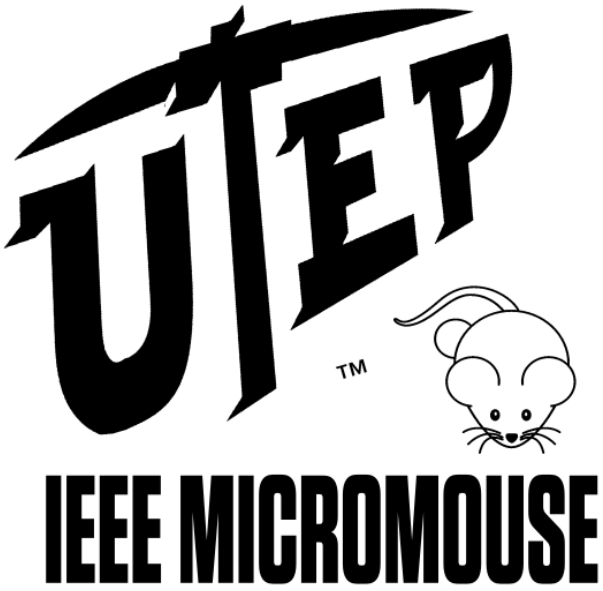

IEEE MICROMOUSE

Workshop Week 7:

Micromouse distance control

Objective:

Traverse one cell distance accurately. Use classical control theory to achieve distance control. Learn the basics of control theory and proportional control loops. Implement a proportional controller in software.

Background information:

Now that we have a firm grasp on basic hardware control, we are ready to proceed into the next section of the workshops. This section depends on everything we have learned and done so far. For this week, we will have a brief introduction on system control theory and how to apply it to minimize the distance error between a target distance and the actual mouse distance.

System control theory deals with the analysis and design of control models and algorithms to drive dynamical systems to a desired state. System control theory has applications on automation, robotics, electronics, finance and numerous other fields in engineering, science and economics. In our case, we can apply control theory to correct for any errors on the traveled distance, turning angle, speed mismatch between wheels and mouse distance from walls.

Before diving into code and theory, let us explain why we even need to use control systems in our mouse. As of right now, the mouse only uses thresholds to stop whenever it reaches a certain distance. This poses a problem because our mouse builds momentum as it accelerates. Starting and stopping the mouse in this way will cause it to overshoot and undershoot the desired travel distance. So, for example, on 10 different tries, we could measure the mouse going over the distance line 7 times and under it 3 times. Introducing a control system will help us minimize this error if we take the time to tune it properly.

Furthermore, we can design other control systems to compensate for error on wheel speeds, turning angle and distance from walls using infrared sensors. With that in mind, let's introduce a figure that shows a practical control loop scheme:

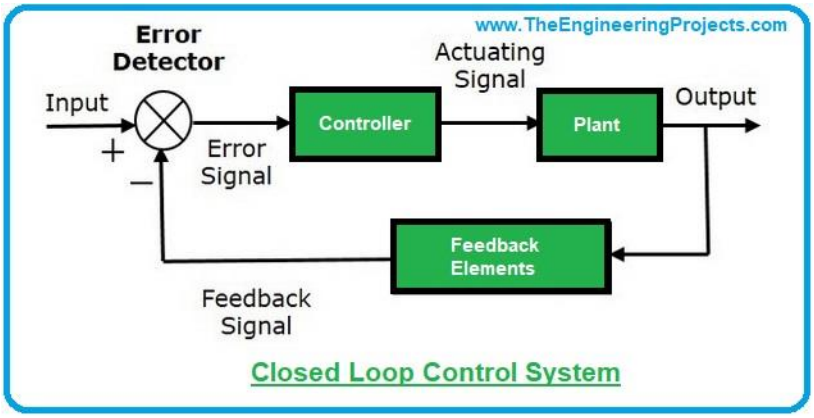

Figure 1 closed loop control system diagram

The figure shows the general diagram for a control system, and while this is useful to understand a generic control scheme, it is not obvious how in translates to our micromouse, so let's begin by explaining it.

INPUT → Motor Speed
ERROR SIGNAL → Target encoder distance – Actual encoder distance
CONTROLLER → Proportional gain * ERROR SIGNAL
PLANT → DC motor
FEEDBACK ELEMENTS → Actual encoder distance
OUTPUT → Motor speed

Hopefully you can see that this controller scheme constantly uses encoder measurements to try and achieve the desired encoder set target. The error is calculated by subtracting the actual encoder distance to the desired encoder distance. Then we use the error to modify the speed of the motors. So, if we have a big error the motors will go faster, and as the error goes to zero the speed also goes to zero. We can control how aggressive our controller corrects for the error by using a *proportional coefficient.* This control scheme is known as a proportional controller, and it is widely used in robotics.

Distance control with proportional gain

At this point you might be confused about how to implement such a control system in software, especially since there are no clear mathematical equations that can be translated into code. So, for completeness, let's introduce some equations that can be derived from looking at figure 1:

$$error = target - actual$$

$$control = kp * error$$

Where,

$$target = desired\ distance\ in\ encoder\ counts$$

$$actual = current\ distance\ in\ encoder\ counts$$

$$kp = proportional\ coefficient$$

The $control$ variable will be used as the output speed to our motors and the $actual$ encoder measurement is the average between the left and right encoders. We can now convert these set of equations into code:

```
1.  #include <Encoder.h>
2.  #include "micromouse.h"
3.
4.  Encoder right_encoder(ENCODER_0_CHA, ENCODER_0_CHB);
5.  Encoder left_encoder(ENCODER_1_CHA, ENCODER_1_CHB);
6.
7.  /*One cell distance is approximately 1090 encoder counts*/
8.  long int target_distance = 1090;
9.  /*Proportional gain coefficient*/
10. float kp = 0.001;
11.
12. void setup()
13. {
14.   /*Initialize motors*/
15.   motors_init();
16.   /*Initialize serial console*/
17.   Serial.begin(9600);
18. }
19.
20. void loop()
21. {
22.   /*Read encoder for left and right motors. Estimate
23.    * mouse traveled distance by taking the average of
24.    * both encoders.
25.    */
26.   long int r_count = right_encoder.read();
27.   long int l_count = -left_encoder.read();
28.   long int encoder_distance = (r_count + l_count) / 2;
29.
30.   /*Distance control loop*/
31.   /*Compute error*/
32.   long int error = target_distance - encoder_distance;
33.   /*Proportional action*/
34.   float output = kp*error;
35.
36.   /*Move motors at specified velocity*/
37.   motor_0_speed((int)output, true);
38.   motor_1_speed((int)output, true);
39.  }
```

The code above will correct the error, but there are still a few issues we need to solve in order to make this control algorithm practical. The 3 main issues are:

1. Determine motor direction based on the sign of the output
2. Clamp output values to a specific MAX_SPEED and MIN_SPEED
3. Make control system repeatable. Once we achieved the desired distance stop the motors and reset encoders so that we can run the control loop again.

Motor Direction

To determine the direction of the motors we can write this code on lines 41-49

```
1.  #include <Encoder.h>
2.  #include "micromouse.h"
3.
4.  Encoder right_encoder(ENCODER_0_CHA, ENCODER_0_CHB);
5.  Encoder left_encoder(ENCODER_1_CHA, ENCODER_1_CHB);
6.
7.  /*One cell distance is approximately 1090 encoder counts*/
8.  long int target_distance = 1090;
9.  /*Proportional gain coefficient*/
10. float kp = 0.001;
11.
12. void setup()
13. {
14.   /*Initialize motors*/
15.   motors_init();
16.   /*Initialize serial console*/
17.   Serial.begin(9600);
18. }
19.
20. void loop()
21. {
22.   /*Read encoder for left and right motors. Estimate
23.    * mouse traveled distance by taking the average of
24.    * both encoders.
25.    */
26.   long int r_count = right_encoder.read();
27.   long int l_count = -left_encoder.read();
28.   long int encoder_distance = (r_count + l_count) / 2;
29.
30.   /*Distance control loop*/
31.   /*Compute error*/
32.   error = target_distance - encoder_distance;
33.   /*Proportional action*/
34.   float output = kp*error;
35.
36.   /*Find the direction of rotation for the wheels*/
37.   bool direction;
38.   if(output > 0)
39.   {
40.     direction = true;
41.   }
42.   else if(output < 0)
43.   {
44.     direction = false;
45.   }
46.
47.   /*Move motors at specified velocity*/
48.   motor_0_speed((int)output, direction);
49.   motor_1_speed((int)output, direction);
50. }
51.
```

Control output clamping

Next, we need to clamp the values to a specific MAX and MIN speed. If we don't include this our control loop won't be able to limit the speed in case the error is too big or too small and our proportional coefficient is not well tuned. This can easily overwhelm a system and cause violent reactions to small error variations.

```
1.  #include <Encoder.h>
2.  #include "micromouse.h"
3.
4.  Encoder right_encoder(ENCODER_0_CHA, ENCODER_0_CHB);
5.  Encoder left_encoder(ENCODER_1_CHA, ENCODER_1_CHB);
6.
7.  const long int MAX_SPEED = 200;
8.  const long int MIN_SPEED = 70;
9.
10. /*One cell distance is approximately 1090 encoder counts*/
11. long int target_distance = 1090;
12. /*Proportional gain coefficient*/
13. float kp = 0.001;
14.
15. void setup()
16. {
17.   /*Initialize motors*/
18.   motors_init();
19.   /*Initialize serial console*/
20.   Serial.begin(9600);
21. }
22.
23. void loop()
24. {
25.   /*Read encoder for left and right motors. Estimate
26.    * mouse traveled distance by taking the average of
27.    * both encoders.
28.    */
29.   long int r_count = right_encoder.read();
30.   long int l_count = -left_encoder.read();
31.   long int encoder_distance = (r_count + l_count) / 2;
32.
33.   /*Distance control loop*/
34.   /*Compute error*/
35.   error = target_distance - encoder_distance;
36.   /*Proportional action*/
37.   float output = kp*error;
38.
39.   /*Find the direction of rotation for the wheels*/
40.   bool direction;
41.   if(output > 0)
42.   {
43.     direction = true;
44.   }
45.   else if(output < 0)
46.   {
47.     direction = false;
48.   }
49.
50.   /*clamp values to a maximum and a minimum*/
51.   if(fabs(output) > MAX_SPEED)
52.   {
53.             output = MAX_SPEED;
54.   }
```

```
55.   else if(fabs(output) < MIN_SPEED)
56.   {
57.             output = MIN_SPEED;
58.   }
59.
60.   /*Move motors at specified velocity*/
61.   motor_0_speed((int)output, direction);
62.   motor_1_speed((int)output, direction);
63. }
64.
```

Repeatable control loop

Finally, we want to make our control repeatable, meaning that we want to execute our control loop as many times as we wish. For this, we need to introduce an MIN_ERROR term that will serve as the absolute minimum error value that we are willing to tolerate. For example, a MIN_ERROR equals to 10 means that we are willing to accept a minimum error of +-10 encoder counts. Ideally MIN_ERROR should be 0, but sometimes achieving this performance requires longer correction time and/or more tuning. Our control algorithm will continuously correct its distance output until we achieve this minimum error tolerance.

Using the code below, you can see that the mouse will repeatedly try to achieve the target distance, then wait for a button press to repeat the process again.

```
1.  #include <Encoder.h>
2.  #include "micromouse.h"
3.
4.  Encoder right_encoder(ENCODER_0_CHA, ENCODER_0_CHB);
5.  Encoder left_encoder(ENCODER_1_CHA, ENCODER_1_CHB);
6.
7.  const long int MAX_SPEED = 255;
8.  const long int MIN_SPEED = 70;
9.  const long int MIN_ERROR = 5;
10.
11. /*One cell distance is approximately 1090 encoder counts*/
12. long int target_distance = 1090;
13. /*Proportional gain coefficient*/
14. float kp = 1.0;
15.
16. void setup()
17. {
18.   /*Initialize motors*/
19.   motors_init();
20.   /*Initialize serial console*/
21.   Serial.begin(9600);
22. }
23.
24. void loop()
25. {
26.   /*Wait for button press*/
27.   while(digitalRead(BUTTON) != HIGH);
28.   delay(1500);
29.
30.   /*Keep running the control loop until the error is less that the
31.    * minimum specified error (MIN_ERROR)
32.    */
33.   long int error;
```

```
34.   do
35.   {
36.     /*Read encoder for left and right motors. Estimate
37.      * mouse traveled distance by taking the average of
38.      * both encoders.
39.      */
40.     long int r_count = right_encoder.read();
41.     long int l_count = -left_encoder.read();
42.     long int encoder_distance = (r_count + l_count) / 2;
43.
44.     /*Distance control loop*/
45.     /*Compute error*/
46.     error = target_distance - encoder_distance;
47.     /*Proportional action*/
48.     float output = kp*error;
49.
50.     /*Find the direction of rotation for the wheels*/
51.     bool direction;
52.     if(output > 0)
53.     {
54.       direction = true;
55.     }
56.     else if(output < 0)
57.     {
58.       direction = false;
59.     }
60.
61.     /*clamp values to a maximum and a minimum*/
62.     if(fabs(output) > MAX_SPEED)
63.     {
64.       output = MAX_SPEED;
65.     }
66.     else if(fabs(output) < MIN_SPEED)
67.     {
68.       output = MIN_SPEED;
69.     }
70.
71.     /*Move motors at specified velocity*/
72.     motor_0_speed((int)output, direction);
73.     motor_1_speed((int)output, direction);
74.     delay(20);
75.   }while(abs(error) > MIN_ERROR);
76.
77.   /*Stop both motors*/
78.   motor_0_speed(0, true);
79.   motor_1_speed(0, true);
80.
81.   /*Reset encoders*/
82.   right_encoder.write(0);
83.   left_encoder.write(0);
84. }
85.
```

Exercise:

Using the code above, write a function for the control loop that follows this prototype:

```
1.  void control_forward(int max_speed)
2.  {
3.   /*Put your control loop code here*/
4.  }
```

Put this function on your micromouse.h and micromouse.cpp files. We will be expanding and using this function in the future.