```
/**
   @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();
}

```