Colorado School of Mines

Memo

To: Instructor and TA

From: Christian Prather

Team #: M420

Date: 10-22-20

Re: Lab 3: Closed loop control

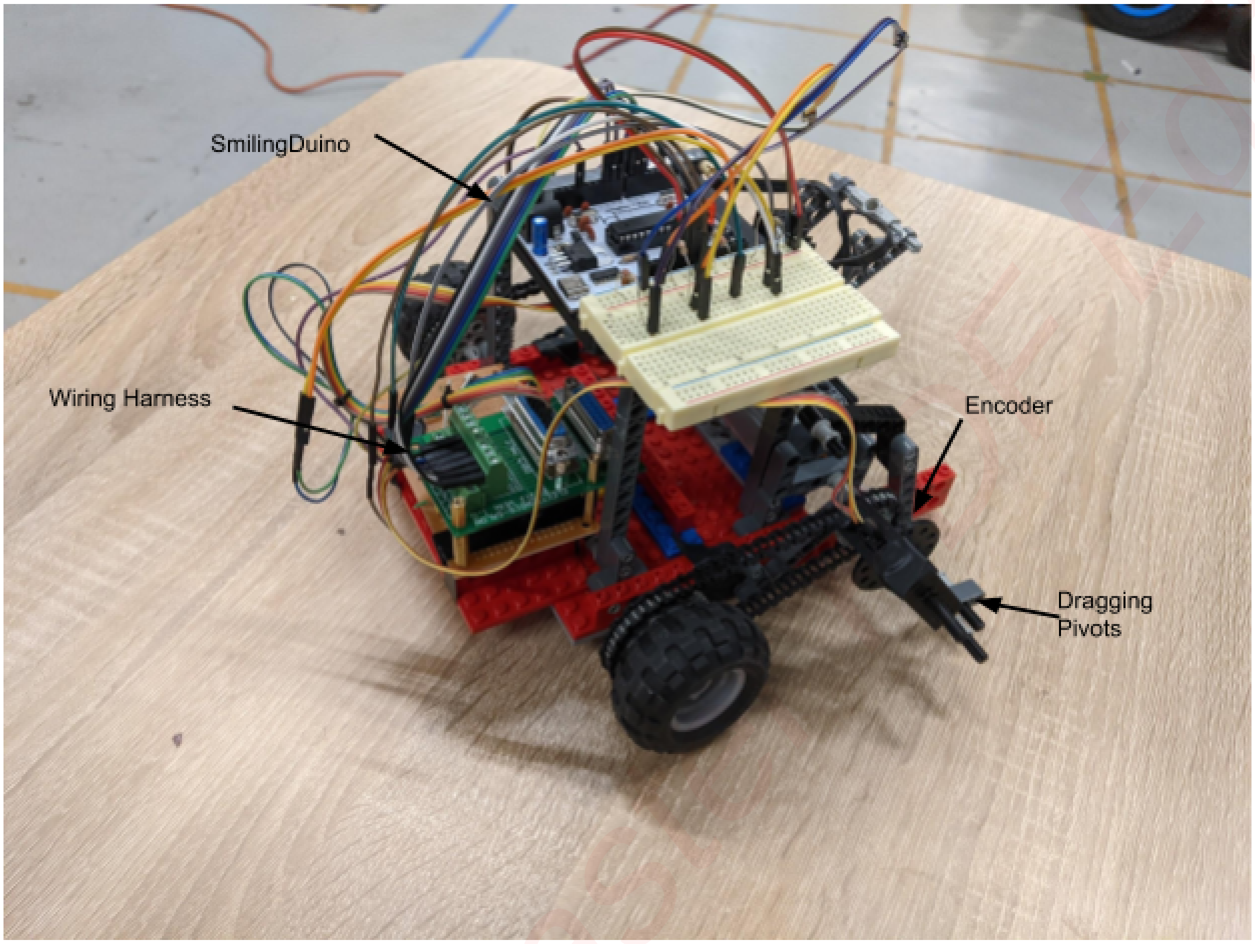
Problem Statement:

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:

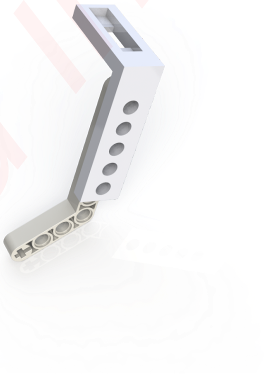
Sensors:

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.



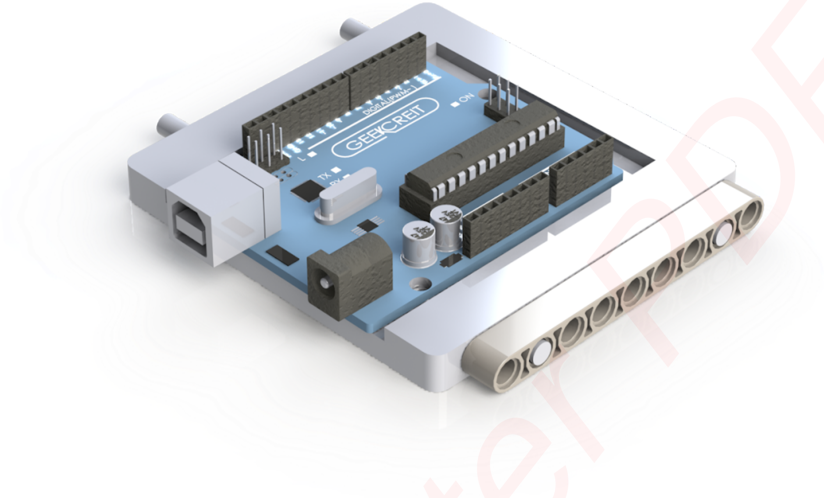
(Fig 1)

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:

Chapter 3

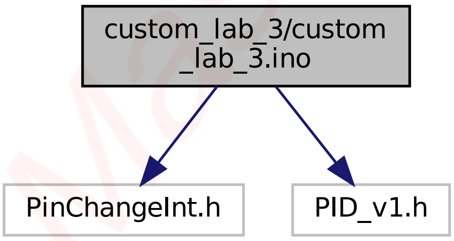
File Documentation

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.

```
{
    // Compute the pid values
    leftPID.Compute();
    rightPID.Compute();

    // Set the pid values within range
    motorLeft_PWM = constrain(leftOutput, 150, 250);
    motorRight_PWM = constrain(rightOutput, 150, 235);
    Serial.print("Left PWM: ");
    Serial.print(motorLeft_PWM);
    Serial.print(" ");
    Serial.println(leftEncoderCount);
    Serial.print("Right PWM: ");
    Serial.print(motorRight_PWM);
    Serial.print(" ");
    Serial.println(rightEncoderCount);
}
```

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.

```
{
    double revolutionsRequired = distance / DistancePerRev;

    desiredCount = revolutionsRequired * EncoderCountsPerRev;
    // Reset encoder counts
    leftEncoderCount = 0;
    rightEncoderCount = 0;
    Serial.print("Desired Count: ");
    Serial.println(desiredCount);
}
```

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.

```
{
    double revolutionsRequired = degrees / DegreesPerRev;
    desiredCount = revolutionsRequired * EncoderCountsPerRev;
    leftEncoderCount = 0;
    rightEncoderCount = 0;
    Serial.print("Desired Count: ");
    Serial.println(desiredCount);
}
```

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.

```
{
    // set up the motor drive ports
    pinMode(pwmA, OUTPUT);
    pinMode(dirA, OUTPUT);
    pinMode(pwmB, OUTPUT);
    pinMode(dirB, OUTPUT);

    pinMode(pushButton, INPUT_PULLUP);

    pinMode(EncoderMotorLeft, INPUT_PULLUP); //set the pin to input
    PCintPort::attachInterrupt(EncoderMotorLeft, indexLeftEncoderCount, CHANGE);

    pinMode(EncoderMotorRight, INPUT_PULLUP); //set the pin to input
    PCintPort::attachInterrupt(EncoderMotorRight, indexRightEncoderCount, CHANGE);
}
```

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.

```
{
    // Iterate over the list jumping by two each time
    for (int i = 0; i < sizeof(moveList); i += 2)
    {
        idle();
        switch (moveList[i])
        {
        case LEFT:
            turnLeft(moveList[i + 1]);
            break;
        case RIGHT:
            turnRight(moveList[i + 1]);
            break;
        case FORWARD:
            driveForward(moveList[i + 1]);
            break;
        default:
            break;
        }
    }
}
```

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.

```
{
    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);
}
```

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.

```
{
    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);
}
```

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.

```
{
    Serial.println("Idle..");
    while (digitalRead(pushButton) == 1)
        ; // wait for button push
    while (digitalRead(pushButton) == 0)
        ; // wait for button release
    delay(2000); // Give time to move hand
}
```

3.1.3.9 indexLeftEncoderCount()

void indexLeftEncoderCount ( )

ISR for left encoder.

Definition at line 88 of file custom_lab_3.ino.

```
{
    leftEncoderCount++;
    //Serial.println("Left Encoder ++");
}
```

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.

```
{
    // TODO: Check which button was hit

    driveBackward(20);
    turnLeft(30);
}
```

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.

```
{
    motorLeft_PWM = 180;
    motorRight_PWM = 200;
}
```

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.

```
{
    Serial.begin(9600);
    Serial.println("Setting up.....");
    configure();
    leftPID.SetMode(AUTOMATIC);
    rightPID.SetMode(AUTOMATIC);
}
```

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.

```
{
    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);
}
```

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.

```
{
    resetPWM(); // Reset pwm
    calculateDesiredCountTurn(degrees);
    // While the encoders are not correct adjust PWM with PID loop
    // Loop unitl the encoders read correct

    while ((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 Right");
    Serial.print("L: ");
    Serial.println(leftEncoderCount);
    Serial.print("R: ");
    Serial.println(rightEncoderCount);
}
```

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
  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
```

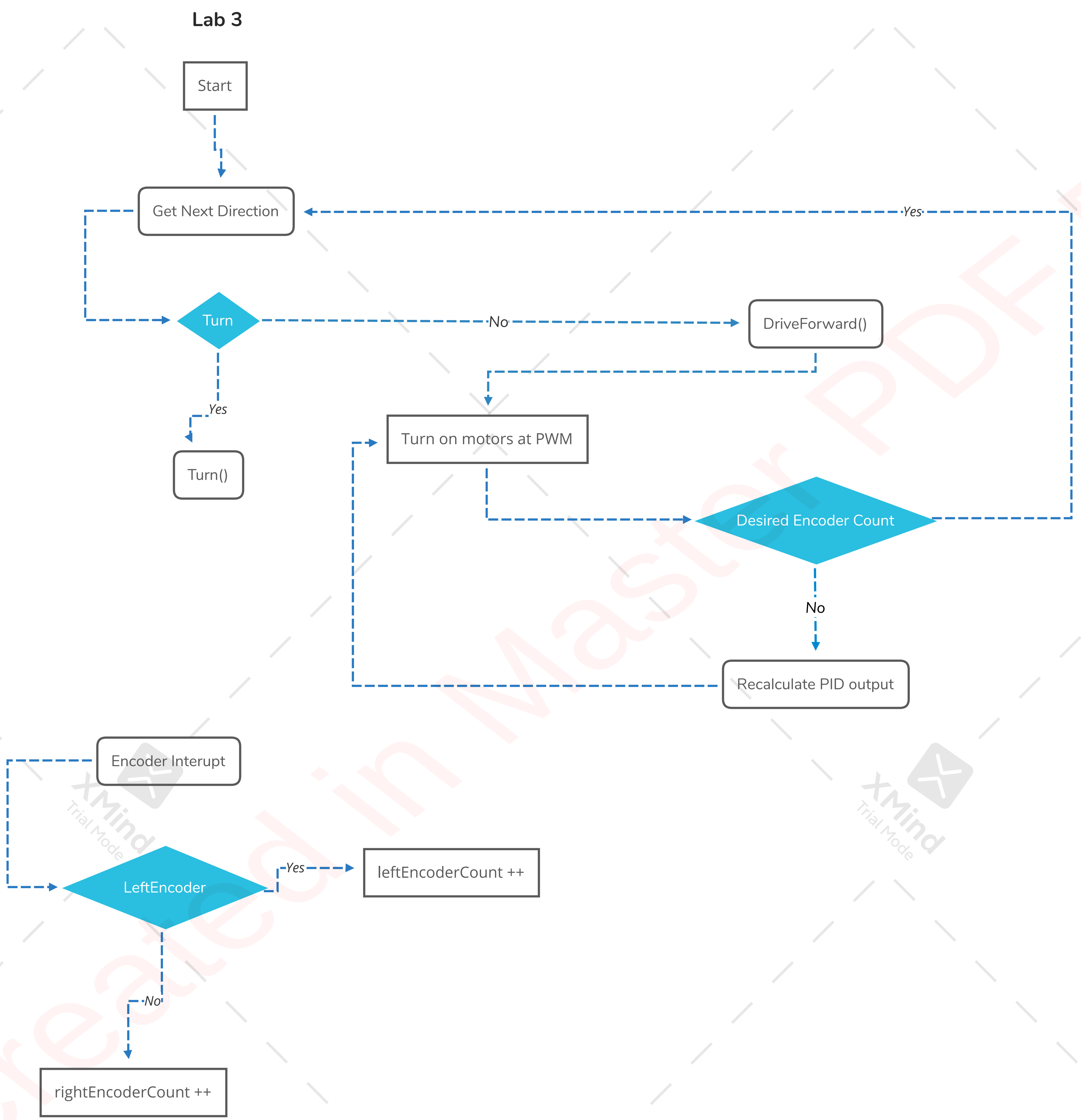
3.2.1.2 run_motor()

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

Definition at line 19 of file motors.ino.

```
{
  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
```


Lab 3
Start
Get Next Direction
Yes
Turn
No
DriveForward()
Yes
Turn()
Turn on motors at PWM
Desired Encoder Count
No
Recalculate PID output
Encoder Interupt
LeftEncoder
Yes
leftEncoderCount ++
No
rightEncoderCount ++

```
/**
   @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;
```

```

/// How many encoder counts for given distance
double desiredCount;

// Global array for tracking move order (move, distance) or (move, degree)
int moveList[] = {FORWARD, 300, LEFT, 90, FORWARD, 300, LEFT, 90, FORWARD, 300, RIGHT, 90, FORWARD, 900, RIGHT, 90, FORWARD, 600, RIGHT, 90, FORWARD, 300};

/**
   @brief PID values
   setpoints = desired counts, output = PWM, input = current counts
*/
double leftOutput, rightOutput;
PID leftPID(&leftEncoderCount, &leftOutput, &desiredCount, 2, 5, 2, DIRECT);
PID rightPID(&rightEncoderCount, &rightOutput, &desiredCount, 2, 5, 2, DIRECT);

/**
   @brief Helper function for setting the PWM back to default value
*/
void resetPWM()
{
    motorLeft_PWM = 180;
    motorRight_PWM = 200;
}

/**
   @brief ISR for left encoder
*/
void indexLeftEncoderCount()
{
    leftEncoderCount++;
    //Serial.println("Left Encoder ++");
}

/**
   @brief ISR for incrementing right encoder
*/
void indexRightEncoderCount()
{
    rightEncoderCount++;
    //Serial.println("Right Encoder ++");
}

/**
   @brief Calculate how many encoder counts we expect given the distance provided
   based on the bot intrinsics

   @param distance
*/
void calculateDesiredCount(int distance)
{
    double revolutionsRequired = distance / DistancePerRev;

    desiredCount = revolutionsRequired * EncoderCountsPerRev;
    // Reset encoder counts
    leftEncoderCount = 0;
    rightEncoderCount = 0;
    Serial.print("Desired Count: ");
    Serial.println(desiredCount);
}

/**
 * @brief Calculate how many encoder counts we expect given the degrees provided
 *
 * @param degrees
 */
void calculateDesiredCountTurn(int degrees)
{
    double revolutionsRequired = degrees / DegreesPerRev;
    desiredCount = revolutionsRequired * EncoderCountsPerRev;
    leftEncoderCount = 0;
    rightEncoderCount = 0;
    Serial.print("Desired Count: ");
    Serial.println(desiredCount);
}

/**
   @brief Turn bot to given degrees

   @param degrees
*/
void turnRight(int degrees)
{
    resetPWM(); // Reset pwm
    calculateDesiredCountTurn(degrees);
    // While the encoders are not correct adjust PWM with PID loop
    // Loop unitl the encoders read correct

    while ((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 Right");
    Serial.print("L: ");
    Serial.println(leftEncoderCount);
    Serial.print("R: ");
    Serial.println(rightEncoderCount);
}

/**
   @brief Turn bot right to given degrees

   @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);
    pinMode(dirB, OUTPUT);

    pinMode(pushButton, INPUT_PULLUP);

    pinMode(EncoderMotorLeft, INPUT_PULLUP); //set the pin to input
    PCintPort::attachInterrupt(EncoderMotorLeft, indexLeftEncoderCount, CHANGE);

    pinMode(EncoderMotorRight, INPUT_PULLUP); //set the pin to input
    PCintPort::attachInterrupt(EncoderMotorRight, indexRightEncoderCount, CHANGE);
}

/**
   @brief Default behavior when not driving, waits for the pushButton to
   be pressed so it can execute next command
   Blocking function
*/
void idle()
{
    Serial.println("Idle..");
    while (digitalRead(pushButton) == 1)
        ; // wait for button push
    while (digitalRead(pushButton) == 0)
        ; // wait for button release
    delay(2000); // Give time to move hand
}

/**
   @brief Run the PID loop calculation and set out put to motors output in PWM

*/
void adjustPWM()
{
    // Compute the pid values
    leftPID.Compute();
    rightPID.Compute();

    // Set the pid values within range
    motorLeft_PWM = constrain(leftOutput, 150, 250);
    motorRight_PWM = constrain(rightOutput, 150, 235);
    Serial.print("Left PWM: ");
    Serial.print(motorLeft_PWM);
    Serial.print(" ");
    Serial.println(leftEncoderCount);
    Serial.print("Right PWM: ");
    Serial.print(motorRight_PWM);
    Serial.print(" ");
    Serial.println(rightEncoderCount);
}

/**
   @brief Entry point of program handles serial setup and PID config
*/
void setup()
{
    Serial.begin(9600);
    Serial.println("Setting up.....");
    configure();
    leftPID.SetMode(AUTOMATIC);
    rightPID.SetMode(AUTOMATIC);
}

/**
   @brief 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_left()
{
    // TODO: Check which button was hit

    driveBackward(20);
    turnRight(30);
}
void react_right()
{
    // TODO: Check which button was hit

    driveBackward(20);
    turnLeft(30);
}
void react_forward()
{
    // TODO: Check which button was hit
    driveBackward(50);
}

/**
   @brief Main drive execution of program, iterates through moves list executing
   next move with corresponding distance or degrees
*/
void drive()
{
    // Iterate over the list jumping by two each time
    for (int i = 0; i < sizeof(moveList); i += 2)
    {
        idle();
        switch (moveList[i])
        {
        case LEFT:
            turnLeft(moveList[i + 1]);
            break;
        case RIGHT:
            turnRight(moveList[i + 1]);
            break;
        case FORWARD:
            driveForward(moveList[i + 1]);
            break;
        default:
            break;
        }
    }
}

/**
   @brief Loop execution of the program
*/
void loop()
```

```
{
    drive();
}

```

```
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

```