The Forward Movement task takes in parameters of X and Y, which are the inches the robot should travel and the time it should take to do it, respectively. In order to obtain the speed the robot must go in inches per second, we must divide x (inches) by y (time).

```
speedIPS = (float)PARAM_X / (float)PARAM_Y;
```

If the speed exceeds 6.5 inches/sec (maximum speedIPS of robot), the request will display as invalid on the LCD. Otherwise, the function will call resetCounts and set the speed of both servos to speedIPS using the setSpeedsIPS function.

In order to stop the robot once it has traveled the specified distance in the specified time interval, we must calculate the maximum number of tick counts that will occur and keep track of the tick counts as it moves.

```
maxTicks = (unsigned long)((PARAM_X / 8.2f) * 32);
```

By dividing the number of inches, X, by 8.2 (the number of inches per rotation), we can determine how many rotations it will take to travel X inches. Multiplying this number by 32 will yield the maximum number of tick counts counted over a distance of X inches.

Using getCounts, we can compare the current tick value to the max tick value; once the current value exceeds the max value, we set speeds to zero.

Task 4 Mathematical Solution – S Shape Movement:

This function takes in parameters of R and Y, where R is the radius of a circle and Y is a time in seconds. When the select button on the Arduino is pressed, the robot must travel clockwise along a circle with radius R, over the course of Y seconds. It will then stop once completed, and wait for the select button to be pressed again. Once pressed, it will perform the same movement, but in a counterclokwise motion, subsequently performing an S-Shape traversal.

In order to know when to stop at the halfway mark, we must once again keep track of the tick counts of each wheel. Since the robot is traveling in a circle, the tick counts will be different from one another, and therefore must be tracked indepedently.

The following equations calculates the max number of ticks for each wheel, given that the robot is moving in a clockwise motion:

```
r_outer = (float)PARAM_R1 + r_axle;
r_inner = (float)PARAM_R1 - r_axle;
```

First calculate the modified radius for the outer and inner wheel, whee **r_axle** is the distance from the center of of the axle to one servo.

```
s_outer = (3.14159f)*(r_outer);
s_inner = (3.14159f)*(r_inner);
```

Use this radius to calculate the arc length that each wheel must travel.

```
ticksL = (unsigned long)((32.0f * s_outer)/(2.0f * 3.14159f *
                         r_wheel));
```

```
ticksR = (unsigned long)((32.0f * s_inner)/(2.0f * 3.14159f *
                         r_wheel));
```

To calculate the max tick counts of each wheel, divide the arc length by the circumfrence of each wheel (r_wheel) to obtain the max number of rotations. Multiply this value by 32 to obtain the max number of tick counts for each wheel.

```
V = (float)(PARAM_R1 * 3.14159f)/(float)(PARAM_Y);

w = V / (float)PARAM_R1;
```

We then use R and Y to calculate the velocity in inches per second. Dividing the velocity by R gives us the angular velocity. We now have the two parameters we need to pass into the setSpeedsvw function.

Using getCounts, we can compare our max tick counts of each wheel with the current tick counts. When the current exceeds the max, we set the speeds to zero.

The robot will then wait for the select button to be pressed again. Once pressed, the robot will perform the same action, but counterclockwise. This is done by swapping the values of ticksL and ticksR, as shown below:

```
ticksL = (unsigned long)((32.0f * s_inner)/(2.0f * 3.14159f *
                         r_wheel));
ticksR = (unsigned long)((32.0f * s_outer)/(2.0f * 3.14159f *
                         r_wheel));
```

Once the max tick values are calculated, we can now pass in our V and w to the setSpeedsvw function, but this time, w (angular velocity) must be negative; doing so will make the robot move counterclockwise.

As before, we use getSpeeds to compare the current tick counts to the max tick counts, and set the speed to zero once current exceeds max.

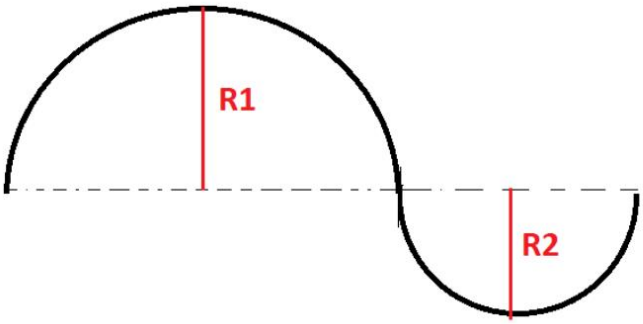


Figure 2. Path to be followed by the robot.

- 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 2). Taking this into consideration, can the robot move in any kind of trajectory? Consider an ellipse as an example.

Yes, the robot can move in any kind of trajectory. Using polar coordinates, a path of trajectory can revolve around any given position with respect to theta. A center of curvature can be calculated by knowing a radial distance and angular velocity.

Section V: Conclusion

During this lab, we experienced a variety of issues that greatly deepened the complexity of each task requested. One major problem we had to take into account consistently was the crookedness of one of the wheels on our robot. This affected the readings of the encoder and impacted the accuracy of our speed calculations throughout each task. This issue proved most disruptive during our implementation of the S-Shape function, which relied heavily on accurate readings of the ticks provided by each encoder to decide when to stop.

Other challenges we encountered included tasks such as checking for proper input for each setSpeeds function. In many cases, input provided in the header files were not feasible by the robot. When the robot operated at slower speeds, we experienced less accuracy in our S-Shape parameter function. We believe that this was due to less frequent encoder ticks.

Also the servos motors are very sensitive and it is difficult to adjust and calibrate as needed.