Colorado School of Mines

# Memo

To: Instructor and TA

From: Christian Prather

Team #: M420

Date: 10-22-20

Re: Lab 3: Closed loop control

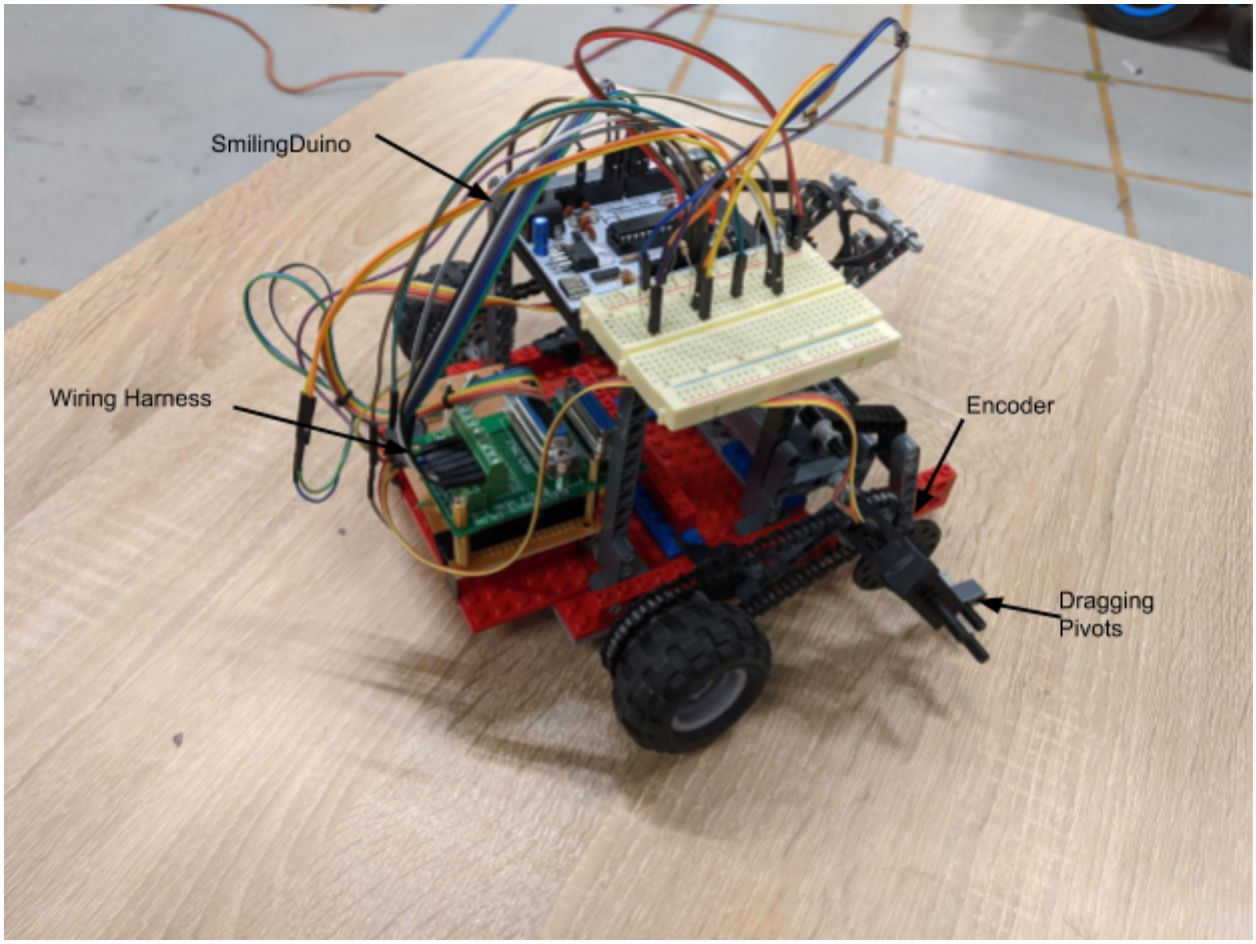
**Problem Statement:**

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.

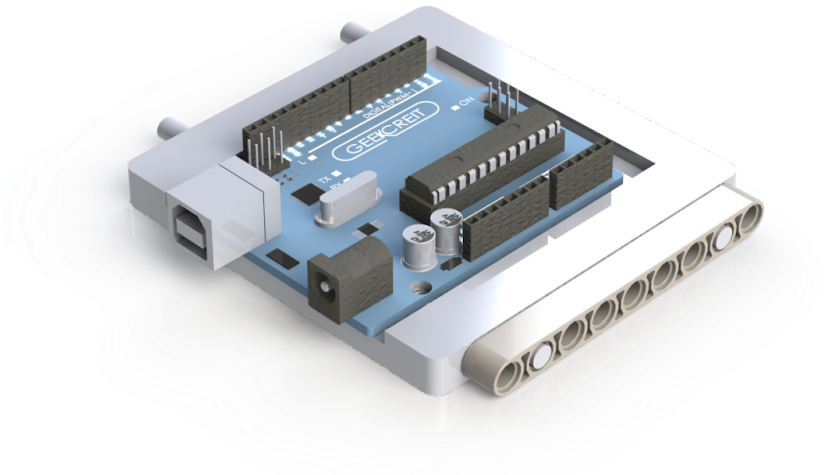



(Fig 1)

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/

Starter Code

**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 . . . 7

3.1.2 Macro Definition Documentation . . . 7

3.1.2.1 A . . . 7

3.1.2.2 B . . . 8

3.1.2.3 DegreesPerRev . . . 8

3.1.2.4 dirA . . . 8

3.1.2.5 dirB . . . 8

3.1.2.6 DistancePerRev . . . 8

3.1.2.7 EncoderCountsPerRev . . . 8

3.1.2.8 EncoderMotorLeft . . . 9

3.1.2.9 EncoderMotorRight . . . 9

3.1.2.10 FORWARD . . . 9

3.1.2.11 IN1 . . . 9

3.1.2.12 IN2 . . . 9

3.1.2.13 IN3 . . . 10

3.1.2.14 IN4 . . . 10

3.1.2.15 LEFT . . . 10

3.1.2.16 pushButton . . . 10

3.1.2.17 pwmA . . . 10

3.1.2.18 pwmB . . . 10

3.1.2.19 RIGHT . . . 11

3.1.3 Function Documentation . . . 11

3.1.3.1 adjustPWM() . . . 11

3.1.3.2 calculateDesiredCount() . . . 11

3.1.3.3 calculateDesiredCountTurn() . . . 12

3.1.3.4 configure() . . . 12

3.1.3.5 drive() . . . 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

3.1.3.15 resetPWM() . . . 17
3.1.3.16 setup() . . . 17
3.1.3.17 turnLeft() . . . 17
3.1.3.18 turnRight() . . . 18
3.1.4 Variable Documentation . . . 19
3.1.4.1 desiredCount . . . 19
3.1.4.2 leftEncoderCount . . . 19
3.1.4.3 leftOutput . . . 19
3.1.4.4 milliSecondsPer90Deg . . . 19
3.1.4.5 motorLeft_PWM . . . 19
3.1.4.6 motorRight_PWM . . . 20
3.1.4.7 moveList . . . 20
3.1.4.8 rightEncoderCount . . . 20
3.1.4.9 rightOutput . . . 20
3.2 custom_lab_3/motors.ino File Reference . . . 20
3.2.1 Function Documentation . . . 20
3.2.1.1 motor_setup() . . . 21
3.2.1.2 run_motor() . . . 21

**Index** **23**

# Chapter 3

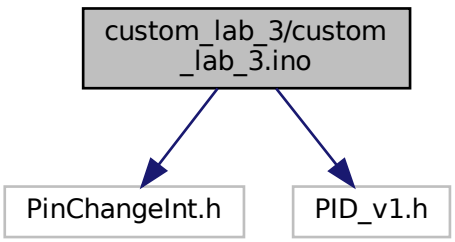
# File Documentation

## 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 shoudl 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
331     motorLeft_PWM = constrain(leftOutput, 150, 250);
332     motorRight_PWM = constrain(rightOutput, 150, 235);
333     Serial.print("Left PWM: ");
334     Serial.print(motorLeft_PWM);
335     Serial.print(" ");
336     Serial.println(leftEncoderCount);
337     Serial.print("Right PWM: ");
338     Serial.print(motorRight_PWM);
339     Serial.print(" ");
340     Serial.println(rightEncoderCount);
341 }
```

#### 3.1.3.2 calculateDesiredCount()

```
void calculateDesiredCount (
              int distance )
```

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.

```
110 {
111     double revolutionsRequired = distance / DistancePerRev;
112 
113     desiredCount = revolutionsRequired * EncoderCountsPerRev;
114     // Reset encoder counts
115     leftEncoderCount = 0;
116     rightEncoderCount = 0;
117     Serial.print("Desired Count: ");
118     Serial.println(desiredCount);
119 }
```

#### 3.1.3.3 calculateDesiredCountTurn()

```
void calculateDesiredCountTurn (
            int degrees )
```

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

**Parameters**

| | |
|---|---|
| *degrees* | |

Definition at line 126 of file custom_lab_3.ino.

```
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 }
```

#### 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.

```
289 {
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.

```
384 {
385     // Iterate over the list jumping by two each time
```

```
386     for (int i = 0; i < sizeof(moveList); i += 2)
387     {
388         idle();
389         switch (moveList[i])
390         {
391         case LEFT:
392             turnLeft(moveList[i + 1]);
393             break;
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 unitl the encoders read correct
259 
260     while ((desiredCount - leftEncoderCount) > 3 || (desiredCount - rightEncoderCount) > 3)
261     {
262         adjustPWM();
263         //To drive backward, motors go in the same direction
264 
265         if ((desiredCount - leftEncoderCount) > 3)
266         {
267             run_motor(A, motorLeft_PWM); //change PWM to your calibrations
268         }
269         if ((desiredCount - rightEncoderCount) > 3)
270         {
271             run_motor(B, motorRight_PWM); //change PWM to your calibrations
272         }
273     }
274 
275     // motors stop
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 }
```

#### 3.1.3.7 driveForward()

```
void driveForward (
              int distance )
```

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) > 3)
224     {
225         adjustPWM();
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 
238     // motors stop
239     run_motor(A, 0);
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 }
```

#### 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.

```
374 {
375     // TODO: Check which button was hit
376     driveBackward(50);
377 }
```

#### 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 shoudl never actually hit the walls.

Definition at line 359 of file custom_lab_3.ino.

```
360 {
361     // TODO: Check which button was hit
362
363     driveBackward(20);
364     turnRight(30);
365 }
```

#### 3.1.3.14 react_right()

```
void react_right ( )
```

Definition at line 366 of file custom_lab_3.ino.

```
367 {
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.

```
347 {
348     Serial.begin(9600);
349     Serial.println("Setting up.....");
350     configure();
351     leftPID.SetMode(AUTOMATIC);
352     rightPID.SetMode(AUTOMATIC);
353 }
```

#### 3.1.3.17 turnLeft()

```
void turnLeft (
            int degrees )
```

Turn bot right to given degrees.

**Parameters**

| | |
|---|---|
| *degrees* | |

Definition at line 178 of file custom_lab_3.ino.

```
179 {
180     resetPWM();
181     calculateDesiredCountTurn(degrees);
182 
183     // Loop unitl the encoders read correct
184 
185     while ((desiredCount - leftEncoderCount) > 3)
186     {
187         adjustPWM();
188         //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         }
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, 0);
202     run_motor(B, 0);
203     Serial.println("Done driving Left");
204     Serial.print("L: ");
205     Serial.println(leftEncoderCount);
206     Serial.print("R: ");
207     Serial.println(rightEncoderCount);
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
144     calculateDesiredCountTurn(degrees);
145     // While the encoders are not correct adjust PWM with PID loop
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 
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 
163     // motors stop
164     run_motor(A, 0);
165     run_motor(B, 0);
166     Serial.println("Done driving Right");
167     Serial.print("L: ");
168     Serial.println(leftEncoderCount);
169     Serial.print("R: ");
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.

```
2 {
3   // if using dual motor driver
4   // define driver pins as outputs
5   pinMode(IN1, OUTPUT);
6   pinMode(IN2, OUTPUT);
7   pinMode(IN3, OUTPUT);
8   pinMode(IN4, OUTPUT);
9   // initialize all pins to zero
10   digitalWrite(IN1, 0);
11   digitalWrite(IN2, 0);
12   digitalWrite(IN3, 0);
13   digitalWrite(IN4, 0);
14   return;
15 } // end function
```

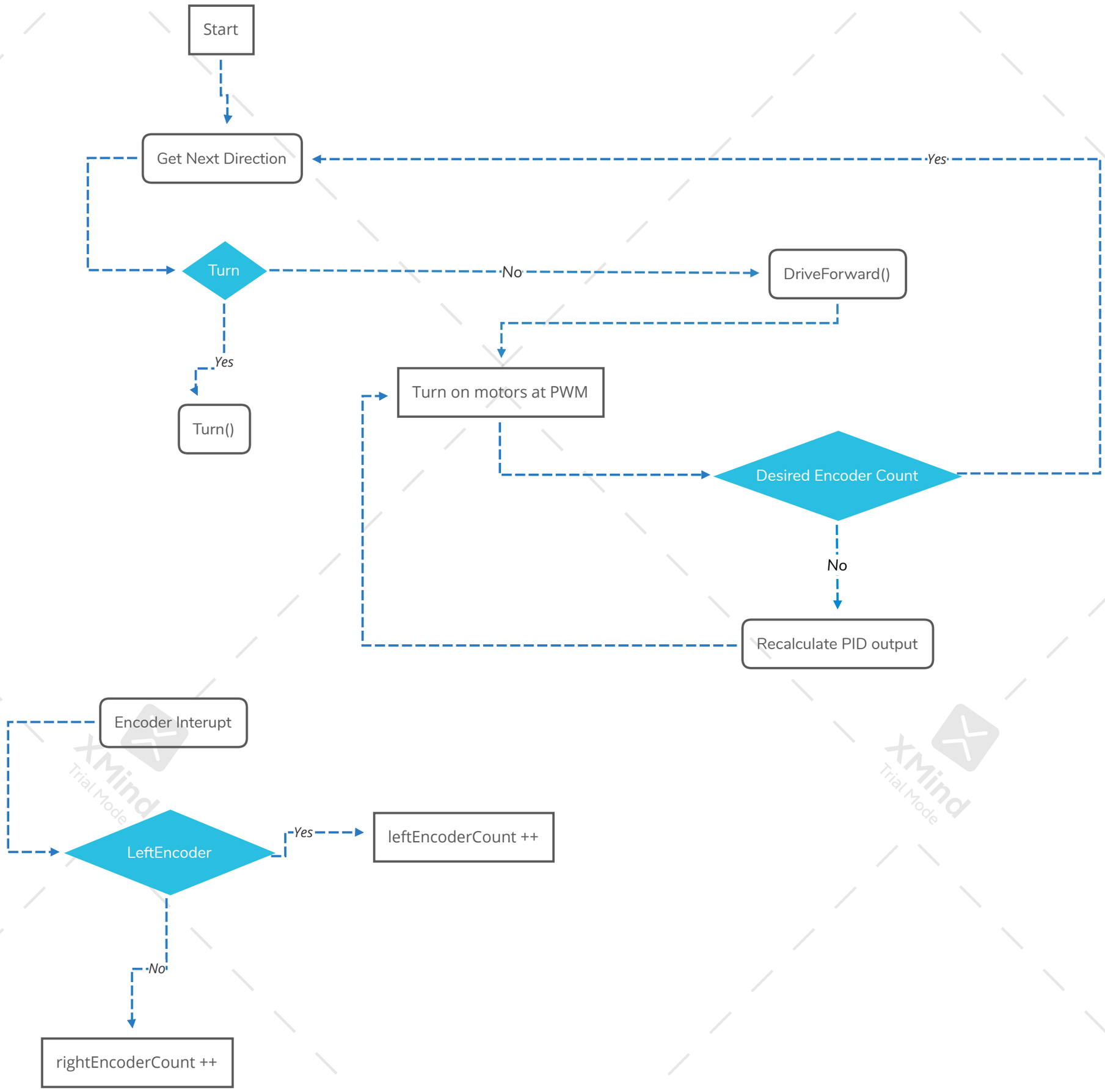
#### 3.2.1.2 run_motor()

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

Definition at line 19 of file motors.ino.

```
20 {
21   int dir = (pwm / abs(pwm)) > 0; // returns if direction is forward (1) or reverse (0)
22   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
29       analogWrite(IN1, pwm);  // IN1 is the forward pwm pin
30       digitalWrite(IN2, LOW); // IN2 is low
31     }
32     else
33     {
34       digitalWrite(IN1, LOW); // IN1 is low
35       analogWrite(IN2, pwm);  // IN2 is the reverse pwm pin
36     }                           // end if
37     break;                      // end case A
38   case B:                       // if B, write B pins
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     break;                      // end case B
50   }                             // end switch case
51   return;
52 } // end function
```

# Lab 3




XMind | Trial Mode

```
/**
   @file custom_lab_3.ino
   @author Christian Prather
   @brief A basic feedback controlled system for an Arduino based robot
   @version 0.1
   @date 2020-10-21

*/

/*! \mainpage Lab 3 Code Documentation
 *
 */


/// Libraries for interrupts and PID
#include <PinChangeInt.h>
#include <PID_v1.h>

/// Global Defines

/// Motor driver connections
#define IN1 9
#define IN2 10
#define IN3 5
#define IN4 6

/// Motor control
#define A 0
#define B 1
#define pwmA 3
#define dirA 12
#define pwmB 11
#define dirB 13

/// Start stop button
#define pushButton 2

/// Drive constants - dependent on robot configuration
#define EncoderCountsPerRev 12.0
#define DistancePerRev 51.0
#define DegreesPerRev 27.0

#define EncoderMotorLeft 7
#define EncoderMotorRight 8

/// Lab specific variables
double leftEncoderCount = 0;
double rightEncoderCount = 0;

/// Enum defines
#define FORWARD 0
#define LEFT 1
#define RIGHT -1

/// Default motor pwm values
int motorLeft_PWM = 180;
int motorRight_PWM = 200;

/// Time it takes to move 90 degrees
int milliSecondsPer90Deg = 900;
```

```
61
62 /// How many encoder counts for given distance
63 double desiredCount;
64
65 // Global array for tracking move order (move, distance) or (move, degree)
66 int moveList[] = {FORWARD, 300, LEFT, 90, FORWARD, 300, LEFT, 90, FORWARD,
   300, RIGHT, 90, FORWARD, 900, RIGHT, 90, FORWARD, 600, RIGHT, 90, FORWARD,
   300};
67
68 /**
69    @brief PID values
70    setpoints = desired counts, output = PWM, input = current counts
71 */
72 double leftOutput, rightOutput;
73 PID leftPID(&leftEncoderCount, &leftOutput, &desiredCount, 2, 5, 2, DIRECT);
74 PID rightPID(&rightEncoderCount, &rightOutput, &desiredCount, 2, 5, 2,
   DIRECT);
75
76 /**
77    @brief Helper function for setting the PWM back to default value
78 */
79 void resetPWM()
80 {
81     motorLeft_PWM = 180;
82     motorRight_PWM = 200;
83 }
84
85 /**
86    @brief ISR for left encoder
87 */
88 void indexLeftEncoderCount()
89 {
90     leftEncoderCount++;
91     //Serial.println("Left Encoder ++");
92 }
93
94 /**
95    @brief ISR for incrementing right encoder
96 */
97 void indexRightEncoderCount()
98 {
99     rightEncoderCount++;
100    //Serial.println("Right Encoder ++");
101 }
102
103 /**
104    @brief Calculate how many encoder counts we expect given the distance
   provided
105    based on the bot intrinsics
106
107    @param distance
108 */
109 void calculateDesiredCount(int distance)
110 {
111     double revolutionsRequired = distance / DistancePerRev;
112
113     desiredCount = revolutionsRequired * EncoderCountsPerRev;
114     // Reset encoder counts
115     leftEncoderCount = 0;
116     rightEncoderCount = 0;
```

```
117     Serial.print("Desired Count: ");
118     Serial.println(desiredCount);
119 }
120 
121 /**
122  * @brief Calculate how many encoder counts we expect given the degrees
    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);
145     // While the encoders are not correct adjust PWM with PID loop
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 
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 
163     // motors stop
164     run_motor(A, 0);
165     run_motor(B, 0);
166     Serial.println("Done driving Right");
167     Serial.print("L: ");
168     Serial.println(leftEncoderCount);
169     Serial.print("R: ");
170     Serial.println(rightEncoderCount);
171 }
172 
173 /**
174    @brief Turn bot right to given degrees
175 
```

```
   @param degrees
*/
void turnLeft(int degrees)
{
    resetPWM();
    calculateDesiredCountTurn(degrees);

    // Loop unitl the encoders read correct

    while ((desiredCount - leftEncoderCount) > 3)
    {
        adjustPWM();
        //To drive forward, motors go in the same direction

        if ((desiredCount - leftEncoderCount) > 3)
        {
            run_motor(A, motorLeft_PWM); //change PWM to your calibrations
        }
        if ((desiredCount - rightEncoderCount) > 3)
        {
            run_motor(B, -motorRight_PWM); //change PWM to your calibrations
        }
    }

    // motors stop
    run_motor(A, 0);
    run_motor(B, 0);
    Serial.println("Done driving Left");
    Serial.print("L: ");
    Serial.println(leftEncoderCount);
    Serial.print("R: ");
    Serial.println(rightEncoderCount);
}

/**
   @brief Function to drive bot forward until encoders are within range

   @param distance
*/
void driveForward(int distance)
{
    Serial.println("Driving Forward...");
    resetPWM();
    calculateDesiredCount(distance);

    // Loop unitl the encoders read correct

    while ((desiredCount - leftEncoderCount) > 3 || (desiredCount -
rightEncoderCount) > 3)
    {
        adjustPWM();
        //To drive forward, motors go in the same direction

        if ((desiredCount - leftEncoderCount) > 3)
        {
            run_motor(A, -motorLeft_PWM); //change PWM to your calibrations
        }
        if ((desiredCount - rightEncoderCount) > 3)
        {
            run_motor(B, -motorRight_PWM); //change PWM to your calibrations
```

```
        }
    }

    // motors stop
    run_motor(A, 0);
    run_motor(B, 0);
    Serial.println("Done driving forward");
    Serial.print("L: ");
    Serial.println(leftEncoderCount);
    Serial.print("R: ");
    Serial.println(rightEncoderCount);
}

/**
   @brief Drive the bot backwards

   @param distance
*/
void driveBackward(int distance)
{
    resetPWM();
    calculateDesiredCount(distance);

    // Loop unitl the encoders read correct

    while ((desiredCount - leftEncoderCount) > 3 || (desiredCount - rightEncoderCount) > 3)
    {
        adjustPWM();
        //To drive backward, motors go in the same direction

        if ((desiredCount - leftEncoderCount) > 3)
        {
            run_motor(A, motorLeft_PWM); //change PWM to your calibrations
        }
        if ((desiredCount - rightEncoderCount) > 3)
        {
            run_motor(B, motorRight_PWM); //change PWM to your calibrations
        }
    }

    // motors stop
    run_motor(A, 0);
    run_motor(B, 0);
    Serial.println("Done driving backwards");
    Serial.print("L: ");
    Serial.println(leftEncoderCount);
    Serial.print("R: ");
    Serial.println(rightEncoderCount);
}

/**
   @brief Function for configuration of pin states and interrupts
*/
void configure()
{
    // set up the motor drive ports
    pinMode(pwmA, OUTPUT);
    pinMode(dirA, OUTPUT);
    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 }
304
305 /**
306    @brief Default behavior when not driving, waits for the pushButton to
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 /**
321    @brief Run the PID loop calculation and set out put to motors output in
    PWM
322
323 */
324 void adjustPWM()
325 {
326     // Compute the pid values
327     leftPID.Compute();
328     rightPID.Compute();
329
330     // Set the pid values within range
331     motorLeft_PWM = constrain(leftOutput, 150, 250);
332     motorRight_PWM = constrain(rightOutput, 150, 235);
333     Serial.print("Left PWM: ");
334     Serial.print(motorLeft_PWM);
335     Serial.print(" ");
336     Serial.println(leftEncoderCount);
337     Serial.print("Right PWM: ");
338     Serial.print(motorRight_PWM);
339     Serial.print(" ");
340     Serial.println(rightEncoderCount);
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();
```

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

```
410 {
411     drive();
412 }
413
```

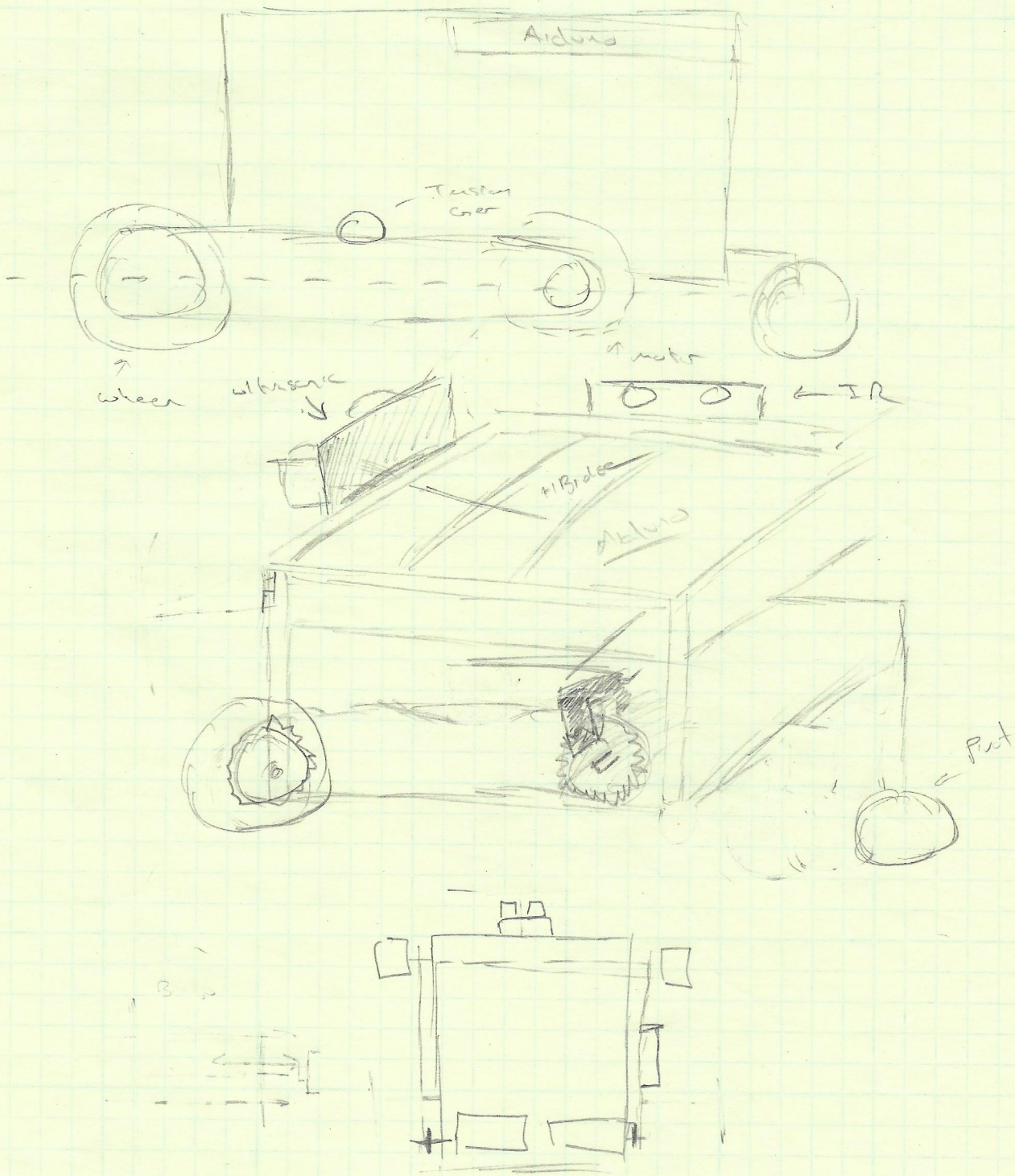
```
void motor_setup()
{
  // if using dual motor driver
  // define driver pins as outputs
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  // initialize all pins to zero
  digitalWrite(IN1, 0);
  digitalWrite(IN2, 0);
  digitalWrite(IN3, 0);
  digitalWrite(IN4, 0);
  return;
} // end function

// int motor is the defined A or B
// pwm = the power cycle you want to use
void run_motor(int motor, int pwm)
{
  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

  switch (motor)
  {       // find which motor to control
  case A: // if A, write A pins
    if (dir)
    {                           // If dir is forward
      analogWrite(IN1, pwm);  // IN1 is the forward pwm pin
      digitalWrite(IN2, LOW); // IN2 is low
    }
    else
    {
      digitalWrite(IN1, LOW); // IN1 is low
      analogWrite(IN2, pwm);  // IN2 is the reverse pwm pin
    }                           // end if
    break;                      // end case A
  case B:                       // if B, write B pins
    if (dir)
    {                           // if dir is forward
      analogWrite(IN3, pwm);  // IN3 is the forward pwm pin
      digitalWrite(IN4, LOW); // IN4 is low
    }
    else
    {
      digitalWrite(IN3, LOW); //IN3 is low
      analogWrite(IN4, pwm);  // IN4 is the reverse pwm pin
    }                           // end if
    break;                      // end case B
  }                             // end switch case
  return;
} // end function

```

Mechanical


Arduino
Tusion Gear
motor
wheel
ultrasonic
IR
H Bridge
Arduino
Pivot

330 = +
GND = E

D = GND
+ = 10k

1 4
2 3

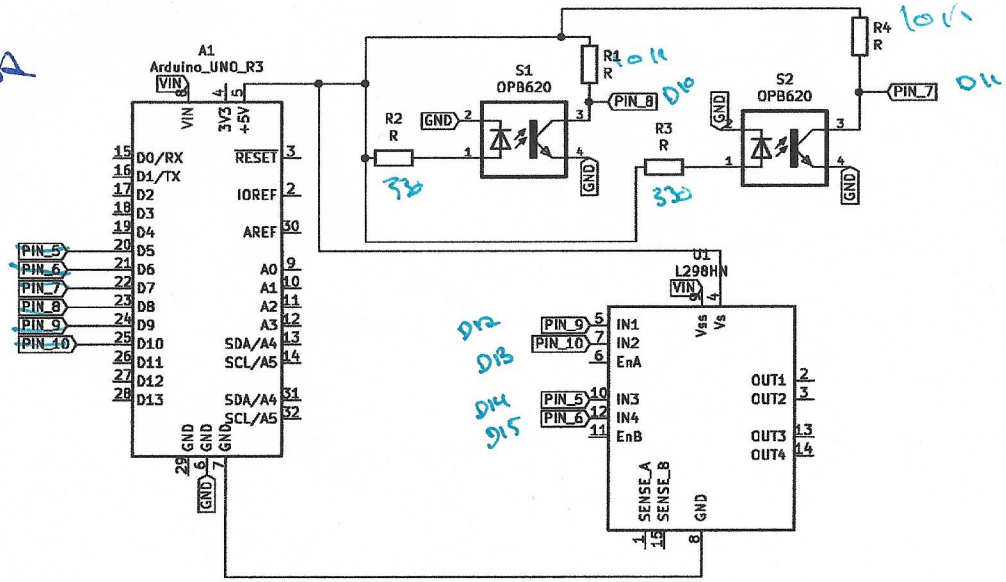
Default = 0 = empty

2 + called D 3 collector
1 E Anode + 4 emitter

Smiling Duno Interrupt works!

16 – 31

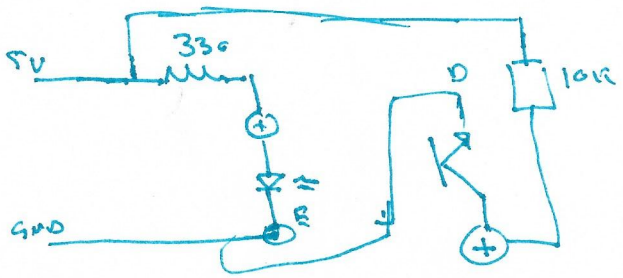
48 49 50 51
32 33 34 35




Bump switch?

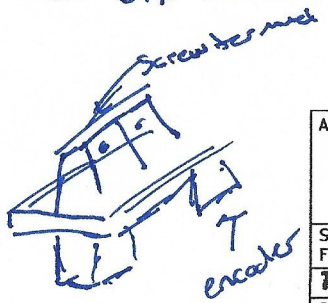
Don't need
Interrupt just
need to check
before every
drive

Switch




- Add screw terminals for more sensors
- Add Dip switch for Pin connection?

- Design a board for encoder bs



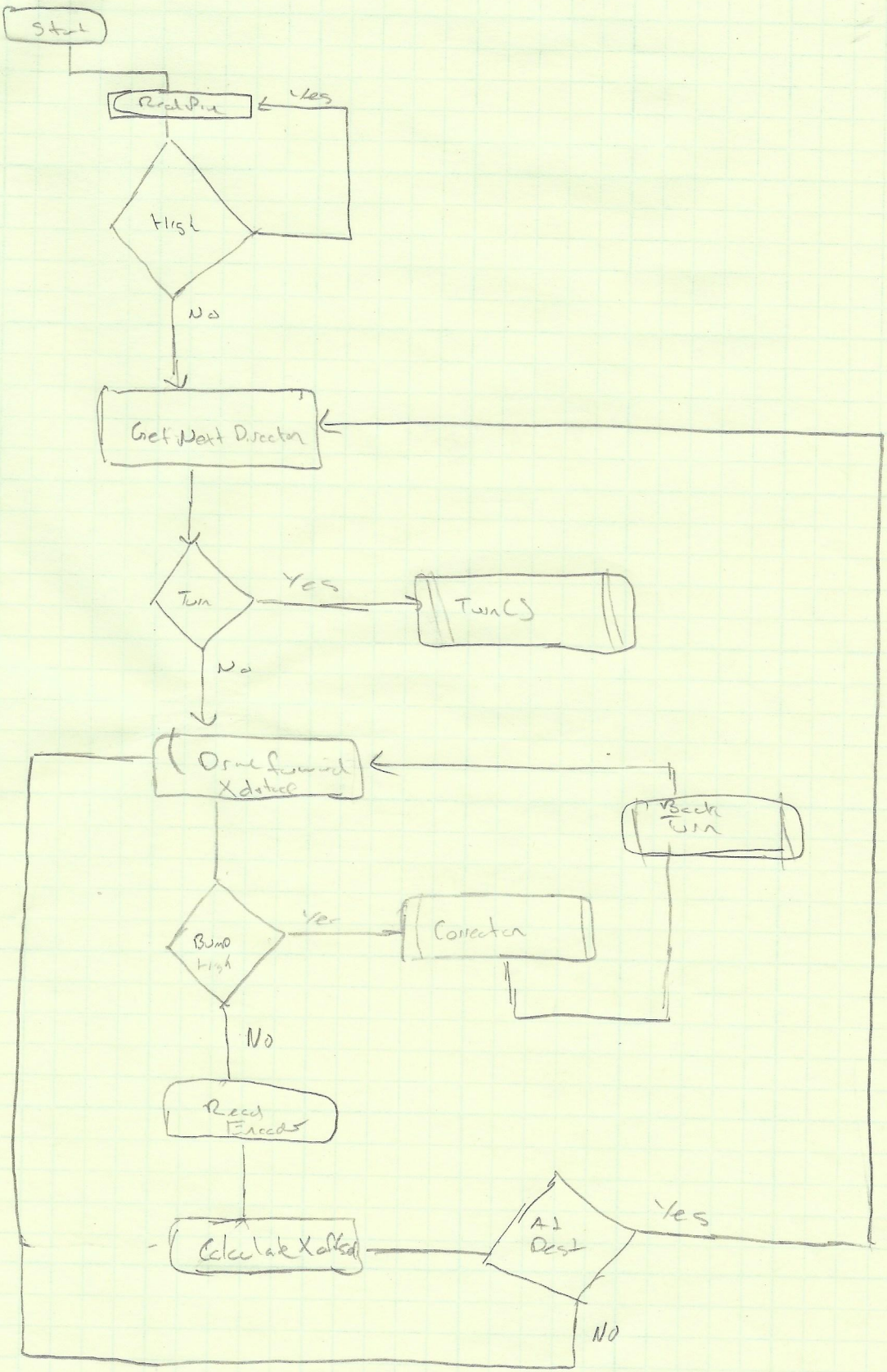

Author: Christian Prather

Sheet: /
File: robot.sch

**Title: Lab 3 Robot**

| | | |
|---|---|---|
| Size: A4 | Date: 2020-10-19 | Rev: Version 3 |
| KiCad E.D.A. kicad 5.1.7-a382d34a8B7ubuntu20.04.1 | | Id: 1/1 |

Flow:


Start
Read Pin
Yes
High
No
Get Next Direction
Turn
Yes
Turn(S)
No
Orientate Xdetect
Back Turn
Bump High
Yes
Correction
No
Read Encoders
Calculate X offset
At Dest
Yes
No

MS