Colorado School of Mines

Memo

To: Instructor and TA

From: Christian Prather

Team #: M420

Date: 10-22-20

Re: Lab 3: Closed loop control

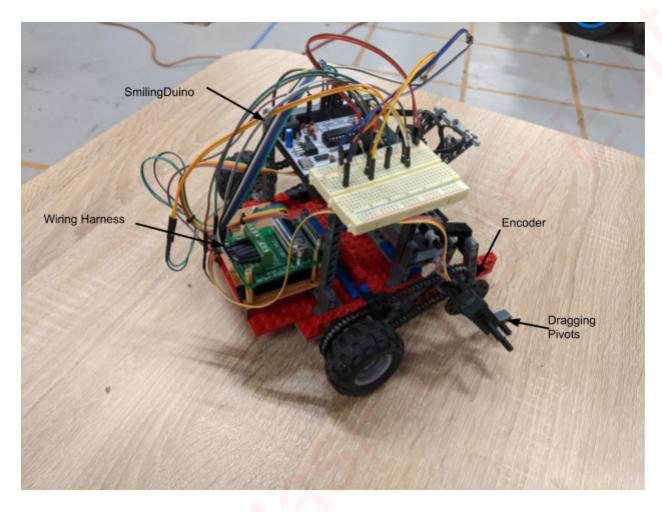
Problem Statement:

Robotic control can be simulated perfectly all day however once you enter the real world variables arise that you can not model. It is the designers job to build a platform that can handle such disturbances. One method of doing so is through sensor feedback. This feedback method closes the loop in the robot's logic and tells it to take into consideration what has happened in the past and adjust to meet your goal. In this lab's case we have introduced encoders to help give feedback as to actual motor rotation.

Methods:

Sensors:

To offer the highest resolution in the encoders values their placement was set at the motor shaft as any placement at the geared down end of my drive train would have resulted in fewer revolutions per linear distance traveled and this would have limited resolution.



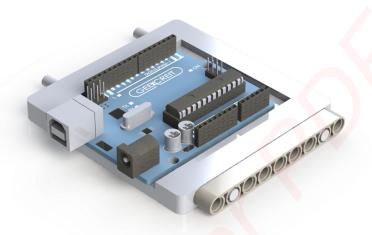
(Fig 1)

I decided that rather than dealing with mounting the encoder using a combination of legos I would design and 3D print a small bracket to mount the encoder to my motor wheel. This went through a couple of revisions but proved a good choice as my largest area of hindrance is my robots chaise.



(Fig 2)

I also designed and printed a mount for my Arduino board that allowed for better mounting then I had in the last lab and added structural integrity to my bot.



(Fig 3)

Algorithm:

I chose to implement my own code from scratch for two primary reasons.

- 1) I wanted the experience as I am very familiar with Arduino programming and wanted to have some more fun with it.
- 2) As labs progress I do not want to be reliant on another person's code (regardless of how good it is) as incorporating new features will be far easier for myself if I am familiar with the code base.

This also allowed me to break out the code to a higher degree utilizing multiple simple functions as well as a standard PID library offered by Arduino which I have used before. A simple implementation (documented in detail in code) follows the flow of taking in next direction command from a list of (command, distance) pairs, calculating the desired encoder counts given constants such as distance/ degrees per rotation and adjusting PWM values of each motor every cycle based on their respective encoder count values (incremented by separate ISRs) compared to a desired count and fed through their individual PID loops. (As a side note I also made a simple wiring harness as I was sick of how delicate my wiring was with the breadboard and connectors, while this is not a great solution in this lab the goal is to order a simple shield from jlcpcb for the next lab. I also replaced the standard Arduino Uno board with a custom one I designed as it has a type c USB port which is only important because it offers less resistance when removing it as I was damaging my bot removing the cable so many times).

Results:

I ran my robot through the test maze a surplus of 20 times and had a mix set of results. The primary conclusion that can be scrubbed is that the software was performing very well, when tested on a bench top with the robot immobile or on hardwood floor the bot would move to the same spot almost 100% of the time and the encoder counts would always be with in 2 away from goal as the deceleration curve was performing great. These results did not hold well when maze testing however as this lab seriously highlighted the shortcoming of the chaise design. Its simple dragging pivots in the back seemed to get caught on anything and everything while the wheels would continue spinning, tricking the feedback loop and throwing the bot entirely off. In runs that proved successful my average completion time was 42.16 seconds making it to square 10.

Table 1 shows the results of 3 relatively successful runs

Note: As my chassis proved a large hindrance I allowed mild overlap of robot – wall boundaries in successful runs)

	Run 1	Run 2	Run 3
Time (seconds)	41	40.5	45

I did need to adjust my kP, kD, and kI values from the starting points provided in the library to compensate for the steady state error. I increased my integral value to allow for better stopping at exact distance with limited overshoot. I also set a boundary for the PID output to max at different values per motor as if both hit 255 early on the variance between them is great enough that I would get drift before it had time to compensate. I also set the lowest to 150 as this allows a slow crawl to move the encoders while not getting stuck in an infinite loop before it enters a deadband.

Conclusions:

There are two major conclusions to be drawn from this lab.

- 1) A simple encoder based feedback system offers an increase in precision given no major disturbance that are capable of tricking them (motor spinning but robot stationary) but the accuracy of the robots final destination is not guaranteed without better world placement sensors.
- 2) My chaise needs a redesign to allow a roller in the back as apposed to two dragging pivots, these pivots get caught far too easily and this lab became an order of magnitude harder due to it. My top priority in the following lab will be to adjust for this issue as well as do a better job accounting for collisions. I would like to add a response handler that utilizes bump sensors to adjust in the event of error beyond what can be handled by the encoders.

I feel my software design and algorithm approach went very well as when tested modularly without impact of non detectable chasie eros its ramping, precision, and accuracy were great. I was surprised however at how bad my chassis is with turning as the pivots get stuck on everything and would cause it to not move while wheels maintained rotation, tricking the encoders. I plan on rectify this with a rear pivot roller in the next lab.

References

PID library documentation: https://playground.arduino.cc/Code/PIDLibrary/

Appendices:

Fig 1: Encoder placement and robot designed

Fig 2: Encoder mount render

Fig 3: Arduino mount render

Table 1: Times of successful maze runs

1 Lab 3 Code Documentation	1
2 File Index	3
2.1 File List	3
3 File Documentation	5
3.1 custom_lab_3/custom_lab_3.ino File Reference	5
3.1.1 Detailed Description	
3.1.2 Macro Definition Documentation	
3.1.2.1 A	
3.1.2.2 B	
3.1.2.3 DegreesPerRev	
3.1.2.4 dirA	
3.1.2.5 dirB	
3.1.2.6 DistancePerRev	
3.1.2.7 EncoderCountsPerRev	
3.1.2.8 EncoderMotorLeft	
3.1.2.9 EncoderMotorRight	
3.1.2.10 FORWARD	
3.1.2.11 IN1	
3.1.2.12 IN2	
3.1.2.13 IN3	
3.1.2.14 IN4	
3.1.2.14 IN4	
3.1.2.16 pushButton	
3.1.2.17 pwmA	
3.1.2.18 pwmB	
3.1.2.19 RIGHT	
3.1.3 Function Documentation	
3.1.3.1 adjustPWM()	11
3.1.3.2 calculateDesiredCount()	11
3.1.3.3 calculateDesiredCountTurn()	12
3.1.3.4 configure()	12
3. <mark>1.3.5 drive()</mark>	12
3.1.3.6 driveBackward()	13
3.1.3.7 driveForward()	13
3.1.3.8 idle()	15
3.1.3.9 indexLeftEncoderCount()	15
3.1.3.10 indexRightEncoderCount()	16
3.1.3.11 loop()	16
3.1.3.12 react_forward()	16
3.1.3.13 react_left()	16
3.1.3.14 react_right()	17

3.2.1.2 run_motor()					
3.2.1 Function Documentation					
3.2 custom_lab_3/motors.ino File Reference					
3.1.4.9 rightOutput					
3.1.4.8 rightEncoderCount					
3.1.4.7 moveList					
3.1.4.6 motorRight_PWM					
3.1.4.5 motorLeft_PWM					
3.1.4.4 milliSecondsPer90Deg	. .				19
3.1.4.3 leftOutput					19
3.1.4.2 leftEncoderCount					19
3.1.4.1 desiredCount		٠.			19
3.1.4 Variable Documentation	٠,			, .	19
3.1.3.18 turnRight()			٠.		18
3.1.3.17 turnLeft()					17
3.1.3.16 setup()				. •	17
3.1.3.15 resetPWM()					17

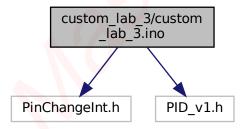
Chapter 3

File Documentation

3.1 custom_lab_3/custom_lab_3.ino File Reference

A basic feedback controlled system for an Arduino based robot.

```
#include <PinChangeInt.h>
#include <PID_v1.h>
Include dependency graph for custom_lab_3.ino:
```



Macros

- #define IN1 9
 - Libraries for interrupts and PID.
- #define IN2 10
- #define IN3 5
- #define IN4 6
- #define A 0

Motor control.

- #define B 1
- #define pwmA 3
- #define dirA 12
- #define pwmB 11

- #define dirB 13
- #define pushButton 2

Start stop button.

• #define EncoderCountsPerRev 12.0

Drive constants - dependent on robot configuration.

- #define DistancePerRev 51.0
- #define DegreesPerRev 27.0
- #define EncoderMotorLeft 7
- #define EncoderMotorRight 8
- #define FORWARD 0

Enum defines.

- #define LEFT 1
- #define RIGHT -1

Functions

void resetPWM ()

Helper function for setting the PWM back to default value.

void indexLeftEncoderCount ()

ISR for left encoder.

void indexRightEncoderCount ()

ISR for incrementing right encoder.

void calculateDesiredCount (int distance)

Calculate how many encoder counts we expect given the distance provided based on the bot intrinsics.

• void calculateDesiredCountTurn (int degrees)

Calculate how many encoder counts we expect given the degrees provided.

void turnRight (int degrees)

Turn bot to given degrees.

• void turnLeft (int degrees)

Turn bot right to given degrees.

void driveForward (int distance)

Function to drive bot forward until encoders are within range.

void driveBackward (int distance)

Drive the bot backwards.

• void configure ()

Function for configuration of pin states and interrupts.

void idle ()

Default behavior when not driving, waits for the pushButton to be pressed so it can execute next command Blocking function.

void adjustPWM ()

Run the PID loop calculation and set out put to motors output in PWM.

void setup ()

Entry point of program handles serial setup and PID config.

void react_left ()

This is the logic to execute if we hit a push button ideally this is never executed as we should never actually hit the walls.

- void react right ()
- void react_forward ()
- void drive ()

Main drive execution of program, iterates through moves list executing next move with corresponding distance or degrees.

void loop ()

Loop execution of the program.

Variables

• double leftEncoderCount = 0

Lab specific variables.

- double rightEncoderCount = 0
- int motorLeft_PWM = 180

Default motor pwm values.

- int motorRight_PWM = 200
- int milliSecondsPer90Deg = 900

Time it takes to move 90 degrees.

· double desiredCount

How many encoder counts for given distance.

- int moveList [] = {FORWARD, 300, LEFT, 90, FORWARD, 300, LEFT, 90, FORWARD, 300, RIGHT, 90, FORWARD, 900, RIGHT, 90, FORWARD, 600, RIGHT, 90, FORWARD, 300}
- double leftOutput

PID values setpoints = desired counts, output = PWM, input = current counts.

double rightOutput

3.1.1 Detailed Description

A basic feedback controlled system for an Arduino based robot.

Author

Christian Prather

Version

0.1

Date

2020-10-21

3.1.2 Macro Definition Documentation

3.1.2.1 A

#define A 0

Motor control.

Definition at line 28 of file custom_lab_3.ino.

3.1.2.2 B

```
#define B 1
```

Definition at line 29 of file custom_lab_3.ino.

3.1.2.3 DegreesPerRev

```
#define DegreesPerRev 27.0
```

Definition at line 41 of file custom_lab_3.ino.

3.1.2.4 dirA

#define dirA 12

Definition at line 31 of file custom_lab_3.ino.

3.1.2.5 dirB

#define dirB 13

Definition at line 33 of file custom_lab_3.ino.

3.1.2.6 DistancePerRev

#define DistancePerRev 51.0

Definition at line 40 of file custom_lab_3.ino.

3.1.2.7 EncoderCountsPerRev

#define EncoderCountsPerRev 12.0

Drive constants - dependent on robot configuration.

Definition at line 39 of file custom_lab_3.ino.

3.1.2.8 EncoderMotorLeft

#define EncoderMotorLeft 7

Definition at line 43 of file custom_lab_3.ino.

3.1.2.9 EncoderMotorRight

#define EncoderMotorRight 8

Definition at line 44 of file custom_lab_3.ino.

3.1.2.10 FORWARD

#define FORWARD 0

Enum defines.

Definition at line 51 of file custom_lab_3.ino.

3.1.2.11 IN1

#define IN1 9

Libraries for interrupts and PID.

Global Defines Motor driver connections

Definition at line 22 of file custom_lab_3.ino.

3.1.2.12 IN2

#define IN2 10

Definition at line 23 of file custom_lab_3.ino.

3.1.2.13 IN3

```
#define IN3 5
```

Definition at line 24 of file custom_lab_3.ino.

3.1.2.14 IN4

```
#define IN4 6
```

Definition at line 25 of file custom_lab_3.ino.

3.1.2.15 LEFT

```
#define LEFT 1
```

Definition at line 52 of file custom_lab_3.ino.

3.1.2.16 pushButton

```
#define pushButton 2
```

Start stop button.

Definition at line 36 of file custom_lab_3.ino.

3.1.2.17 pwmA

```
#define pwmA 3
```

Definition at line 30 of file custom_lab_3.ino.

3.1.2.18 pwmB

#define pwmB 11

Definition at line 32 of file custom_lab_3.ino.

3.1.2.19 RIGHT

```
#define RIGHT -1
```

Definition at line 53 of file custom lab 3.ino.

3.1.3 Function Documentation

3.1.3.1 adjustPWM()

```
void adjustPWM ( )
```

Run the PID loop calculation and set out put to motors output in PWM.

Definition at line 324 of file custom_lab_3.ino.

```
325 {
326
           // Compute the pid values
327
          leftPID.Compute();
328
          rightPID.Compute();
329
330
          // Set the pid values within range
          motorLeft_PWM = constrain(leftOutput, 150, 250);
motorRight_PWM = constrain(rightOutput, 150, 235);
Serial.print("Left PWM: ");
331
332
333
          Serial.print(motorLeft_PWM);
Serial.print(" ");
334
335
336
          Serial.println(leftEncoderCount);
337
          Serial.print("Right PWM: ");
          Serial.print(motorRight_PWM);
Serial.print(" ");
338
339
340
          Serial.println(rightEncoderCount);
341 }
```

3.1.3.2 calculateDesiredCount()

Calculate how many encoder counts we expect given the distance provided based on the bot intrinsics.

Parameters

distance

Definition at line 109 of file custom_lab_3.ino.

3.1.3.3 calculateDesiredCountTurn()

Calculate how many encoder counts we expect given the degrees provided.

Parameters

degrees

Definition at line 126 of file custom_lab_3.ino.

3.1.3.4 configure()

```
void configure ( )
```

Function for configuration of pin states and interrupts.

Definition at line 288 of file custom_lab_3.ino.

```
290
         // set up the motor drive ports
291
        pinMode(pwmA, OUTPUT);
292
        pinMode(dirA, OUTPUT);
293
       pinMode(pwmB, OUTPUT);
294
       pinMode(dirB, OUTPUT);
295
296
       pinMode(pushButton, INPUT_PULLUP);
297
298
        pinMode(EncoderMotorLeft, INPUT_PULLUP); //set the pin to input
299
        PCintPort::attachInterrupt(EncoderMotorLeft, indexLeftEncoderCount, CHANGE);
300
301
        pinMode(EncoderMotorRight, INPUT_PULLUP); //set the pin to input
302
        PCintPort::attachInterrupt(EncoderMotorRight, indexRightEncoderCount, CHANGE);
303 }
```

3.1.3.5 drive()

```
void drive ( )
```

Main drive execution of program, iterates through moves list executing next move with corresponding distance or degrees.

Definition at line 383 of file custom_lab_3.ino.

```
386
        for (int i = 0; i < sizeof(moveList); i += 2)</pre>
387
388
            idle();
389
            switch (moveList[i])
390
391
            case LEFT:
392
                turnLeft(moveList[i + 1]);
393
394
            case RIGHT:
395
                turnRight(moveList[i + 1]);
396
                break:
397
            case FORWARD:
398
                driveForward(moveList[i + 1]);
399
                break;
400
            default:
401
                break;
402
403
404 }
```

3.1.3.6 driveBackward()

```
void driveBackward (
          int distance )
```

Drive the bot backwards.

Parameters

distance

Definition at line 253 of file custom_lab_3.ino.

```
254 {
255
        resetPWM();
256
        calculateDesiredCount(distance);
257
258
        // Loop unit1 the encoders read correct
259
260
        while ((desiredCount - leftEncoderCount) > 3 || (desiredCount - rightEncoderCount) > 3)
261
262
            adjustPWM();
            //{\mbox{To drive backward, motors go in the same direction}}
263
264
265
            if ((desiredCount - leftEncoderCount) > 3)
266
267
                run_motor(A, motorLeft_PWM); //change PWM to your calibrations
268
            if ((desiredCount - rightEncoderCount) > 3)
269
270
271
                run_motor(B, motorRight_PWM); //change PWM to your calibrations
272
273
274
275
        // motors stop
276
        run_motor(A, 0);
run_motor(B, 0);
278
        Serial.println("Done driving backwards");
279
        Serial.print("L: ");
280
        Serial.println(leftEncoderCount);
281
        Serial.print("R: ");
282
        Serial.println(rightEncoderCount);
283 }
```

3.1.3.7 driveForward()

Function to drive bot forward until encoders are within range.

Parameters

distance

Definition at line 215 of file custom_lab_3.ino.

```
216 {
217
         Serial.println("Driving Forward...");
218
         resetPWM();
219
        calculateDesiredCount(distance);
220
221
        // Loop unitl the encoders read correct
222
223
        while ((desiredCount - leftEncoderCount) > 3 || (desiredCount - rightEncoderCount)
224
225
             adiustPWM();
226
             //To drive forward, motors go in the same direction
227
228
             if ((desiredCount - leftEncoderCount) > 3)
229
230
                 run_motor(A, -motorLeft_PWM); //change PWM to your calibrations
231
232
             if ((desiredCount - rightEncoderCount) > 3)
233
234
                 run_motor(B, -motorRight_PWM); //change PWM to your calibrations
235
        }
236
237
         // motors stop
238
        run_motor(A, 0);
run_motor(B, 0);
239
240
241
         Serial.println("Done driving forward");
242
         Serial.print("L: ");
        Serial.println(leftEncoderCount);
Serial.print("R: ");
243
244
245
        Serial.println(rightEncoderCount);
246 }
```

3.1.3.8 idle()

```
void idle ( )
```

Default behavior when not driving, waits for the pushButton to be pressed so it can execute next command Blocking function.

Definition at line 310 of file custom_lab_3.ino.

```
311 {
312     Serial.println("Idle..");
313     while (digitalRead(pushButton) == 1)
314     ; // wait for button push
315     while (digitalRead(pushButton) == 0)
316     ; // wait for button release
317     delay(2000); // Give time to move hand
318 }
```

3.1.3.9 indexLeftEncoderCount()

```
void indexLeftEncoderCount ( )
```

ISR for left encoder.

Definition at line 88 of file custom_lab_3.ino.

```
89 {
90    leftEncoderCount++;
91    //Serial.println("Left Encoder ++");
92 }
```

3.1.3.10 indexRightEncoderCount()

```
void indexRightEncoderCount ( )
```

ISR for incrementing right encoder.

Definition at line 97 of file custom_lab_3.ino.

```
98 {
99    rightEncoderCount++;
100    //Serial.println("Right Encoder ++");
101 }
```

3.1.3.11 loop()

```
void loop ( )
```

Loop execution of the program.

Definition at line 409 of file custom_lab_3.ino.

```
410 {
411 drive();
412 }
```

3.1.3.12 react_forward()

```
void react_forward ( )
```

Definition at line 373 of file custom_lab_3.ino.

3.1.3.13 react_left()

```
void react_left ( )
```

This is the logic to execute if we hit a push button ideally this is never executed as we should never actually hit the walls.

Definition at line 359 of file custom_lab_3.ino.

3.1.3.14 react_right()

```
void react_right ( )
```

Definition at line 366 of file custom_lab_3.ino.

```
368 // TODO: Check which button was hit
369
370 driveBackward(20);
371 turnLeft(30);
372 }
```

3.1.3.15 resetPWM()

```
void resetPWM ( )
```

Helper function for setting the PWM back to default value.

Definition at line 79 of file custom_lab_3.ino.

```
80 {
81     motorLeft_PWM = 180;
82     motorRight_PWM = 200;
83 }
```

3.1.3.16 setup()

```
void setup ( )
```

Entry point of program handles serial setup and PID config.

Definition at line 346 of file custom_lab_3.ino.

3.1.3.17 turnLeft()

Turn bot right to given degrees.

Parameters

degrees

Definition at line 178 of file custom_lab_3.ino.

```
180
         resetPWM();
        calculateDesiredCountTurn(degrees);
181
182
183
         // Loop unit1 the encoders read correct
184
185
        while ((desiredCount - leftEncoderCount) > 3)
186
187
             adjustPWM();
188
             //{\tt To} drive forward, motors go in the same direction
189
190
             if ((desiredCount - leftEncoderCount) > 3)
191
192
                  run_motor(A, motorLeft_PWM); //change PWM to your calibrations
193
             if ((desiredCount - rightEncoderCount) > 3)
194
195
196
                 run_motor(B, -motorRight_PWM); //change PWM to your calibrations
197
198
199
        // motors stop
200
        run_motor(A, 0);
run_motor(B, 0);
Serial.println("Done driving Left");
201
202
203
204
        Serial.print("L: ");
205
        Serial.println(leftEncoderCount);
206
        Serial.print("R: ");
        Serial.println(rightEncoderCount);
207
208 }
```

3.1.3.18 turnRight()

```
void turnRight (
          int degrees )
```

Turn bot to given degrees.

Parameters

degrees

Definition at line 141 of file custom_lab_3.ino.

```
142 {
143
         resetPWM(); // Reset pwm
        calculateDesiredCountTurn(degrees);
144
145
         // While the encoders are not correct adjust PWM with PID loop
146
         // Loop unit1 the encoders read correct
147
148
        while ((desiredCount - rightEncoderCount) > 3)
149
             adiustPWM();
150
151
             //To drive forward, motors go in the same direction
152
153
             if ((desiredCount - leftEncoderCount) > 3)
154
155
                 run_motor(A, -motorLeft_PWM); //change PWM to your calibrations
156
157
             if ((desiredCount - rightEncoderCount) > 3)
158
159
                 run_motor(B, motorRight_PWM); //change PWM to your calibrations
160
161
162
         // motors stop
163
        run_motor(A, 0);
run_motor(B, 0);
164
165
166
         Serial.println("Done driving Right");
167
        Serial.print("L: ");
        Serial.println(leftEncoderCount);
Serial.print("R: ");
168
169
170
        Serial.println(rightEncoderCount);
171 }
```

3.1.4 Variable Documentation

3.1.4.1 desiredCount

double desiredCount

How many encoder counts for given distance.

Definition at line 63 of file custom_lab_3.ino.

3.1.4.2 leftEncoderCount

```
PID leftPID & leftEncoderCount = 0
```

Lab specific variables.

Definition at line 47 of file custom_lab_3.ino.

3.1.4.3 leftOutput

double leftOutput

PID values setpoints = desired counts, output = PWM, input = current counts.

Definition at line 72 of file custom_lab_3.ino.

3.1.4.4 milliSecondsPer90Deg

```
int milliSecondsPer90Deg = 900
```

Time it takes to move 90 degrees.

Definition at line 60 of file custom_lab_3.ino.

3.1.4.5 motorLeft_PWM

int motorLeft_PWM = 180

Default motor pwm values.

Definition at line 56 of file custom_lab_3.ino.

3.1.4.6 motorRight_PWM

```
int motorRight_PWM = 200
```

Definition at line 57 of file custom lab 3.ino.

3.1.4.7 moveList

```
int moveList[] = {FORWARD, 300, LEFT, 90, FORWARD, 300, LEFT, 90, FORWARD, 300, RIGHT, 90,
FORWARD, 900, RIGHT, 90, FORWARD, 600, RIGHT, 90, FORWARD, 300}
```

Definition at line 66 of file custom_lab_3.ino.

3.1.4.8 rightEncoderCount

```
PID rightPID & rightEncoderCount = 0
```

Definition at line 48 of file custom_lab_3.ino.

3.1.4.9 rightOutput

```
double rightOutput
```

Definition at line 72 of file custom_lab_3.ino.

3.2 custom_lab_3/motors.ino File Reference

Functions

- void motor_setup ()
- void run_motor (int motor, int pwm)

3.2.1 Function Documentation

3.2.1.1 motor_setup()

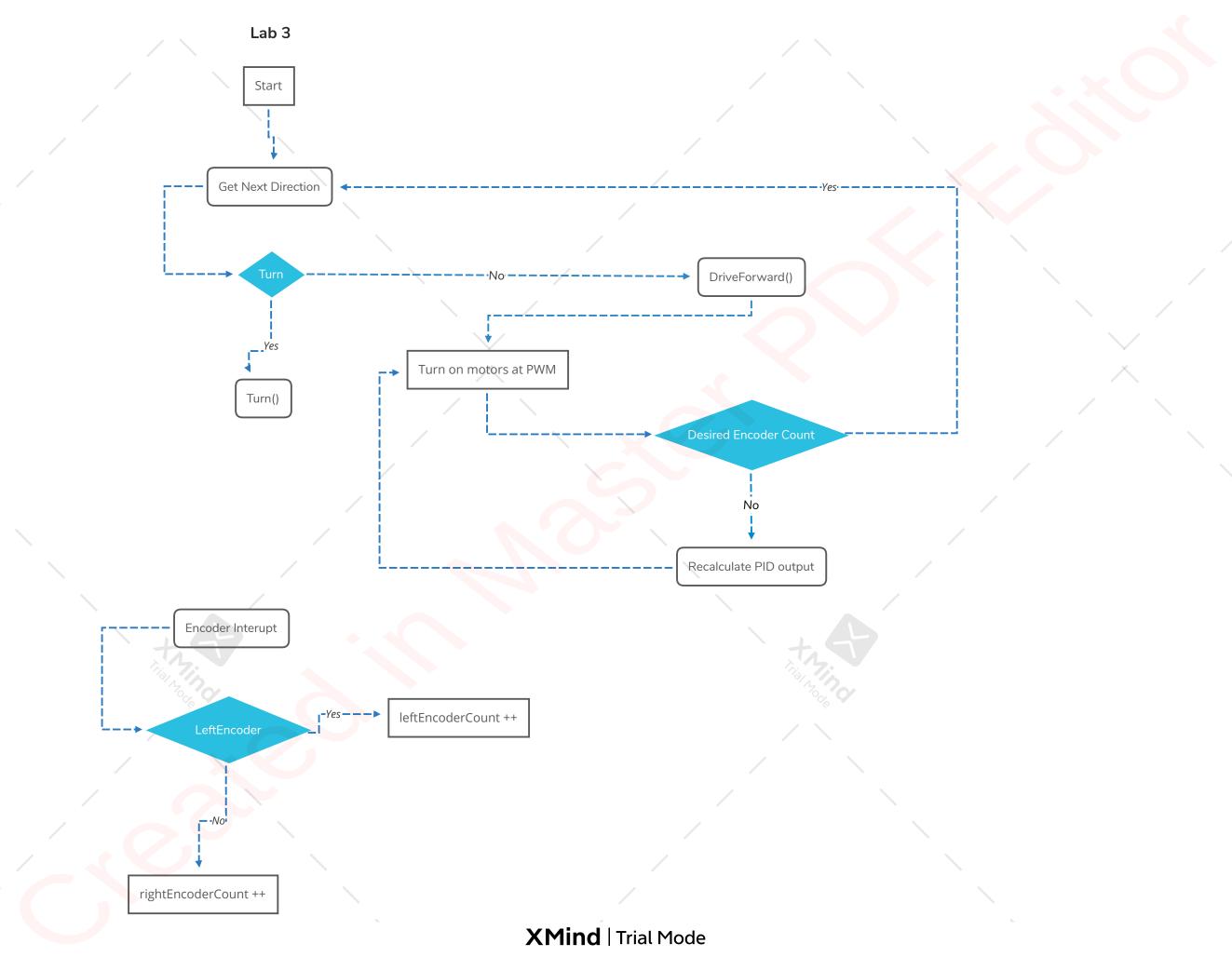
```
void motor_setup ( )
Definition at line 1 of file motors.ino.
     // if using dual motor driver
     // define driver pins as outputs
5
     pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
pinMode(IN3, OUTPUT);
6
   pinMode(IN4, OUTPUT);
    // initialize all pins to zero
10
    digitalWrite(IN1, 0);
digitalWrite(IN2, 0);
digitalWrite(IN3, 0);
digitalWrite(IN4, 0);
14
     return;
15 } // end function
```

3.2.1.2 run_motor()

```
void run_motor (
          int motor,
          int pwm )
```

Definition at line 19 of file motors.ino.

```
20 {
    2.1
22
23
    switch (motor)
    { // find which motor to control case A: // if A, write A pins
25
26
2.7
     if (dir)
        // If dir is forward analogWrite(IN1, pwm); // IN1 is the forward pwm pin digitalWrite(IN2, LOW); // IN2 is low
28
29
30
31
32
33
         digitalWrite(IN1, LOW); // IN1 is low
34
        analogWrite(IN2, pwm); // IN2 is the reverse pwm pin // end if
35
36
37
       break;
                                 // end case A
                              // if B, write B pins
38
     case B:
     if (dir)
39
        // if dir is forward analogWrite(IN3, pwm); // IN3 is the forward pwm pin digitalWrite(IN4, LOW); // IN4 is low
40
41
42
43
44
4.5
        46
47
48
49
                                 // end case B
50
                                 // end switch case
    return;
51
   } // end function
```



```
custom lab 3.ino
 1 /**
 2
      @file custom lab 3.ino
 3
      @author Christian Prather
 4
      @brief A basic feedback controlled system for an Arduino based robot
 5
      @version 0.1
 6
      @date 2020-10-21
 7
 8
 9
  /*! \mainpage Lab 3 Code Documentation
10
11
   */
12
13
15 /// Libraries for interrupts and PID
16 #include <PinChangeInt.h>
17 #include <PID v1.h>
18
19 /// Global Defines
20
21 /// Motor driver connections
22 #define IN1 9
23 #define IN2 10
24 #define IN3 5
25 #define IN4 6
26
27 /// Motor control
28 #define A 0
29 #define B 1
30 #define pwmA 3
31 #define dirA 12
32 #define pwmB 11
33 #define dirB 13
34
35 /// Start stop button
36 #define pushButton 2
37
38 /// Drive constants - dependent on robot configuration
39 #define EncoderCountsPerRev 12.0
40 #define DistancePerRev 51.0
41 #define DegreesPerRev 27.0
42
43 #define EncoderMotorLeft 7
44 #define EncoderMotorRight 8
45
46 /// Lab specific variables
47 double leftEncoderCount = 0;
48 double rightEncoderCount = 0;
49
50 /// Enum defines
51 #define FORWARD 0
52 #define LEFT 1
53 #define RIGHT -1
54
55 /// Default motor pwm values
56 int motorLeft PWM = 180;
57 int motorRight PWM = 200;
58
59 /// Time it takes to move 90 degrees
60 int milliSecondsPer90Deg = 900;
```

localhost:4649/?mode=clike 1/8

```
109 void calculateDesiredCount(int distance)
110 | {
        double revolutionsRequired = distance / DistancePerRev;
111
112
113
        desiredCount = revolutionsRequired * EncoderCountsPerRev;
114
        // Reset encoder counts
115
        leftEncoderCount = 0;
116
        rightEncoderCount = 0;
```

localhost:4649/?mode=clike 2/8

```
10/22/2020
                                            custom lab 3.ino
117
         Serial.print("Desired Count: ");
118
         Serial.println(desiredCount);
119 }
120
 121 /**
     * @brief Calculate how many encoder counts we expect given the degrees
 122
    provided
123
124
     * @param degrees
 125
     */
 126 void calculateDesiredCountTurn(int degrees)
 127 {
128
         double revolutionsRequired = degrees / DegreesPerRev;
129
         desiredCount = revolutionsRequired * EncoderCountsPerRev;
130
         leftEncoderCount = 0;
131
         rightEncoderCount = 0;
132
         Serial.print("Desired Count: ");
133
         Serial.println(desiredCount);
134 }
135
136 /**
137
        @brief Turn bot to given degrees
138
139
        @param degrees
 140 */
141 void turnRight(int degrees)
142 | {
 143
         resetPWM(); // Reset pwm
 144
         calculateDesiredCountTurn(degrees);
         // While the encoders are not correct adjust PWM with PID loop
 145
146
         // Loop unitl the encoders read correct
 147
148
         while ((desiredCount - rightEncoderCount) > 3)
 149
 150
             adjustPWM();
151
             //To drive forward, motors go in the same direction
152
             if ((desiredCount - leftEncoderCount) > 3)
153
154
             {
 155
                 run motor(A, -motorLeft PWM); //change PWM to your calibrations
156
 157
             if ((desiredCount - rightEncoderCount) > 3)
 158
159
                 run motor(B, motorRight PWM); //change PWM to your calibrations
 160
             }
         }
161
162
163
         // motors stop
164
         run motor(A, 0);
         run_motor(B, 0);
165
166
         Serial.println("Done driving Right");
167
         Serial.print("L: ");
168
         Serial.println(leftEncoderCount);
169
         Serial.print("R: ");
         Serial.println(rightEncoderCount);
 170
171 }
172
 173 /**
174
        @brief Turn bot right to given degrees
 175
```

localhost:4649/?mode=clike 3/8

```
10/22/2020
                                            custom lab 3.ino
176
        @param degrees
 177 */
 178 void turnLeft(int degrees)
179 {
 180
         resetPWM();
 181
         calculateDesiredCountTurn(degrees);
 182
183
         // Loop unitl the encoders read correct
184
         while ((desiredCount - leftEncoderCount) > 3)
 185
 186
         {
 187
             adjustPWM();
             //To drive forward, motors go in the same direction
 188
189
             if ((desiredCount - leftEncoderCount) > 3)
 190
 191
                 run motor(A, motorLeft PWM); //change PWM to your calibrations
 192
 193
 194
             if ((desiredCount - rightEncoderCount) > 3)
 195
196
                 run motor(B, -motorRight PWM); //change PWM to your calibrations
 197
             }
198
         }
199
200
         // motors stop
201
         run motor(A, \Theta);
         run_motor(B, 0);
202
         Serial.println("Done driving Left");
203
         Serial.print("L: ");
204
         Serial.println(leftEncoderCount);
205
206
         Serial.print("R: ");
207
         Serial.println(rightEncoderCount);
 208 }
209
210 /**
        @brief Function to drive bot forward until encoders are within range
211
 212
213
        @param distance
214 */
215 void driveForward(int distance)
216 {
 217
         Serial.println("Driving Forward...");
 218
         resetPWM():
         calculateDesiredCount(distance);
219
220
221
         // Loop unitl the encoders read correct
222
223
         while ((desiredCount - leftEncoderCount) > 3 || (desiredCount -
     rightEncoderCount) > 3)
224
225
             adjustPWM();
226
             //To drive forward, motors go in the same direction
227
             if ((desiredCount - leftEncoderCount) > 3)
228
229
                 run motor(A, -motorLeft PWM); //change PWM to your calibrations
 230
231
             if ((desiredCount - rightEncoderCount) > 3)
 232
 233
234
                 run motor(B, -motorRight PWM); //change PWM to your calibrations
```

localhost:4649/?mode=clike 4/8

```
10/22/2020
                                            custom lab 3.ino
235
             }
         }
236
237
238
         // motors stop
239
         run motor(A, \Theta);
240
         run motor(B, 0);
241
         Serial.println("Done driving forward");
242
         Serial.print("L: ");
243
         Serial.println(leftEncoderCount);
244
         Serial.print("R: ");
245
         Serial.println(rightEncoderCount);
246 }
247
248 /**
249
        @brief Drive the bot backwards
250
 251
        @param distance
252 */
253 void driveBackward(int distance)
254 {
255
         resetPWM();
256
         calculateDesiredCount(distance);
257
         // Loop unitl the encoders read correct
258
259
260
         while ((desiredCount - leftEncoderCount) > 3 || (desiredCount -
     rightEncoderCount) > 3)
261
         {
262
             adjustPWM();
             //To drive backward, motors go in the same direction
263
264
265
             if ((desiredCount - leftEncoderCount) > 3)
266
267
                  run motor(A, motorLeft PWM); //change PWM to your calibrations
268
             if ((desiredCount - rightEncoderCount) > 3)
269
 270
             {
271
                  run motor(B, motorRight PWM); //change PWM to your calibrations
272
             }
         }
 273
274
         // motors stop
275
 276
         run motor(A, 0);
277
         run motor(B, 0);
278
         Serial.println("Done driving backwards");
279
         Serial.print("L: ");
280
         Serial.println(leftEncoderCount);
281
         Serial.print("R: ");
282
         Serial.println(rightEncoderCount);
283 }
284
285 /**
        @brief Function for configuration of pin states and interrupts
286
287 */
 288 void configure()
289 {
290
         // set up the motor drive ports
291
         pinMode(pwmA, OUTPUT);
292
         pinMode(dirA, OUTPUT);
         pinMode(pwmB, OUTPUT);
293
```

localhost:4649/?mode=clike 5/8

```
10/22/2020
                                            custom lab 3.ino
294
         pinMode(dirB, OUTPUT);
295
296
         pinMode(pushButton, INPUT_PULLUP);
297
298
         pinMode(EncoderMotorLeft, INPUT PULLUP); //set the pin to input
         PCintPort::attachInterrupt(EncoderMotorLeft, indexLeftEncoderCount,
299
     CHANGE);
300
301
         pinMode(EncoderMotorRight, INPUT PULLUP); //set the pin to input
         PCintPort::attachInterrupt(EncoderMotorRight, indexRightEncoderCount,
302
     CHANGE);
303 }
304
305 /**
        @brief Default behavior when not driving, waits for the pushButton to
306
307
        be pressed so it can execute next command
308
        Blocking function
309 */
310 void idle()
 311 {
312
         Serial.println("Idle..");
313
         while (digitalRead(pushButton) == 1)
314
             ; // wait for button push
315
         while (digitalRead(pushButton) == 0)
316
             ; // wait for button release
317
         delay(2000); // Give time to move hand
318 }
319
320 /**
        @brief Run the PID loop calculation and set out put to motors output in
 321
     PWM
322
 323 */
324 void adjustPWM()
325 | {
326
         // Compute the pid values
 327
         leftPID.Compute();
 328
         rightPID.Compute();
329
         // Set the pid values within range
 330
331
         motorLeft PWM = constrain(leftOutput, 150, 250);
332
         motorRight PWM = constrain(rightOutput, 150, 235);
 333
         Serial.print("Left PWM: ");
334
         Serial.print(motorLeft PWM);
         Serial.print(" ");
335
         Serial.println(leftEncoderCount);
336
337
         Serial.print("Right PWM: ");
338
         Serial.print(motorRight PWM);
339
         Serial.print(" ");
         Serial.println(rightEncoderCount);
 340
341 }
342
 343 /**
344
        @brief Entry point of program handles serial setup and PID config
345 */
 346 void setup()
347 {
 348
         Serial.begin(9600);
 349
         Serial.println("Setting up.....");
350
         configure();
```

localhost:4649/?mode=clike 6/8

```
10/22/2020
                                             custom lab 3.ino
351
         leftPID.SetMode(AUTOMATIC);
352
         rightPID.SetMode(AUTOMATIC);
353 }
354
 355 /**
        @brief This is the logic to execute if we hit a push button
356
357
        ideally this is never executed as we shoudl never actually hit the walls
358 */
359 void react left()
 360 {
         // TODO: Check which button was hit
361
362
363
         driveBackward(20);
364
         turnRight(30);
365 }
 366 void react_right()
367 {
         // TODO: Check which button was hit
 368
369
 370
         driveBackward(20);
371
         turnLeft(30);
372 }
 373 void react forward()
374 {
         // TODO: Check which button was hit
 375
376
         driveBackward(50);
377 }
 378
379 /**
        @brief Main drive execution of program, iterates through moves list
380
     executing
381
        next move with corresponding distance or degrees
 382 */
383 void drive()
384 {
385
         // Iterate over the list jumping by two each time
         for (int i = 0; i < sizeof(moveList); i += 2)
386
 387
         {
388
             idle();
             switch (moveList[i])
 389
390
391
             case LEFT:
 392
                  turnLeft(moveList[i + 1]);
393
                 break;
394
             case RIGHT:
395
                  turnRight(moveList[i + 1]);
396
                 break:
 397
             case FORWARD:
                 driveForward(moveList[i + 1]);
398
399
                  break;
             default:
400
401
                 break;
402
403
404 }
405
406 /**
407
        @brief Loop execution of the program
408 */
409 void loop()
```

localhost:4649/?mode=clike 7/8

10/22/2020 custom_lab_3.ino

410 { 411 drive(); 412 } 413

localhost:4649/?mode=clike

10/22/2020 motors.ino

```
1 void motor_setup()
 2 {
 3
     // if using dual motor driver
 4
     // define driver pins as outputs
 5
     pinMode(IN1, OUTPUT);
     pinMode(IN2, OUTPUT);
 6
 7
     pinMode(IN3, OUTPUT);
     pinMode(IN4, OUTPUT);
 8
 9
     // initialize all pins to zero
10
     digitalWrite(IN1, 0);
     digitalWrite(IN2, 0);
11
     digitalWrite(IN3, 0);
12
     digitalWrite(IN4, 0);
13
     return;
14
15 } // end function
16
17 // int motor is the defined A or B
18 // pwm = the power cycle you want to use
19 void run motor(int motor, int pwm)
20 | {
21
     int dir = (pwm / abs(pwm)) > 0; // returns if direction is forward (1) or
   reverse (0)
     pwm = abs(pwm);
                                      // only positive values can be sent to the
  motor
23
24
     switch (motor)
25
             // find which motor to control
26
     case A: // if A, write A pins
27
       if (dir)
28
       {
                                  // If dir is forward
         analogWrite(IN1, pwm); // IN1 is the forward pwm pin
29
         digitalWrite(IN2, LOW); // IN2 is low
30
31
       }
32
       else
33
       {
34
         digitalWrite(IN1, LOW); // IN1 is low
         analogWrite(IN2, pwm); // IN2 is the reverse pwm pin
35
                                  // end if
36
37
       break;
                                  // end case A
                                  // if B, write B pins
38
     case B:
39
       if (dir)
40
                                  // if dir is forward
41
         analogWrite(IN3, pwm); // IN3 is the forward pwm pin
42
         digitalWrite(IN4, LOW); // IN4 is low
43
       }
44
       else
45
       {
46
         digitalWrite(IN3, LOW); //IN3 is low
47
         analogWrite(IN4, pwm); // IN4 is the reverse pwm pin
48
                                  // end if
49
                                  // end case B
       break;
                                  // end switch case
50
     }
51
     return;
52 } // end function
53
```

localhost:4649/?mode=clike 1/1