```cpp
#include <FEHLCD.h>
#include <FEHIO.h>
#include <FEHUtility.h>
#include <FEHMotor.h>
#include <FEHRPS.h>
#include <FEHServo.h>
#include <FEHSD.h>


// class for PID
class Adjust {
    public:
        Adjust(float a = 0, int b = 0);

        void rForward(float distance, float Speed);
        void rReverse(float distance, float Speed);
        void turnRIGHT(float degrees, float Speed);
        void turnLEFT(float degrees, float Speed);
        void resetPID();
        float RPID(float currenttime, float expectedvelocity, int i);

        float oldTime[2], oldError[2], errorSum[2], oldMotorPower[2];
        int oldCounts[2];
        float startTime;
};

// Front bump switch
DigitalInputPin frontBump(FEHIO::P1_7);

// Declare CDS Cell
// Declare Magnet
AnalogInputPin cdsCell(FEHIO::P3_7);
DigitalOutputPin magnet(FEHIO::P3_0);

// Declare Left Motor
// Declare Right Motor
// Declare Crank Servo
// Declare Wrench Servo

FEHMotor left_drive(FEHMotor::Motor0,7.2);
FEHMotor right_drive(FEHMotor::Motor1,7.2);
FEHServo crankServo(FEHServo::Servo7);
FEHServo wrenchServo(FEHServo::Servo0);

// declare encoders
DigitalEncoder left_encoder(FEHIO::P2_0);
DigitalEncoder right_encoder(FEHIO::P1_0);


// function constructors
void stopAll();
void resetCounts();
void check_x_plus(float x_coordinate);
void check_x_minus(float x_coordinate);
void check_y_plus(float y_coordinate);
void check_y_minus(float y_coordinate);
void check_heading(float heading);
```

```
void check_heading_time(float heading, float timeout);
void check_heading360();
void calibrateCrank(int turn);
void returnCoords();
float lightSpot();

// constant variable declarations
#define PI 3.14159

#define Pconst 0.50
#define Iconst 0.04
#define Dconst 0.25

// beginning of main function
int main(void)
{
    // initialize floats for touch screen
    float touch_x,touch_y;

    // select RPS region
    RPS.InitializeTouchMenu();

    // initialize variable for current time
    float startTime = TimeNow();

    // reset screen
    LCD.Clear( FEHLCD::Black );
    LCD.SetFontColor( FEHLCD::White );
    LCD.SetBackgroundColor(BLACK);

    // open SD card log for logging
    SD.OpenLog();
    SD.Printf("START LOG\n");

    // create an instance of the class Adjust for PID
    Adjust PID;

    // initialize x coordinate variable for button board
    float lightX = lightSpot();
    Sleep(1.0);

    // WAIT FOR START LIGHT
    bool wait = true;
    float lightThreshold = .70;
    int timeOut = 0;
     while ((wait) && (timeOut < 30)){

         // wait until the light turns on
         // OR 30 seconds have elapsed
        LCD.WriteLine(cdsCell.Value());
        if ((cdsCell.Value() < lightThreshold)){
            wait = false;
        }
        Sleep(.25);
        timeOut += .25;
        LCD.Clear();
    }
```

```
Sleep(.5);

// Turn Magnet ON
LCD.WriteLine("Magnet ON");
magnet.Write(true);

// initialize starting RPS coordinates
float startX = RPS.X();
float startY = RPS.Y();
// single point of control of the velocity used in most PID method calls
float vel = 30.0;

// DRIVE OUT OF START ZONE
PID.rForward(5.8, vel);
check_y_minus(startY - 6.1);

// SET WRENCH ARM UP
wrenchServo.SetMin(500);
wrenchServo.SetMax(2300);
wrenchServo.SetDegree(0);

// TURN TOWARDS CORNER
PID.turnRIGHT(35, vel);
check_heading(230);

// DRIVE INTO CORNER
PID.rForward(19.295, vel);

// TURN TOWARDS CAR JACK
PID.turnLEFT(115, vel);
check_heading360();

// DRIVE INTO CAR JACK
PID.rForward(4, 15);

// REVERSE FROM CAR JACK
PID.rReverse(3, 15);
check_x_plus(startX - 9.9);

// TURN LEFT TOWARDS START
PID.turnLEFT(35, vel);
check_heading(51);

// DRIVE TOWARDS START
PID.rForward(8.5, vel);
check_heading(51);

// TURN TOWARDS WRENCH
PID.turnLEFT(125, vel);
check_heading(180);

// DRIVE TOWARDS WRENCH
PID.rForward(1, vel);
check_x_minus(7.1);
```

```
//1st time
// LOWER WRENCH ARM
for (int k = 65; k < 85; k++){
    wrenchServo.SetDegree(k);
    Sleep(.3);
    k++;
}
Sleep(.4);

// SLOWLY RAISE ARM
for (int k = 85; k > 80; k--){
    wrenchServo.SetDegree(k);
    Sleep(.3);
    k--;
}
Sleep(.4);
LCD.WriteLine("Raising Arm");
int i = 5;
while (i < 80){
    Sleep(.1);
    wrenchServo.SetDegree(80 - i);
    i += 5;
}

//2nd try
// DRIVE TOWARDS WRENCH
PID.rReverse(.5, vel);
check_x_minus(7.5);

// LOWER ARM
for (int k = 65; k < 85; k++){
    wrenchServo.SetDegree(k);
    Sleep(.3);
    k++;
}
Sleep(.4);

// SLOWLY RAISE ARM
for (int k = 85; k > 80; k--){
    wrenchServo.SetDegree(k);
    Sleep(.3);
    k--;
}
Sleep(.4);
LCD.WriteLine("Raising Arm");
i = 5;
while (i < 80){
    Sleep(.1);
    wrenchServo.SetDegree(80 - i);
    i += 5;
}

//3rd try
// DRIVE TOWARDS WRENCH
PID.rReverse(.5, vel);
check_x_minus(8.1);
```

```
// LOWER ARM
for (int k = 65; k < 85; k++){
    wrenchServo.SetDegree(k);
    Sleep(.3);
    k++;
}
Sleep(.4);

//PID.rReverse(2.4, vel);

// SLOWLY RAISE ARM
for (int k = 85; k > 80; k--){
    wrenchServo.SetDegree(k);
    Sleep(.3);
    k--;
}
Sleep(.4);
LCD.WriteLine("Raising Arm");
i = 5;
while (i < 80){
    Sleep(.1);
    wrenchServo.SetDegree(80 - i);
    i += 5;
}

// REVERSE AWAY FROM WRENCH
PID.rReverse(1, 15);

// TURN Away from CAR JACK
PID.turnRIGHT(140, vel);
check_heading(51);

// Forward  INCHES TOWARDS START BUTTON
PID.rForward(3.5, 20);

// TURN Away from WRENCH
PID.turnRIGHT(40, vel);
check_heading360();
PID.turnLEFT(5,vel);

/*
 * LIGHT CODE
 */


// DRIVE OVER LIGHT
PID.rForward(9, vel);

// check if the robot is at the pre-defined light X coordinate
Sleep(.1);
check_x_plus(lightX);
check_heading360();

// determime which route to take
int route = 0;
 while (route == 0){
     Sleep(.25);
```

```
     LCD.Clear();
    LCD.WriteLine(cdsCell.Value());
    if ((cdsCell.Value() > 1.1)){
        route = 1;
        // BLUE
        LCD.WriteLine("BLUE");
        PID.rForward(1.8, 15);
    }else{
        route = 2;
        // RED
        LCD.WriteLine("RED");
        PID.rReverse(1.8, 15);
    }
}
 // write the final CDS value used to determine the route
 LCD.WriteLine(cdsCell.Value());

 // FACE BUTTON BOARD
 PID.turnRIGHT(90, 15);
 PID.check_heading(270);

 // DRIVE INTO BUTTON
 PID.rForward(3.0, 15);
 Sleep(.1);

 // BACK AWAY FROM BUTTON BOARD
 PID.rReverse(3.0, 15);

 // FACE Wrench
 PID.turnRIGHT(70, 15);
 check_heading(180);

 // DRIVE TOWARD FINAL BUTTON
 if (route==1){
     // BLUE
     LCD.WriteLine("BLUE");
     PID.rForward(8, vel);
 }else{
     route = 2;
     // RED
     LCD.WriteLine("RED");
     PID.rForward(5, vel);
 }

 // ensure robot is in proper position
 check_x_minus(startX);

 //turns toward button
 PID.turnRIGHT(87, vel);

 //drive into final button
 PID.rForward(20,vel);

 Sleep(1.0);

 // clear screen and close SD log
 LCD.Clear();
```

```
      SD.CloseLog();

    return 0;
}

// function to calibrate RPS coordinate before each run
float lightSpot(){


    float touch_x,touch_y;
    bool flag = true;

    // wait until screen is touched
    while (flag){
        if (LCD.Touch(&touch_x, &touch_y)){
         flag = false;
        }
        LCD.Clear();
        LCD.WriteLine("X:");
        LCD.WriteLine(RPS.X());
        LCD.WriteLine("Y:");
        LCD.WriteLine(RPS.Y());
        LCD.WriteLine("Heading:");
        LCD.WriteLine(RPS.Heading());
        LCD.WriteLine("PRESS TO SET LIGHT VALUE");
        Sleep(.5);
    }

    // return current RPS X coordinate
     return RPS.X();

}


// function to stop motors for both wheels
void stopAll()
{
    // stop both motors for both wheels
    left_drive.Stop();
    right_drive.Stop();
     Sleep(.2);
}

void resetCounts()
{
    // reset encoder counts for both motors
    right_encoder.ResetCounts();
    left_encoder.ResetCounts();
}

void check_x_plus(float x_coordinate)
//using RPS while robot is in the +x direction

{

    //check whether the robot is within an acceptable range
    while(RPS.X() < x_coordinate - 1 || RPS.X() > x_coordinate + 1)
```

```
    {
        LCD.Clear();
        LCD.WriteLine("CHECK X PLUS");
        LCD.WriteLine("Current X: ");
        LCD.WriteLine(RPS.X());
        LCD.WriteLine("Correcting to: ");
        LCD.WriteLine(x_coordinate);

        if(RPS.X() > x_coordinate)
        {

            //pulse the motors for a short duration in the correct direction
            right_drive.SetPercent(-15);
            left_drive.SetPercent(-15);
            Sleep(100);
            stopAll();
        }
        else if(RPS.X() < x_coordinate)
        {

            //pulse the motors for a short duration in the correct direction

            right_drive.SetPercent(15);
            left_drive.SetPercent(15);
            Sleep(100);
            stopAll();
        }
    }
}

void check_x_minus(float x_coordinate)
//using RPS while robot is in the -x direction

{
    //check whether the robot is within an acceptable range
    while(RPS.X() < x_coordinate - 1 || RPS.X() > x_coordinate + 1)
    {
        LCD.Clear();
        LCD.WriteLine("CHECK X MINUS");
        LCD.WriteLine("Current X: ");
        LCD.WriteLine(RPS.X());
        LCD.WriteLine("Correcting to: ");
        LCD.WriteLine(x_coordinate);

        if(RPS.X() > x_coordinate)
        {

            //pulse the motors for a short duration in the correct direction
            right_drive.SetPercent(20);
            left_drive.SetPercent(20);
            Sleep(100);
            stopAll();
        }
        else if(RPS.X() < x_coordinate)
        {

            //pulse the motors for a short duration in the correct direction
```

```
            right_drive.SetPercent(-20);
            left_drive.SetPercent(-20);
            Sleep(100);
            stopAll();
        }
    }
}

void check_y_plus(float y_coordinate)
//using RPS while robot is in the +y direction

    {

//check whether the robot is within an acceptable range
    while(RPS.Y() < y_coordinate - 1 || RPS.Y() > y_coordinate + 1)
    {

        LCD.Clear();
        LCD.WriteLine("CHECK Y PLUS");
        LCD.WriteLine("Current Y: ");
        LCD.WriteLine(RPS.Y());
        LCD.WriteLine("Correcting to: ");
        LCD.WriteLine(y_coordinate);

        if(RPS.Y() > y_coordinate)
        {

            //pulse the motors for a short duration in the correct direction
            right_drive.SetPercent(-20);
            left_drive.SetPercent(-20);
            Sleep(100);
            stopAll();
        }
        else if(RPS.Y() < y_coordinate)
        {

            //pulse the motors for a short duration in the correct direction
            right_drive.SetPercent(20);
            left_drive.SetPercent(20);
            Sleep(100);
            stopAll();
        }
    }
    }

void check_y_minus(float y_coordinate)
//using RPS while robot is in the -y direction

{

//check whether the robot is within an acceptable range
    while(RPS.Y() < y_coordinate - 1 || RPS.Y() > y_coordinate + 1)
    {

        LCD.Clear();
        LCD.WriteLine("CHECK Y MINUS");
        LCD.WriteLine("Current Y: ");
```

```
        LCD.WriteLine(RPS.Y());
        LCD.WriteLine("Correcting to: ");
        LCD.WriteLine(y_coordinate);

        if(RPS.Y() > y_coordinate)
        {

            //pulse the motors for a short duration in the correct direction
            right_drive.SetPercent(20);
            left_drive.SetPercent(20);
            Sleep(100);
            stopAll();
        }
        else if(RPS.Y() < y_coordinate)
        {

            //pulse the motors for a short duration in the correct direction
            right_drive.SetPercent(-20);
            left_drive.SetPercent(-20);
            Sleep(100);
            stopAll();
        }
    }
}


void check_heading(float heading)
// pulse slightly until robot is facing the correct direction
{

    float newHeading = 0;

    // how to handle headings on the verge of 0/360
    if (heading != -1){
        if((heading < 2 && heading > 0) || (heading < 360 && heading > 358))
        {
            // set heading as 5
            heading = 5;
            if(RPS.Heading() <= 2 && RPS.Heading() >= 0){
                newHeading = 5 + RPS.Heading();         }
            // set newHeading is current heading + 5
            else
            {
                // set newheading is 5 - 360 - current heading
                newHeading = 5 - (360-RPS.Heading());
            }
            while(newHeading < heading - 2 || newHeading > heading + 2)
            {
                if(newHeading > heading)
                {

                    //pulse the motors for a short duration in the correct
direction
                    right_drive.SetPercent(-15);
                    left_drive.SetPercent(15);
                    Sleep(100);
                    stopAll();
```

```
                }
                else if(newHeading < heading)
                {

                    //pulse the motors for a short duration in the correct
direction
                    right_drive.SetPercent(15);
                    left_drive.SetPercent(-15);
                    Sleep(100);
                    stopAll();
                }
                if(RPS.Heading() <= 2 && RPS.Heading() >= 0)
                {
                    newHeading = 5 + RPS.Heading();
                    // set new heading = 5 + current heading
                }
                else
                {
                    newHeading = 5 - (360-RPS.Heading());
                    // set newing = 5 - 360 - RPS heading
                }
            }
        }
        else
        {
            while(RPS.Heading() < heading - 2 || RPS.Heading() > heading + 2)
            {
                if(RPS.Heading() > heading)
                {
                    //pulse the motors for a short duration in the correct
direction
                    right_drive.SetPercent(-15);
                    left_drive.SetPercent(15);
                    Sleep(100);
                    stopAll();
                }
                else if(RPS.Heading() < heading)
                {
                    //pulse the motors for a short duration in the correct
direction
                    right_drive.SetPercent(15);
                    left_drive.SetPercent(-15);
                    Sleep(100);
                    stopAll();
                }
            }
        }
    }else{
        LCD.WriteLine("In Dead Zone!");
        // write that the robot is in the dead zone
    }
    Sleep(.1);
}

void check_heading360()
// more specialized RPS check to handle directions around 0/360
```

```
{

    // reset new heading
    float newHeading = 0;
    LCD.WriteLine("CHECKING 360");

    // set heading to actual heading
    float heading = RPS.Heading();

    // if the robot is facing to the left of 360/0
    if (heading != -1){
        if((heading < 90) &&( heading > 0))
        {

            newHeading = RPS.Heading();

            while((newHeading < 90)&&(newHeading > 0))
            {
                // needs to turn RIGHT
                newHeading = RPS.Heading();
                    //pulse the motors for a short duration in the correct
direction
                    right_drive.SetPercent(-20);
                    left_drive.SetPercent(20);
                    Sleep(200);
                    stopAll();
            }
        }else{
            {
                newHeading = RPS.Heading();

                // if the robot is facing to the right of 360/0
                while((newHeading > 270)&&(newHeading < 359))
                {
                    // needs to turn LEFT
                        //pulse the motors for a short duration in the
correct direction
                        right_drive.SetPercent(20);
                        left_drive.SetPercent(-20);
                        Sleep(200);
                        newHeading = RPS.Heading();
                        stopAll();
                }
        }
    }
    }else{
        LCD.WriteLine("In Dead Zone!");
    }
    Sleep(.1);

}

void calibrateCrank(int turn){
    // calibrate crank servo

    // set min/max
    crankServo.SetMin(917);
```

```
    crankServo.SetMax(2500);

    // set degree based on whether or not the crank needs to be turned
    // clockwise or counter clockwise
    if (turn == 1){
        crankServo.SetDegree(0);
        LCD.WriteLine("CALIBRATED: Turn 1");

    }else{
        crankServo.SetDegree(180);
        LCD.WriteLine("CALIBRATED: Turn 2");
    }
}


void returnCoords()
// return current coords and heading of robot's position
// NOTE: used during testing, not any offical runs
{

    float touch_x,touch_y;

    // keep updating until screen is pressed
     while(!LCD.Touch(&touch_x,&touch_y))
     {
        LCD.Clear();
        LCD.WriteLine("X:");
        LCD.WriteLine(RPS.X());
        LCD.WriteLine("Y:");
        LCD.WriteLine(RPS.Y());
        LCD.WriteLine("Heading:");
        LCD.WriteLine(RPS.Heading());
        LCD.WriteLine("PRESS TO EXIT");
        Sleep(.5);
    }
}


// CLASS CONSTRUCTOR
// and CLASS FUNCTIONS

Adjust::Adjust(float a, int b)
// initialize class variables
{
    // right motor
    oldError[0] = a;
    oldMotorPower[0] = a;
    oldTime[0] = a;
    errorSum[0] = a;
    oldCounts[0] = b;

    // left motor
    oldError[1] = a;
    oldMotorPower[1] = a;
    oldTime[1] = a;
    errorSum[1] = a;
    oldCounts[1] = b;
```

```
    startTime = TimeNow();
}


void Adjust::rForward(float distance, float Speed)
{
    float currentTime, dummy;
    bool flag;
    LCD.WriteLine("rFORWARD");

    // reset variables
    oldMotorPower[0] = 0;
    oldMotorPower[1] = 0;
    oldTime[0] = TimeNow();
    oldTime[1] = TimeNow();

    // double speed for faster driving
    Speed = Speed * 2;

    // call external reset function
    resetPID();
    flag = true;

    // while flag is true
    while (flag){
        // reset oldTime for both motors
        oldTime[0] = TimeNow();
        oldTime[1] = TimeNow();

        Sleep(.1);
        // set dummy to the average encounter counts converted into a
distance
        dummy = (((right_encoder.Counts() + left_encoder.Counts()) / 2.0 ) *
(((3.0*PI)/180.0)));

        // set flag to whether or not the encoder distance is less than
desired distance
        flag = (dummy < distance);
        currentTime = TimeNow();

        //go forward, setting the percent to the speed returned by the PID
function)
        right_drive.SetPercent(RPID(currentTime,Speed, 0));
        left_drive.SetPercent(RPID(currentTime,Speed, 1));
        }
    // stop all motors
    stopAll();
    Sleep(.1);
}

void Adjust::rReverse(float distance, float Speed)
{
    float currentTime, dummy;
    bool flag;
    LCD.WriteLine("rREVERSE");
```

```
    // reset variables
    oldMotorPower[0] = 0;
    oldMotorPower[1] = 0;
    oldTime[0] = TimeNow();
    oldTime[1] = TimeNow();

    // double speed for faster driving
    Speed = Speed * 2;

    // call external reset function
    resetPID();
    flag = true;

    // while flag is true
    while (flag){
        // reset oldTime for both motors
        oldTime[0] = TimeNow();
        oldTime[1] = TimeNow();

        Sleep(.1);
        // set dummy to the average encounter counts converted into a
distance
        dummy = (((right_encoder.Counts() + left_encoder.Counts()) / 2.0 ) *
(((3.0*PI)/180.0)));

        // set flag to whether or not the encoder distance is less than
desired distance
        flag = (dummy < distance);
        currentTime = TimeNow();

        //go in reverse, setting the percent to the speed returned by the PID
function)
        right_drive.SetPercent(-1 * RPID(currentTime,Speed, 0));
        left_drive.SetPercent(-1 * RPID(currentTime,Speed, 1));
        }
    // stop all motors
    stopAll();
    Sleep(.1);
}

void Adjust::turnRIGHT(float degrees, float Speed){

    float currentTime, dummy;
    bool flag;
    LCD.WriteLine("turnRIGHT");

    // convert desired distance based on input degrees
    float distance = degrees / 15;

    // reset motor powers and speeds
    oldMotorPower[0] = 0;
    oldMotorPower[1] = 0;
    oldTime[0] = TimeNow();
    oldTime[1] = TimeNow();

    // call external PID reset function
    resetPID();
```

```
    flag = true;

    // while robot has not turned far enough
    while (flag){
        // reset time variables
        oldTime[0] = TimeNow();
        oldTime[1] = TimeNow();

        // calculate theoretical distance traveled as dummy
        Sleep(.1);
        dummy = (((right_encoder.Counts() + left_encoder.Counts()) / 2.0 ) *
(((3.0*PI)/180.0)));

        // set flag to whether theoretical distance is less than traveled
distance
        flag = (dummy < distance);
        currentTime = TimeNow();

        //turn right with speed set to RPID function
        right_drive.SetPercent(-1 * RPID(currentTime,Speed, 0));
        left_drive.SetPercent(RPID(currentTime,Speed, 1));
        }
    // stop all motors
    stopAll();
    Sleep(.1);

}

void Adjust::turnLEFT(float degrees, float Speed){

    float currentTime, dummy;
    bool flag;
    LCD.WriteLine("turnLEFT");

    // convert desired distance based on input degrees
    float distance = degrees / 15;

    // reset motor powers and speeds
    oldMotorPower[0] = 0;
    oldMotorPower[1] = 0;
    oldTime[0] = TimeNow();
    oldTime[1] = TimeNow();

    // call external PID reset function
    resetPID();
    flag = true;

    // while robot has not turned far enough
    while (flag){
        // reset time variables
        oldTime[0] = TimeNow();
        oldTime[1] = TimeNow();

        // calculate theoretical distance traveled as dummy
        Sleep(.1);
        dummy = (((right_encoder.Counts() + left_encoder.Counts()) / 2.0 ) *
(((3.0*PI)/180.0)));
```

```
        // set flag to whether theoretical distance is less than traveled
distance
        flag = (dummy < distance);
        currentTime = TimeNow();

        //turn left with speed set to RPID function
        right_drive.SetPercent(RPID(currentTime,Speed, 0));
        left_drive.SetPercent(-1 * RPID(currentTime,Speed, 1));
        }
    // stop all motors
    stopAll();
    Sleep(.1);
}


void Adjust::resetPID()
// reset PID variables between each method call
{
    //reset variables
    //record time
    //reset encoders
    right_encoder.ResetCounts();
    left_encoder.ResetCounts();
    float a = 0;
    int b = 0;

    // reset right motor
    oldError[0] = a;
    oldMotorPower[0] = a;
    oldTime[0] = a;
    oldCounts[0] = b;

    // reset left left motor
    oldError[1] = a;
    oldMotorPower[1] = a;
    oldTime[1] = a;
    oldCounts[1] = b;

    startTime = TimeNow();


    //wait
    Sleep(.1);
}

float Adjust::RPID(float currentTime, float expectedVelocity, int i)
// adjust speed to ensure linear driving and/or account for differences
// between motors
{
    int dcounts;
    float dtime, velocity, PTerm, ITerm, DTerm, error;
    //dtime is delta time
    dtime=currentTime - oldTime[i];

    if (i == 0){
        dcounts=right_encoder.Counts() - oldCounts[i];
```

```
        SD.Printf("||||| RIGHT WHEEL |||||\n");
    }else{
        dcounts=left_encoder.Counts() - oldCounts[i];
        SD.Printf("||||| LEFT WHEEL  |||||\n");
    }


    //Calculate velocity using time and encoder counts
    SD.Printf("Dcounts: %d\n",dcounts);
    SD.Printf("Dtime: %f\n",dtime);
    velocity= (((3*PI)/90))*(dcounts/dtime);
    error = expectedVelocity - velocity;
    SD.Printf("Velocity: %f\n",velocity);
    SD.Printf("Error: %f\n",error);

    errorSum[i] += error;
    //Prop term
    PTerm = error * Pconst;
    //Integral term
    ITerm = errorSum[i] * Iconst;
    //Derivative term *don't use until other two are working
    DTerm = (error - oldError[i]) * Dconst;
    //Replace variables

    if (i == 0){
    oldCounts[i] = right_encoder.Counts();
    }else{
    oldCounts[i] = left_encoder.Counts();
    }

    // ***** oldTime[i] = currentTime;
    oldError[i] = error;

    //return adjusted power, I and D terms commented out

    oldMotorPower[i] += PTerm; //+ DTerm; // + ITerm;

    SD.Printf("Set Motor Percent To: %f\n",oldMotorPower[i]);
    if (i == 0){
        SD.Printf("||||| RIGHT WHEEL  |||||\n");
    }else{
        SD.Printf("||||| LEFT WHEEL   |||||\n\n\\nn");
    }

    return(oldMotorPower[i]);//+ITerm+DTerm)
    // for competitions, the I and D terms were not used
}
```