Control of Mobile Robotics
Lab 1 Report

Jason Nguyen

Weiwei Su

February 5, 2018

1) Requested Code:

- Servos.cpp: Contains functions to calibrate and set speeds to servos (wheels).

- Servos.h: Header file for Servos.cpp. Includes function prototypes and variables viewable to other modules.

- Encoders.cpp: Contains encoder functions to measure speed by counting the amount of ticks that have passed in the Servos.

- Encoders.cpp: Header file for Encoders.cpp. Includes function prototypes and variables viewable to other modules.

- Task1.ino: Contains test function for Task 1.

- Task2.ino: Contains test function for Task 2.

- Forward.ino: Contains test function for Task 3, which is to make the robot move in a straight path with given distance and time.

- ForwardParams.h: File that defines the parameters for Task 3 (Forward.ino).

-SShape.ino: Contains test function for Task 4, which is to make the robot move in two semi circles with a given radius and time.

-SShapeParams.h: File that defines the parameters for Task 4 (SShape.ino).

2) Equations for Encoder and Servos library:

- In the function getSpeeds, we used the equation $\frac{\frac{1}{\#\ of\ holes}}{time}$ to calculate the instantaneous speeds in RPS. The following two codes were implemented:

```
leftRPS = ((float)(1.00/32.00)) / ((leftLatestTime - leftPrevTime)) * 1000;
rightRPS = ((float)(1.00/32.00)) / ((rightLatestTime - rightPrevTime)) * 1000;
```

32 represents the total holes in each wheel. $LatestTime - PrevTime$ represents the time it takes for one hole to pass in milliseconds, which is why we multiply by 1000.

- setSpeedsIPS() equations:

During calibration, several values of speeds (writeMircoseconds) are tested and converted into IPS. They are then stored into an array for later use. These IPS values are calculated by:

```
LeftIPS[i] = (float)leftTickCount * (((2.61 * 3.14)/32.00) / 5.00);
RightIPS[i] = (float)rightTickCount * (((2.61 * 3.14)/32.00) / 5.00);
```

The next equation is used to find the speed needed to match the IPS arguments passed into the function.

```
else if(LeftIPS[i] > ipsLeft){
  left_IPS = (5 * i) + (5* (ipsLeft - LeftIPS[i-1])/(LeftIPS[i-1] + LeftIPS[i]));
  break;
}
```

```
else if(RightIPS[j] > ipsRight){
  right_IPS = (5 * j) + (5* (ipsRight - RightIPS[j-1])/(RightIPS[j-1] + RightIPS[j]));
  break;
}
```

When the for loop finds the first value that is greater than the argument, we use that index to calculate the speed. $5 * i$ is used to calculate the base speed, while the rest of the equation is used to determine whether we need to round up or down.

- setSpeedsvw() equations:

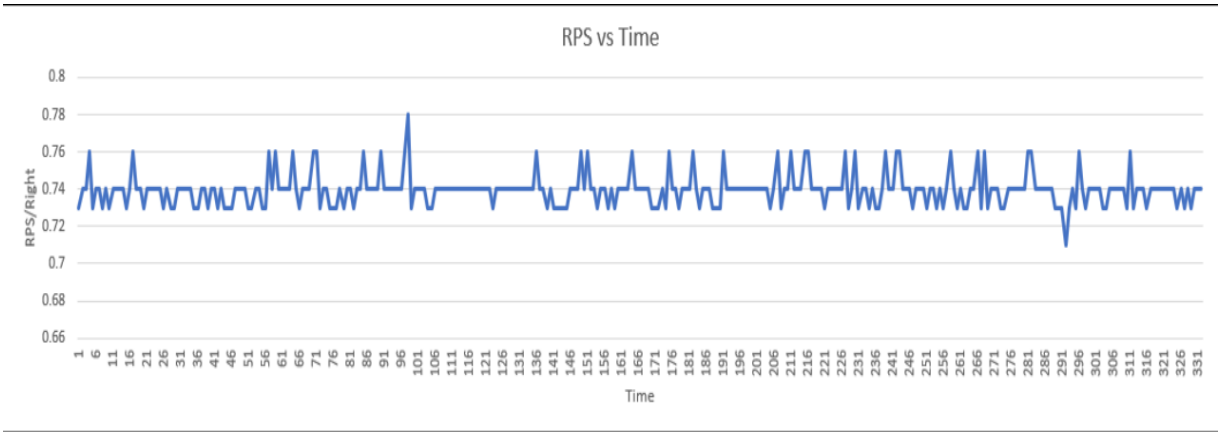
```
else if(w > 0){
  float R = v / w;
  left = w * (R - (3.95/2.00));
  right = w * (R + (3.95 / 2.00));
```

```
else if(w < 0){
  float R = v / w;
  left = (-1)* w * (((-1) * R) + (3.95 / 2.00));
  right = (-1)* w * (((-1) * R) - (3.95 / 2.00));
```

The base equation used to calculate the speed of each wheel is $V_{wheel} = w * (R \pm d_{axis})$. We add or subtract d_{axis} depending on the position of the wheel. If it is the wheel closer to the center of the circle, we subtract and vice versa. When angular velocity is positive, we want it to spin counter clock wise, and the opposite for a negative angular velocity. Therefore, we need two different equations for each scenario.

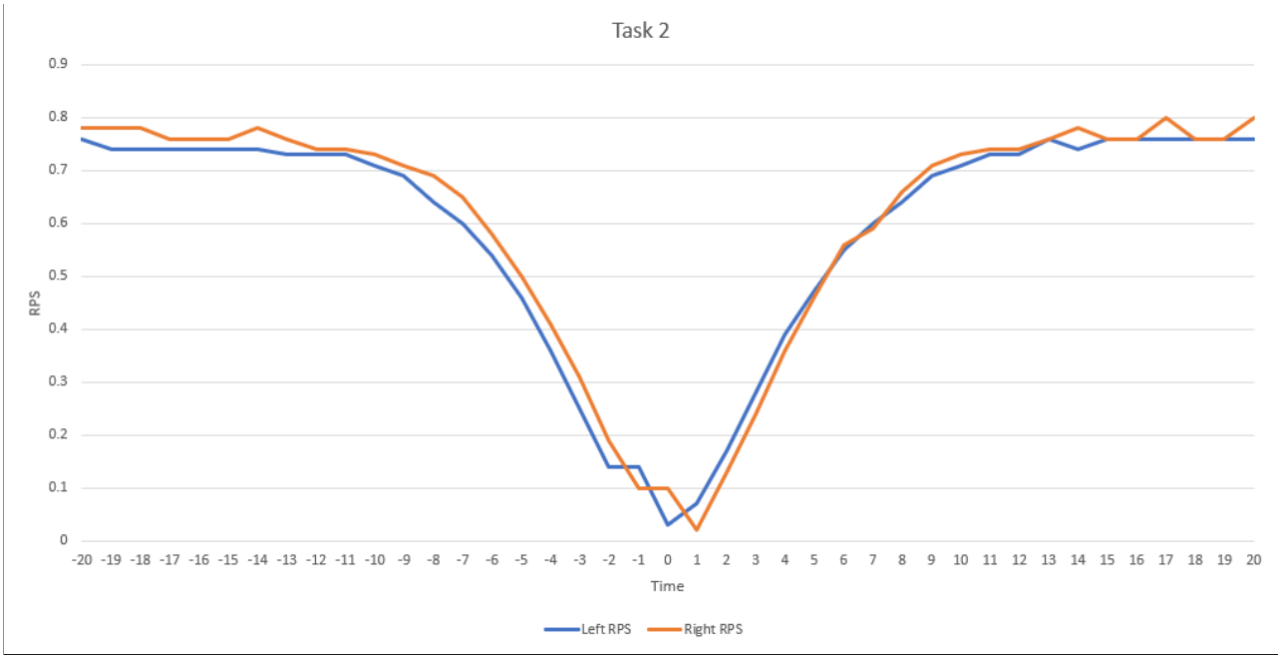
3) Plots for section B4.

Task 1:



This graph is measuring the RPS of the right wheel vs time. The value is not constant; however, it does show consistency. It seems to stick around the mid .7's for a majority of its movement. Factors like the servo itself and the surface that it is moving on could all influence the waveform.

Task 2:



This graph measures and compares the RPS of both wheels vs time. Values are fairly close the whole movement. Again factors like the surface could have had an effect on the RPS difference.

4) Kinematics

Task 3:

For task 3, we were given a time and a distance. We can calculate the velocity needed with $v = \frac{distance}{time}$. Implemented as,

```
float checkSpeed = PARAM_X/PARAM_Y;
```

Once calibrated, the highest possible speed should be stored for each wheel. If our checkSpeed is greater than either value, the robot cannot complete the movement in the given time.

```
if(checkSpeed > LeftIPS[20] || checkSpeed > RightIPS[20]){
  fwlcd.clear();
  fwlcd.setCursor(0,0);
  fwlcd.print("FORWARD");
  fwlcd.setCursor(0,1);
  fwlcd.print("ERROR-PARAM HIGH");
```

Because we don't want the robot to move backwards, we also output an error message if checkSpeed is negative.

```
}else if(checkSpeed < 0){
  fwlcd.clear();
  fwlcd.setCursor(0,0);
  fwlcd.print("FORWARD");
  fwlcd.setCursor(0,1);
  fwlcd.print("ERROR-NO REVERSE");
```

Otherwise, we can just use checkSpeed as our parameter for setSpeedsIPS. This value should make the robot complete the run within the parameters.

For the next part we needed to set a section that made the robot stop once it hits X distance. We can use the distance formula using ticks to measure the distance of each wheel. Once one of the wheels reach distance X, we set the speeds of the wheels to 0. The formulas used to calculate the distance are:

```
int leftDis = leftTick * ((2.61*3.14)/32);
int rightDis = rightTick * ((2.61*3.14)/32);
```

Then we use an if statement to check when distance X is reached.

```
if(leftDis >= PARAM_X || rightDis >= PARAM_X){
  fwlcd.clear();
  fwlcd.setCursor(0,0);
  fwlcd.print("FORWARD");
  fwlcd.setCursor(0,1);
  fwlcd.print("COMPLETED");
  setSpeeds(0,0);
  initEncoders();
}
```

Task 4:

For task 4, we need to make the robot move in 2 semi circles given 2 radii's and a time.

First thing is to calculate the linear velocity. Because we want a constant linear velocity throughout the whole run, we take the distance of both semi circles and divide that by the required time.

```
float linearV = ((3.14 * Radius1)+(3.14*Radius2))/PARAM_Y;
```

Because we have two different semi-circles, we need to find two different angular velocities. But first we need to calculate the time needed for each circle. We can use a ratio to determine how much time is needed. For example, if the first radius is 20 and the second is 10. The first semi-circle is going to need $\frac{2}{3}$ of the total time.

```
float totalDis = firstRadius + secondRadius;

R1Time = (firstRadius / totalDis) * totalTime;
R2Time = (secondRadius / totalDis) * totalTime;

reqTime[0] = R1Time;
reqTime[1] = R2Time;
```

Next is to calculate both angular velocities. Because $w = \frac{v}{r}$, we substitute $v = \frac{distance}{time}$ into w. Distance of a semi-circle is πr. So, our equation for velocity ends up being $w = \frac{\pi}{\frac{t}{2}}$. If we want the robot to move clockwise, we multiply by -1.

```
float angularV1 = (-1)*(3.14/(RequiredTime[0]/2)); //first angularV (Clockwise)
float angularV2 = 3.14/(RequiredTime[1]/2); //second angularV (Counter-Clockwise)
```

We use the same method as we did in task 3 to determine whether the length can be traveled in the given time.

```
if((linearV > LeftIPS[20] && linearV > RightIPS[20])){
  sslcd.clear();
  sslcd.setCursor(0,0);
  sslcd.print("S-SHAPE");
  sslcd.setCursor(0,1);
  sslcd.print("ERROR-PARAM HIGH");
}else if((linearV < 0) || (linearV < 0)){
  sslcd.clear();
  sslcd.setCursor(0,0);
  sslcd.print("S-SHAPE");
  sslcd.setCursor(0,1);
  sslcd.print("ERROR-NO REVERSE");
```

The same concept is transferred over from task 3 to find the stopping point. We measure the distance the robot needs to travel and then measure the distance traversed by the wheels using the encoders. When the distance traversed reaches the distance of the semi-circle, the robot will stop and wait for further inputs.

```
float wheelR = (((leftTick*(2.61*3.14)/32)+(rightTick*(2.61*3.14)/32))/ 2.00);
if((wheelR>firstHalf) && flag == false){
       setSpeeds(0,0);
       flag = true;
       initEncoders();
       sslcd.setCursor(0,0);
       sslcd.print(erraseString);
       sslcd.setCursor(0,1);
       sslcd.print(erraseString);
       sslcd.setCursor(0,0);
       sslcd.print("S-SHAPE -SH");
       sslcd.setCursor(0,1);
       sslcd.print("PRESS SLT TO RUN");
 }
if((wheelR>secondHalf) && flag == true){
   flag == false;
   sslcd.clear();
   sslcd.setCursor(0,0);
   sslcd.print("S-SHAPE");
   sslcd.setCursor(0,1);
   sslcd.print("COMPLETED");
   setSpeeds(0,0);
   initEncoders();
 }
```

For the comparison, we took the average of both wheels. Once the first if statement runs, it sets the flag to TRUE, so it can run the second semi-circle.

Task 5:

The curvature of an ellipse is defined by $\kappa = \frac{|T'(t)|}{|\alpha'(t)|}$.

For an ellipse, $\alpha'(t) =< -a\sin t, b\cos t >$ and $\alpha''^{(t)} =< -a\cos t, -b\sin t >$.

Thus, $|\alpha'(t)| = \sqrt{a^2\sin^2 t + b^2\cos^2 t}$.

$$T(t) = \frac{\alpha'(t)}{|\alpha'(t)|} =< -\frac{a\sin t}{\sqrt{a^2\sin^2 t + b^2\cos^2 t}}, \frac{b\cos t}{\sqrt{a^2\sin^2 t + b^2\cos^2 t}} >$$

$$\kappa = \frac{|T'(t)|}{|\alpha'(t)|} = \frac{ab}{\left(\sqrt{a^2\sin^2 t + b^2\cos^2 t}\right)^3}$$

5) Videos

LINKS GO HERE

https://youtu.be/lXIppvnIacQ

6) Conclusion

Lab 1 introduced us to the fundamentals of Arduino, servos, encoders, and kinematics and how to apply it to make the robot complete certain tasks. Learning the syntax and code for the Arduino was simply. Applying it and learning to wrap our head around what the question was asking was of the bigger challenges. Over time, we started to realize that most of the time is used testing the robot rather than programming the robot. It was obvious that parts to our robot were worn out. The wheels have lost all its treading and were very scratched up, which gave us major problems later during the SShape task. When testing the first semi-circle, our robot was doing a circle with about a 40 in radius. At first, we thought it was the fault of our code, soon finding out the problem was the wheels. The wheels were not gripping onto the tile and it was causing the wheels to slip. We had to re locate to an area with a surface with more friction. Even with these problems, the process was still enjoyable. We have learned so much about programming in Arduino and the several functions used to make the robot perform tasks. As tedious as the robot was, the overall lab was very much fun!