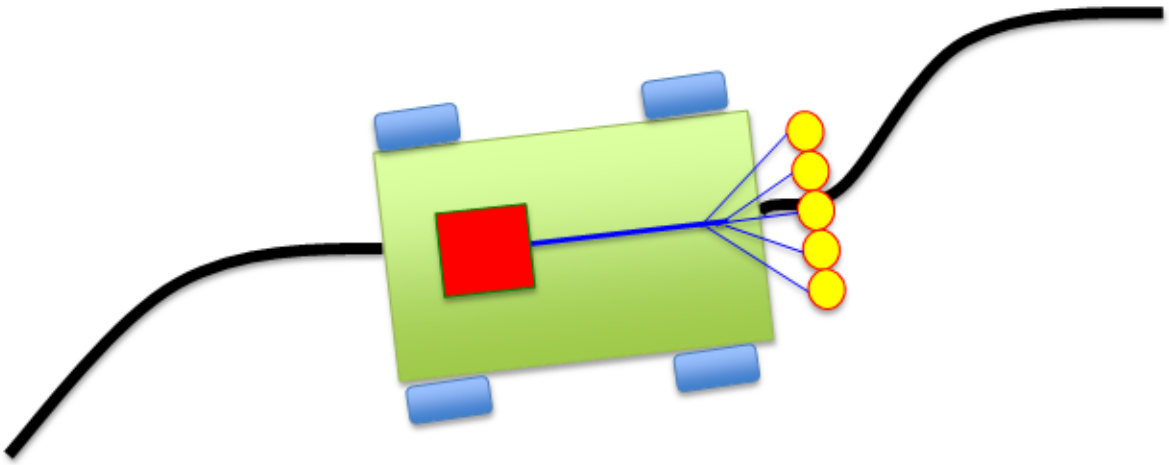
Robotics



Electrical and Computer Engineering 5

Embedded Systems and Control:
Line Following Robot

Developed by Karcher Morris & Dr. John Eldon

Professor John Eldon | Professor Vikash Gilja | Professor Drew Hall | Professor Truong Nguyen

Objective

The objective of this final project is to create a line following robot using much of the knowledge gained throughout the quarter and adding to it, a better understanding of sensors, actuators, programming, systems, and controls. Upon completing this objective, you will have the opportunity to compete against your classmates and push your creativity further in controller and system design.

Outline

What You Will Need

Materials:

- 1 Arduino Mega
- 1 USB Cable A-B
- 1 Cart Chassis – 3D Printed
- 1 Caster
- 1 Adafruit Motor Shield
- 3 Resistor (330 Ohms)
- 3 LEDs
- 2 DC Motors
- 2 Wheels
- 2 Breadboards
- 2 Battery Packs
- 4 Potentiometers
- 7 Resistors (10k Ohms)
- 7 Photoresistor
- 6 AA Batteries
- Colored Wire
- Solder
- Electrical tape

Machinery:

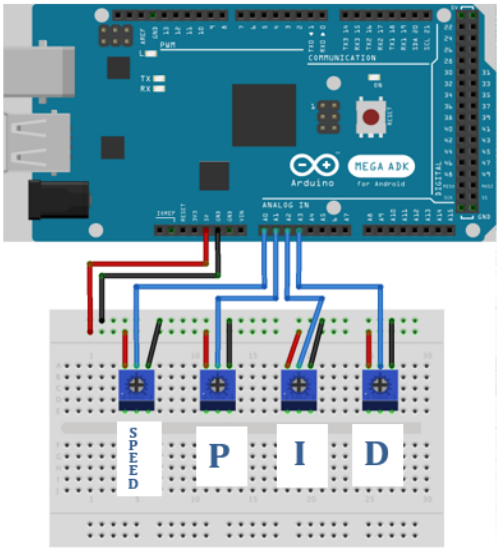
- Computer / Laptop
- Solder Station
- Screw Driver

Software:

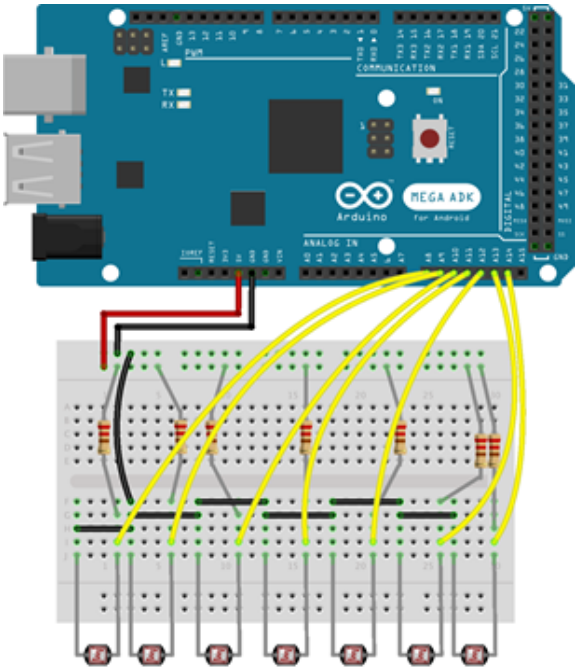
- Arduino Software (IDE)

Challenge #1: Potentiometers

For the first challenge, set up potentiometers on a bread board and follow the wiring diagram shown on the right. This challenge is not setting up your PID controller, it is only setting up a way to interface/change your Speed, P, I, and D coefficient values (variables) without the need to re-upload new code to your Arduino each time. Use the code in Appendix 10.1. Test this to ensure it prints the potentiometer output to the serial monitor. Remember that a potentiometer acts as a variable resistor. Do counter clockwise or clockwise twists of the potentiometers increase the value printed to the serial monitor? What are different ways to switch which direction, CCW or CW, increases the value?



Challenge #2: Photoresistors



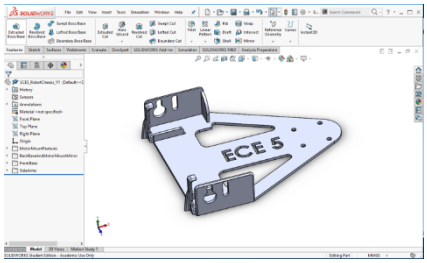
Develop multiple voltage dividers with, and instead of variable resistors being made of knobs, now use sensors that change resistance dependent on light. The voltage dividers should use 10kΩ resistors.

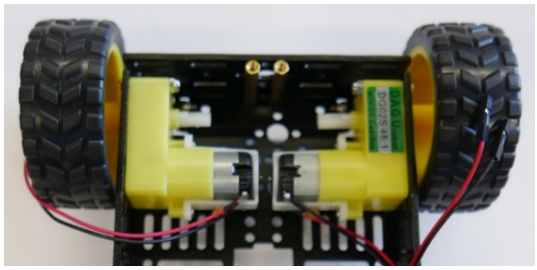
Once you have the board set up, use the code in Appendix 10.2 where you can read the values from each of your photo resistors and print them to the serial monitor. Test this to be sure each photo resistor is working properly.

Try placing it close to different colored surfaces and then try it out on a line. What is the best distance for your sensors to be away from the surface to properly differentiate when the sensors are hovering over black or white (this is very important!)? How does surrounding light or shadows affect the readings (this is also very important!)?

Challenge #3: Motors, Caster & Chassis Design

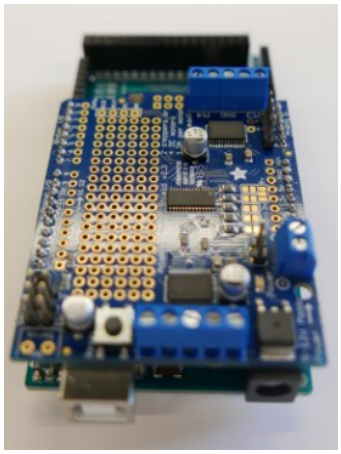
Now we want to start working on our chassis and motors. Design a chassis in SolidWorks. It may be based off of the SolidWorks part given to you in class but some modifications must be made. Improve the design or start from scratch. For this design we will only need 2 motors and 2 wheels. Instead of 2 more wheels on the front, we will use a caster. Remember that measurements should be made for your design and remember you will be 3D printing!



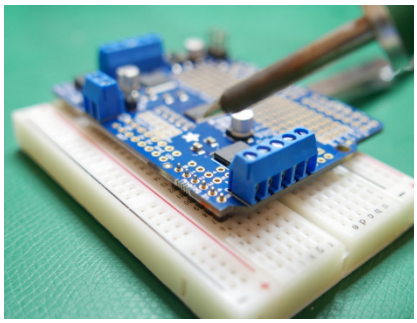


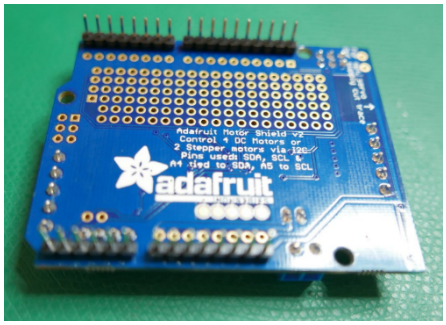
Challenge #4: Motor Driver and Arduino Mega

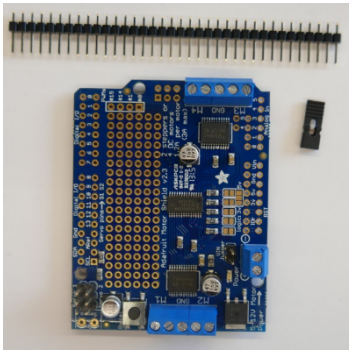
For the fourth challenge, first solder the header pins onto the motor shield. It helps to solder your first one or two pins of each strip while it is attached to an Arduino or a breadboard for alignment purposes. Check out the motor shield manual for some helpful hints. Note that the header pins on the outer edge of motor shield seen in the figures should have the longer ends on the bottom side capable of connecting to the Arduino. The header pins just on the inside of those should have the longer ends on the other (top) side for easy connectivity to other devices using female jumper wires.



Use the motor shield manual to download and install the Adafruit motor shield library for Arduino. Without the motors connected, but with the motor shield attached to the Arduino, ensure the code from Appendix 10.3 compiles and uploads without error.



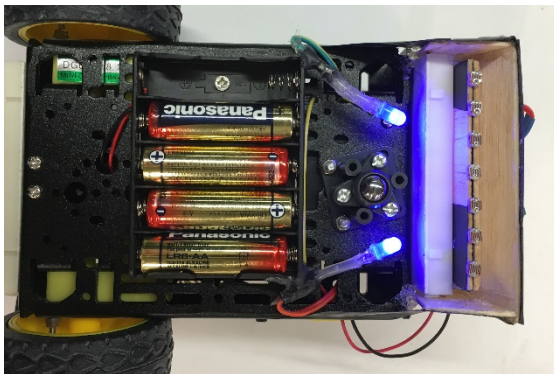




Challenge #5: Hardware Additions

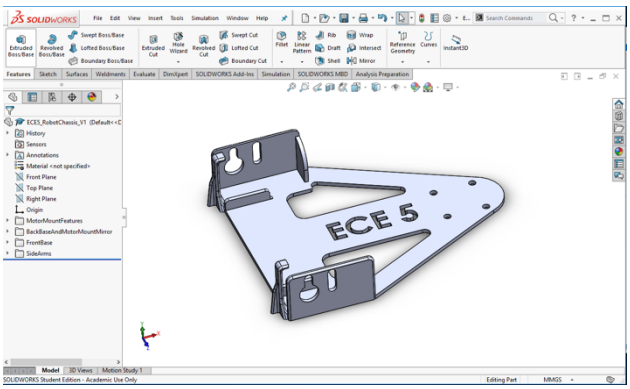
Light Shield and Light Source Additions:

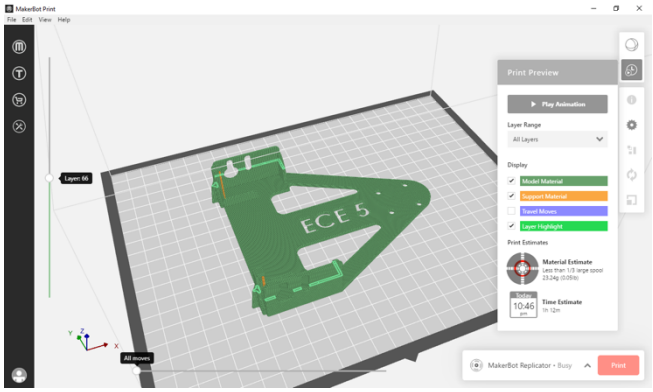
One problem that arises is the robot's sensitivity to ambient light changes and shadows. Add a light shield and light source on your robot to help mitigate this problem. You may do something similar to the robots shown below. You can be creative with the 3D printer follow the 3D printing tutorial, or simply cover with paper or balsa wood. Add LEDs that are always on by taking advantage of the 5V rails on the breadboard of your photoresistors and don't forget to add a resistor in series with each of your LEDs.

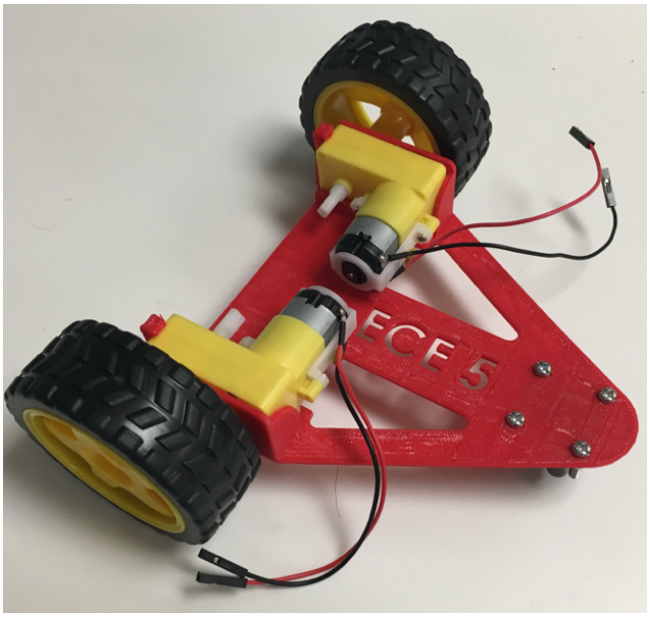


3D Printed Chassis or Chassis Modification:

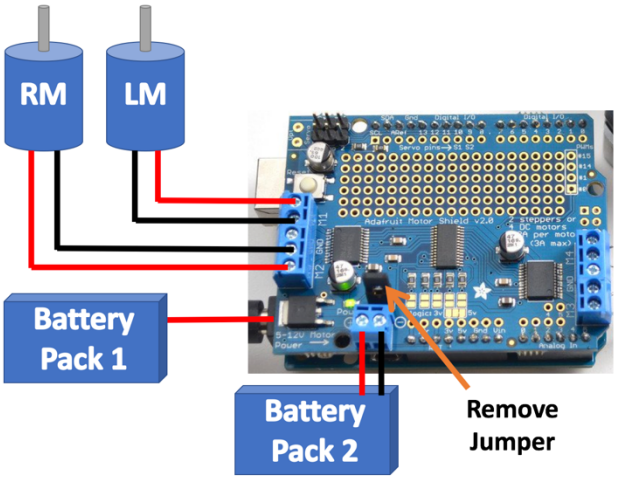
Ensure you make modifications to the design or start from scratch. Then take it to the 3D printer or laser cutter so you can put the whole vehicle together!







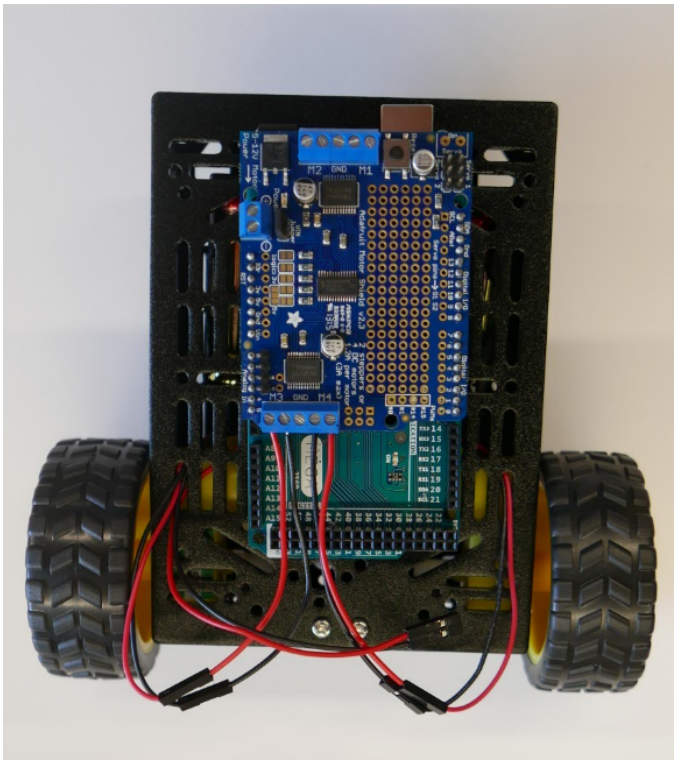
Challenge #6: System Integration Part 1



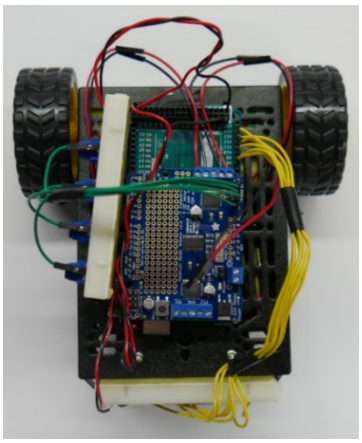
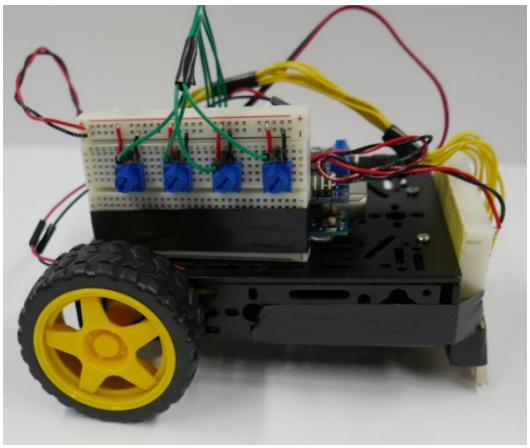
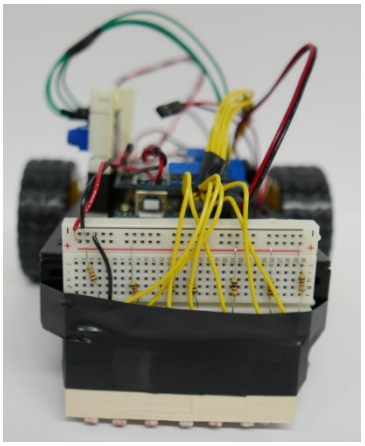
Looking at the diagram to the left, connect your motors to M1 and M2 on your motor shield. Ensure both power sources are available to connect, one for the Arduino Mega (BP 1), and one for the motor shield (BP 2). Do not connect the power sources and have your computer plugged into the Arduino's USB at the same time. You may need to switch the arrangement of your red and black wires connected to your motors depending on the set up. Switching black with red wires on one of your motors would switch the direction it is rotating.

Code it! You can use the code in Appendix 10.3 or build your own. You can use the Appendix 10.3 test code to move forward for 3 seconds and backward for 3 seconds. After reading through the code, can you try another maneuver like going in a circle or making a figure 8. Be careful/maybe do not try on top of a table or near coffee, because it can get away from you.

Note: With code 10.3, if you notice that one wheel moves forward and one wheel moves backward, or both wheels move backwards when it should be moving forward, you could simply change the polarity of wires coming from your motorshield from M1 and M2 by inserting the black wire where the red wire is and vice versa. You can also use M3 and M4 slots instead of M1 and M2 with a small code alteration declaring M3 and M4 as the motors to "attach" instead of M1 and M2.



Challenge #6: System Integration Part 2

First, there are many ways to put your robot together from here.

Remember to consider the height of those photoresistors from the ground for good sensitivity measuring white vs. black surfaces. This is very important. It may also be good to use a thin piece of balsa wood or thick paper to keep photoresistors aligned and kept from being easily bent.

Tape helps with nearly everything including putting your bread boards in place holding the Arduino Mega to a specific spot on your cart, keeping the wires harnessed/organized together, etc. Tape does not help so much with coding.

Run each of the programs from Challenges #1-#4 (10.1-10.3) individually. Does each potentiometer work? Does each photoresistor work and sense the difference between a black and white surface? Does the cart still drive forward and backward?

Challenge #7: Calibration

Calibrating your sensors is very important since sensor readings can easily alter due to changes in sensor position/alignment, changes in surrounding lighting, or other various reasons. This challenge will walk you through the process of calibrating your device using code found in Appendix 10.4. The main goal with your calibration is to take the possible readings from each sensor shown at the top of the two figures below, and then map those readings to a more uniform measure making it more clear where a black line may be located.

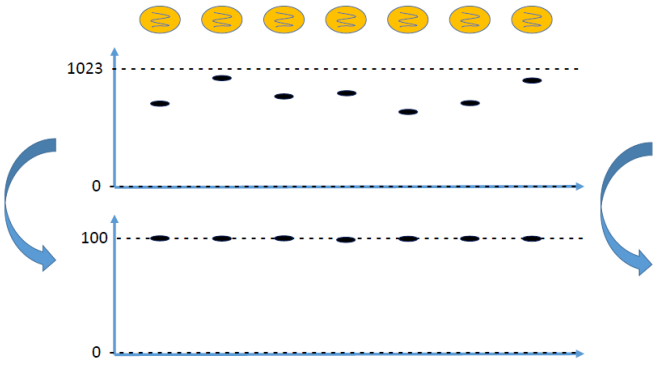


Figure 1: Mapped reading for a black line

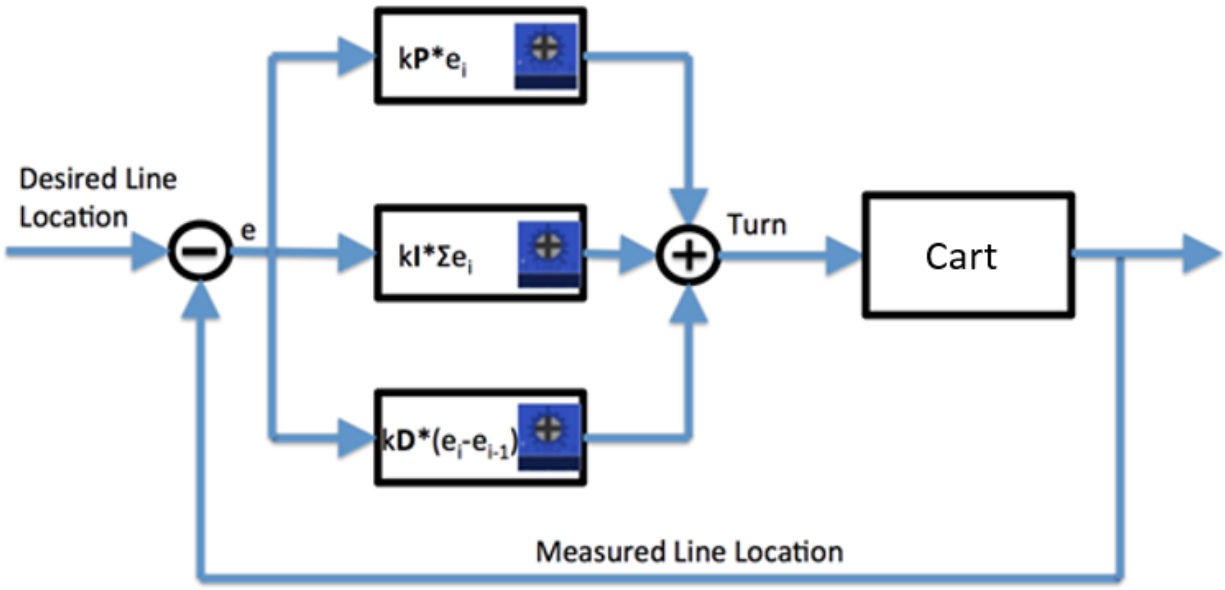
Figure 2: Mapped reading for a white line

Before implementing the new code, you may want to add an LED to make the high/low output on pin 13 more visible or choose another more accessible digital pin to control the LED, like digital pin 31 since pin 13 is hiding below the Adafruit Motor Shield. This LED is used to help you know what step of the calibration you should be performing.

After that, read through the code and comments found in Appendix 10.4. Test this code by uploading to the Arduino and following these steps:

1. While the connected LED is blinking slowly, place the photoresistors attached to the breadboard and cart over a white surface.
2. After a few seconds the LED will blink quickly and at this point it is important not to touch the cart or cast any shadows over it because it is finding the nominal photoresistor reading of each pin for what correlates with pure white.
3. The LED will then blink slowly again giving you time to move the photoresistors over a pure black surface or line. Ensure that each photoresistor is fully over the black surface.
4. Again, after a few seconds the LED will blink quickly and again it is important not to touch the cart or cast any shadows over it because it is finding the nominal photoresistor reading of each pin for what correlates with pure black.
5. Now the sensors should be calibrated and the photoresistors/Arduino should be able to distinguish clearly between what is black and white and the readings should be more uniform moving between 0 and 100. You should see these readings through the serial monitor.
6. Error, or distance from the centerline of the cart, is then calculated through a weighted average.

Challenge #8: PID Control

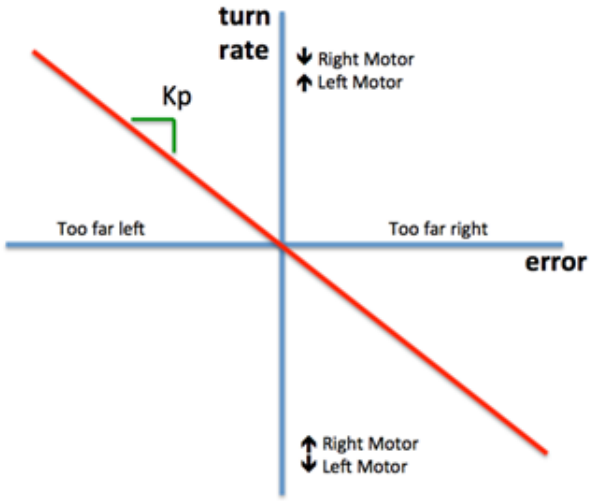


Now you will be able to apply what you learned in this week's lecture on PID control. In the past four sections you have been putting together each of the main components seen in the block diagram above, and now you will need to organize your code in a manner to connect everything together. The first step is to realize what is your error, e, and how to quantify it. This is what your sensors are for and your error may be found from comparing your measured value to your desired value.

error = (measured value - desired value)

You now need to operate on this error and you may break this portion down into 3 main operations: proportional, integral, and derivative control.

Proportional control, as explained in class, takes your current error value and scales it by your proportional gain, kP. This scaling should transform your error into a desired turn rate to correct for that error. The tank's turn rate is controlled through your left motor and right motor. If your error indicates that you are slightly to the right of the line you will need to increase the speed of your right motor and/or decrease speed to your left motor. You can imagine that as you are further and further to the right your turn rate will increase in magnitude to respond to this error more intensely. The next figure explains this proportional control graphically.



Integral control takes into account a sum of your past errors. This can be implemented into your code with:

sumerror = (sumerror + error)

This can be considered as memory of your error because as your error remains on one side of center, it will grow into a more and more impactful role affecting your turn rate. This can eliminate small errors and steady state error since as your error is small, maybe your proportional error won't affect the system enough to push it into your desired state, but as the error persists the gain, kI will be multiplied by a greater and greater "sumerror" to counter that small yet persistent inaccuracy.

Derivative control can predict an increase or decrease in your error and prepare your turn rate in advance. You can imagine this being very helpful on a sudden and sharp turn. By taking your current error and subtracting your previous error, you will be able to see how much your error has changed during this one time step.

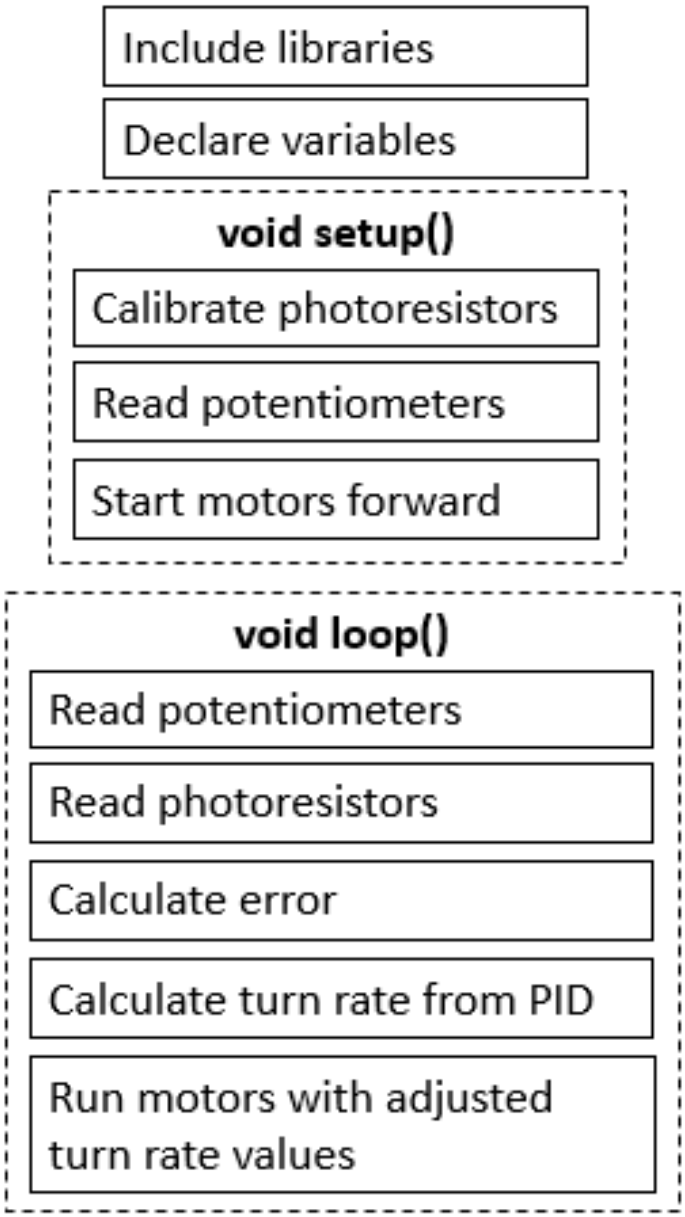
deriverror = (error – previous error)

Odds are high that, unless corrected for, the rate at which your error changes will remain the same from time step to time step. By recognizing this worsening (or improvement) of your error, you will be able to increase (or decrease) the amount of additional right or left motor speed necessary to counter such a change.

Now we can add all of these terms together to calculate our turn rate to control our tank.

*Turnrate = (kP*error + kI*sumerrror + kD*deriverror)*

This turn rate should be transferred to an increase or decrease in motor speed for your left and/or right motor.
Code Flow:


Include libraries
Declare variables
void setup()
Calibrate photoresistors
Read potentiometers
Start motors forward
void loop()
Read potentiometers
Read photoresistors
Calculate error
Calculate turn rate from PID
Run motors with adjusted turn rate values

Challenge #9: Competition

1) A series of line courses

Each course will have different criteria for score including (1) distance of line followed, (2) time to complete certain portions of the course, (3) error off center at certain points, etc. With the ability to control your speed and PID knobs, your robot will compete against your classmate's robots.

2) Innovation/Creativity challenge

Using any of the sensors or actuators you have learned about throughout the quarter or any others that are mentioned above, you will have the opportunity to piece together additional components to improve your robot's control, add to usefulness, or increase the level of interactivity/fun.

Challenge #10: Appendix

Note: Arduino (.ino) files are also provided for 10.1-10.5 so it is not necessary to copy and paste.

10.1 - Potentiometer Code Outline

```
/* Potentiometer Code UCSD ECE 5 Lab 4*/
/* Declare Variables for Potentiometer */
const int S_pin = A0; // proportional - analog pin 0
const int P_pin = A1; // proportional - analog pin 1
const int I_pin = A2; // integral - analog pin 2
const int D_pin = A3; // derivative - analog pin 3
int Sp = 0; // speed gain coeficient
int kP = 0; // proportional gain coeficient
int kI = 0; // integral gain coeficient
int kD = 0; // derivative gain coeficient

void setup() {                  /* Setup - runs once (when power is supplied or after reset) */

  Serial.begin(9600);          // For serial communication set up
}

void loop() {                   /* Loop - loops forever (until unpowered or reset) */

  ReadPotentiometers();       // Call on user-defined function to read Potentiometer values

  Print();                     // Call on user-defined function to print values from potentiometers

}

// ********************************************************************************************** //
// function to read and map values from potentiometers and map into 0-100
void ReadPotentiometers()
{
  Sp = map(analogRead(S_pin), 0, 1023, 0, 100);
  kP = map(analogRead(P_pin), 0, 1023, 0, 100);
  kI = map(analogRead(I_pin), 0, 1023, 0, 100);
  kD = map(analogRead(D_pin), 0, 1023, 0, 100);
}

// ********************************************************************************************** //
// function to print values of interest
void Print()
{


  Serial.print(Sp); Serial.print(" ");
  Serial.print(kP); Serial.print(" ");
  Serial.print(kI); Serial.print(" ");
  Serial.println(kD);

  delay(200); //just here to slow down the output for easier reading if desired

}
```

10.2 Photoresistor Code Outline

```
/* Photoresistor Code UCSD ECE 5 Lab 4 */
/* Connect the seven photoresistor V_out, left to right, to analog pin 8-14 */
/* Variables for Light Sensors*/
int LDR_Pin0 = A8 ; // analog pin  8
int LDR_Pin1 = A9 ; // analog pin  9
int LDR_Pin2 = A10; // analog pin 10
int LDR_Pin3 = A11; // analog pin 11
int LDR_Pin4 = A12; // analog pin 12
int LDR_Pin5 = A13; // analog pin 13
int LDR_Pin6 = A14; // analog pin 14

// Initialize Photo Resistor Variables
int LDR0 = 0, LDR1 = 0, LDR2 = 0, LDR3 = 0, LDR4 = 0, LDR5 = 0, LDR6 = 0;

void setup() {

  Serial.begin(9600);         // For serial communication set up

}

void loop() {

  ReadPhotoResistors(); // Read photoresistors and map to 0-100 based on calibration

  Print();              // Print values to serial monitor
}

// ********************************************************************************************** //
// function to read photo resistors
void ReadPhotoResistors()
{

  LDR0 = analogRead(LDR_Pin0);
  delay(2);
  LDR1 = analogRead(LDR_Pin1);
  delay(2);
  LDR2 = analogRead(LDR_Pin2);
  delay(2);
  LDR3 = analogRead(LDR_Pin3);
  delay(2);
  LDR4 = analogRead(LDR_Pin4);
  delay(2);
  LDR5 = analogRead(LDR_Pin5);
  delay(2);
  LDR6 = analogRead(LDR_Pin6);
  delay(2);
}

// ********************************************************************************************** //
// function to print values of interest
void Print()
{

  Serial.print(LDR0); Serial.print(" ");
  Serial.print(LDR1); Serial.print(" ");
  Serial.print(LDR2); Serial.print(" ");
  Serial.print(LDR3); Serial.print(" ");
```

```
  Serial.print(LDR4); Serial.print(" ");
  Serial.print(LDR5); Serial.print(" ");
  Serial.println(LDR6);

  delay(200); //just here to slow down the output for easier reading if desired

}
```

10.3 Motor Driver Code

```
/* Motor Driver Testing Code UCSD ECE 5 Lab 4*/
// Libraries for Motor
#include <Wire.h>
#include <Adafruit_MotorShield.h>  // Must add libary - see MotorShield Manual
//https://cdn-learn.adafruit.com/downloads/pdf/adafruit-motor-shield-v2-for-arduino.pdf

// Initialize Motors
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *Motor1 = AFMS.getMotor(1);  // ******Adjust if needed******
Adafruit_DCMotor *Motor2 = AFMS.getMotor(2);  // Motors can be switched here (1) <--> (2)

//Set Initial Speed of Motors
int M1Sp = 60; // initial speed may vary and later can be increased with Sp potentiometer
int M2Sp = 60;

//Set LED Pin
int led_Pin = 13;  // can change to another digital pin and connect extra LED to me more easily seen

// setup - runs once
void setup() {
  Serial.begin(9600);
  AFMS.begin(); //for Motor

  pinMode(led_Pin, OUTPUT); // set pin mode to output voltage
  // Gives you a moment before tank actually moves
  for (int waitii = 0; waitii < 20; waitii++) {
    digitalWrite(led_Pin, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(100);                    // wait for 100 milliseconds
    digitalWrite(led_Pin, LOW);  // turn the LED off by making the voltage LOW
    delay(100);                    // wait for 100 milliseconds
  }
}
// loop - loops forever
void loop() {
  // Start Motors in forward direction
  Motor1->setSpeed(M1Sp);
  Motor1->run(FORWARD);
  Motor2->setSpeed(M2Sp);
  Motor2->run(FORWARD);
  delay(3000);                   // let run forward for 3 seconds

  // Start Motors in backward direction
  Motor1->setSpeed(M1Sp);
  Motor1->run(BACKWARD);
  Motor2->setSpeed(M2Sp);
  Motor2->run(BACKWARD);
  delay(3000);                   // let run backward for 3 seconds

  // Stop Motors
  Motor1->setSpeed(M1Sp);
  Motor1->run(RELEASE);
  Motor2->setSpeed(M2Sp);
  Motor2->run(RELEASE);
  delay(3000);                // stop for 3 seconds
}
```

10.4 Calibration Code

```
/* Calibration Test Code UCSD ECE 5 Lab 4 */
/* After calibration, you should see on Serial all photoresistors read ~100
on black, ~0 on white */
// Variables for Light Sensors (read void ReadPhotoResistors())
int LDR_Pin[7] = {A8, A9, A10, A11, A12, A13, A14}; // store photoresistor pins
int LDR[7]; // store photoresistor readings

// Calibration Variables (read void Calibrate())
int led_Pin = 13;      // This pin is for a led built into the Arduino that
                       //indicates what part of the calibration you are on
// You can use any digital pin like digital pin 31 with an LED connected for better visibility
float Mn[7];
float Mx[7];
float LDRf[7] = {0., 0., 0., 0., 0., 0., 0.};

// Error Calculation Variables (read void CalcError())
int   MxRead;
int   MxIndex;
float AveRead;
int   CriteriaForMax;
float WeightedAve;
int   ii;
int   im0, im1, im2;
float error;

// ********************************************************************************************* //
// setup - runs once
void setup()
{

  Serial.begin(9600);         // For serial communication set up
  pinMode(led_Pin, OUTPUT);  // Note that all analog pins used are INPUTs by
default so don't need pinMode

  Calibrate();               // Calibrate black and white sensing

}  // end setup()

// ********************************************************************************************* //
// loop - runs/loops forever
void loop()
{

  ReadPhotoResistors(); // Read photoresistors and map to 0-100 based on calibration

  CalcError();

  Print();          // Print values to serial monitor
//currently commented out but could be good for debugging =)

}  // end loop()

// ********************************************************************************************* //
// function to calibrate
void Calibrate()
{
  // wait to make sure in position
  for (int calii = 0; calii < 4; calii++)
  {
```

```
    digitalWrite(led_Pin, HIGH);   // turn the LED on
    delay(100);                    // wait for 0.1 seconds
    digitalWrite(led_Pin, LOW);    // turn the LED off
    delay(900);                    // wait for 0.9 seconds
  }

  // Calibration
  // White Calibration
  int numMeas = 10;  // number of samples to read for calibration
  for (int calii = 0; calii < numMeas; calii++)
  {
    digitalWrite(led_Pin, HIGH);   // turn the LED on
    delay(100);                    // wait for 0.1 seconds
    digitalWrite(led_Pin, LOW);    // turn the LED off
    delay(100);                    // wait for 0.1 seconds

    for ( int ci = 0; ci < 7; ci++ )
    {
      LDRf[ci] = LDRf[ci] + (float) analogRead(LDR_Pin[ci]);
      delay(2);
    }
  }

  for ( int cm = 0; cm < 7; cm++ )
  {
    Mn[cm] = round(LDRf[cm] / (float)numMeas); // take average
    LDRf[cm] = 0.;
  }

  // Time to move from White to Black Surface
  for (int calii = 0; calii < 6; calii++)
  {
    digitalWrite(led_Pin, HIGH);
    delay(100);
    digitalWrite(led_Pin, LOW);
    delay(900);
  }

  // Black Calibration
  for (int calii = 0; calii < numMeas; calii++)
  {
    digitalWrite(led_Pin, HIGH);
    delay(100);
    digitalWrite(led_Pin, LOW);
    delay(100);

    for ( int ci = 0; ci < 7; ci++ )
    {
      LDRf[ci] = LDRf[ci] + (float) analogRead(LDR_Pin[ci]);
      delay(2);
    }
  }
  for ( int cm = 0; cm < 7; cm++ )
  {
    Mx[cm] = round(LDRf[cm] / (float)numMeas); // take average
    LDRf[cm] = 0.;
  }

}  // end Calibrate()
```

```
// ********************************************************************************************** //
// function to read photo resistors, map from 0 to 100
void ReadPhotoResistors()
{
  for (int Li = 0; Li < 7; Li++)
  {
    LDR[Li] = map(analogRead(LDR_Pin[Li]), Mn[Li], Mx[Li], 0, 100);
    delay(2);
  }
}   // end ReadPhotoResistors()

// ********************************************************************************************** //
// Calculate error from photoresistor readings
void CalcError()
{
  MxRead = -99;
  AveRead = 0.0;
  // Step 1)  Iterate through photoresistor mapped values, find darkest/max (MxRead),
  // it's index (im1) and weighted index (MxIndex)
  // Weighted Index: from left to right: 3 - 2 - 1 - 0 (center) - 1 - 2 - 3
  for (int ii = 0; ii < 7; ii++)
  {
    if (MxRead < LDR[ii])
    {
      MxRead = LDR[ii];
      MxIndex = -1 * (ii - 3);
      im1 = (float)ii;
    }
    AveRead = AveRead + (float)LDR[ii] / 7.;
  }

  // Step 2)  Calculate error from weighted average, based off the readings around the maximum value
  CriteriaForMax = 2;  // max should be at least twice as big as the other values
  if (MxRead > CriteriaForMax * AveRead)  // only when the max is big enough
  {
    if (im1 != 0 && im1 != 6)  // max not on either ends
    {
      im0 = im1 - 1;  // index for left
      im2 = im1 + 1;  // index for right
      // if the denominator calculates to 0, jump out and do not update error
      if (LDR[im0] + LDR[im1] + LDR[im2] == 0)
        return;
      WeightedAve = ((float)(LDR[im0] * im0 + LDR[im1] * im1 + LDR[im2] *
im2)) / ((float)(LDR[im0] + LDR[im1] + LDR[im2]));
      error = -1 * (WeightedAve - 3);
    }
    else if (im1 == 0)  // max on left end
    {
      im2 = im1 + 1;
      // if the denominator calculates to 0, jump out and do not update error
      if (LDR[im1] + LDR[im2] == 0)
        return;
      WeightedAve = ((float)(LDR[im1] * im1 + LDR[im2] * im2)) /
((float)(LDR[im1] + LDR[im2]));
      error = -1 * (WeightedAve - 3);
    }
    else if (im1 == 6)  // max on right end
    {
```

```
      im0 = im1 - 1;
      // if the denominator calculates to 0, jump out and do not update error
      if (LDR[im0] + LDR[im1] == 0)
        return;
      WeightedAve = ((float)(LDR[im0] * im0 + LDR[im1] * im1)) /
((float)(LDR[im0] + LDR[im1]));
      error = -1 * (WeightedAve - 3);
    }
  }
}  // end CalcError()

// ********************************************************************************************** //
// function to print values of interest
void Print()
{

  Serial.print(LDR[0]); Serial.print(" ");  // Each photo resistor value is shown
  Serial.print(LDR[1]); Serial.print(" ");
  Serial.print(LDR[2]); Serial.print(" ");
  Serial.print(LDR[3]); Serial.print(" ");
  Serial.print(LDR[4]); Serial.print(" ");
  Serial.print(LDR[5]); Serial.print(" ");
  Serial.print(LDR[6]); Serial.print("    ");

  // the maximum value from the photo resistors is shown again
  Serial.print(MxRead); Serial.print(" ");
  // this is the index of that maximum (0 through 6) (aka which element in LDR)
  Serial.print(MxIndex); Serial.print("    ");
  // this will show the calculated error (-3 through 3)
  Serial.print(error); Serial.println("    ");

  delay(100); //just here to slow down the output for easier reading if wanted

}  // end Print()
```

10.5 Line Follower Code

```
/* ********************************************************************************************** */
// * UCSD ECE 5 Lab 4 Code: Line Following Robot with PID * //
/* ********************************************************************************************** */

//   This is code for your PID controlled line following robot.
//
//
//
//
// Code Table of Contents
//  1)  Declare Variables     - declares many variables as global variables so each variable can be accessed from every function
//  2)  Setup (Main)          - runs once at beginning when you press button on arduino or motor drive or when you open serial monitor
//  3)  Loop  (Main)          - loops forever calling on a series of functions
//  4)  Calibration           - makes white = 0 and black = 100 (a few seconds to prep, a few seconds on white, a few seconds to move to black, a few seconds of black)
//  5)  Read Potentiometers   - reads each potentiometer
//  6)  Run Motors            - runs motors
//  7)  Read Photoresistors   - reads each photoresistor
//  8)  Calculate Error       - calculate error from photoresistor readings
//  9)  PID Turn              - takes the error and implements PID control
//  10) Print                 - used for debugging but should comment out when not debugging because it slows down program

// ********************************************************************************************** //
// Declare Variables


// Variables and Libaries for Motor (This part is from Motor_Driver_Code 10_3)
#include <Wire.h>
#include <Adafruit_MotorShield.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *Motor1 = AFMS.getMotor(1);  //  Motors can be switched here (1) <--> (2) (if you see motors responding to error in the opposite way they should be)
Adafruit_DCMotor *Motor2 = AFMS.getMotor(2);  //  ******Adjust the same way if you did in 10_3******

int M1Sp = 20, M2Sp = 20;   // this is the nominal speed for the motors when not using potentiometer
int M1SpeedtoMotor, M2SpeedtoMotor;

// Variables for Potentiometer (This part is from Potentiometer_Code 10_1)
const int S_pin = A0; //proportional control
const int P_pin = A1; //proportional control
const int I_pin = A2; //integral control
const int D_pin = A3; //derivative control
int       SpRead = 0; int Sp; //Speed Increase
int       kPRead = 0; //proportional gain
int       kIRead = 0; //integral gain
int       kDRead = 0; //derivative gain

// Variables for Light Sensors (This part is from Calibration_Code 10_4)
int LDR_Pin[7] = {A8, A9, A10, A11, A12, A13, A14}; // store photoresistor pins
int LDR[7]; // store photoresistor readings

// Calibration Variables (This part is from Calibration_Code 10_4)
int   led_Pin = 13;      /// This is a led set up to indicate what part of the calibration you are on.
float Mn[7];             ///     You could also connect a larger/more visible LED to Pin 13 or other digital pin
float Mx[7];
float LDRf[7] = {0., 0., 0., 0., 0., 0., 0.};

// Error Calculation Variables (This part is from Calibration_Code 10_4)
int   MxRead;
int   MxIndex;
float AveRead;
int   CriteriaForMax;
float WeightedAve;
int   ii;
int   im0, im1, im2;

// For Motor/Control
int   Turn, M1P = 0, M2P = 0;
float error, lasterror = 0, sumerror = 0;
float kP, kI, kD;



// ********************************************************************************************** //
// setup - runs once
void setup()
{

  Serial.begin(9600);        // For serial communication set up
  AFMS.begin();              // For motor setup
  pinMode(led_Pin, OUTPUT);  // Note that all analog pins used are INPUTs by default so don't need pinMode

  Calibrate();               // Calibrate black and white sensing

  ReadPotentiometers();      // Read potentiometer values (Sp, P, I, & D)

  delay(2000);

  RunMotors();               // Starts motors forward and strait depending on Sp (Speed from potentiometer) and M1Sp/M2Sp (Nominal values)

}  // end setup()

// ********************************************************************************************** //
// loop - runs/loops forever
void loop()
{

  ReadPotentiometers(); // Only if you want to see Potentiometers working in set up as you run the line following

  ReadPhotoResistors(); // Read photoresistors and map to 0-100 based on calibration

  CalcError();

  PID_Turn();         // PID Control and Output to motors to turn

  RunMotors();        // Uses info from

  //   Print();          // Print values to serial monitor  //currently commented out but could be good for debugging =)

}  // end loop()

// ********************************************************************************************** //
// function to calibrate
void Calibrate()
{
  // wait to make sure in position
  for (int calii = 0; calii < 4; calii++)
  {
    digitalWrite(led_Pin, HIGH);   // turn the LED on
    delay(100);                    // wait for 0.1 seconds
    digitalWrite(led_Pin, LOW);    // turn the LED off
    delay(900);                    // wait for 0.9 seconds
  }

  // Calibration
  // White Calibration
  int numMeas = 10;  // number of samples to read for calibration
  for (int calii = 0; calii < numMeas; calii++)
  {
    digitalWrite(led_Pin, HIGH);   // turn the LED on
    delay(100);                    // wait for 0.1 seconds
    digitalWrite(led_Pin, LOW);    // turn the LED off
    delay(100);                    // wait for 0.1 seconds

    for ( int ci = 0; ci < 7; ci++ )
    {
      LDRf[ci] = LDRf[ci] + (float) analogRead(LDR_Pin[ci]);
      delay(2);
    }
  }

  for ( int cm = 0; cm < 7; cm++ )
  {
    Mn[cm] = round(LDRf[cm] / (float)numMeas); // take average
    LDRf[cm] = 0.;
```

```
  }

  // Time to move from White to Black Surface
  for (int calii = 0; calii < 10; calii++)
  {
    digitalWrite(led_Pin, HIGH);
    delay(100);
    digitalWrite(led_Pin, LOW);
    delay(900);
  }

  // Black Calibration
  for (int calii = 0; calii < numMeas; calii++)
  {
    digitalWrite(led_Pin, HIGH);
    delay(100);
    digitalWrite(led_Pin, LOW);
    delay(100);

    for ( int ci = 0; ci < 7; ci++ )
    {
      LDRf[ci] = LDRf[ci] + (float) analogRead(LDR_Pin[ci]);
      delay(2);
    }
  }
  for ( int cm = 0; cm < 7; cm++ )
  {
    Mx[cm] = round(LDRf[cm] / (float)numMeas); // take average
    LDRf[cm] = 0.;
  }

}  // end Calibrate()

// ********************************************************************************************** //
// function to read and map values from potentiometers
void ReadPotentiometers()
{
  SpRead = map(analogRead(S_pin), 0, 1023, 0, 50); Sp = SpRead;
  kPRead = map(analogRead(P_pin), 0, 1023, 0, 10);
  kIRead = map(analogRead(I_pin), 0, 1023, 0, 5);
  kDRead = map(analogRead(D_pin), 0, 1023, 0, 10);
}    // end ReadPotentiometers()

// ********************************************************************************************** //
// function to start motors using nominal speed + speed addition from potentiometer
void RunMotors()
{
  // ******Remember you can insert your custom control in here******
  // ******To control motor behaviour from error, etc******
  M1SpeedtoMotor = min(M1Sp + Sp + M1P, 255); // limits speed to 255
  M2SpeedtoMotor = min(M2Sp + Sp + M2P, 255); // remember M1Sp & M2Sp is defined at beginning of code (default 60)
  Motor1->setSpeed(abs(M1SpeedtoMotor));
  Motor2->setSpeed(abs(M2SpeedtoMotor));

  if (M1SpeedtoMotor > 0)
  {
    Motor1->run(FORWARD);
  }
  else
  {
    Motor1->run(BACKWARD);
  }

  if (M2SpeedtoMotor > 0)
  {
    Motor2->run(FORWARD);
  }
  else
  {
    Motor2->run(BACKWARD);
  }

}  // end RunMotors()

// ********************************************************************************************** //
// function to read photo resistors, map from 0 to 100, and find darkest photo resitor (MxIndex)
void ReadPhotoResistors()
{
  for (int Li = 0; Li < 7; Li++)
  {
    LDR[Li] = map(analogRead(LDR_Pin[Li]), Mn[Li], Mx[Li], 0, 100);
    delay(2);
  }
}   // end ReadPhotoResistors()

// ********************************************************************************************** //
// Calculate error from photoresistor readings
void CalcError()
{
  MxRead = -99;
  AveRead = 0.0;
  // Step 1)  Iterate through photoresistor mapped values, find darkest/max (MxRead), it's index (im1) and weighted index (MxIndex)
  //          Weighted Index: from left to right: 3 - 2 - 1 - 0 (center) - 1 - 2 - 3
  for (int ii = 0; ii < 7; ii++)
  {
    if (MxRead < LDR[ii])
    {
      MxRead = LDR[ii];
      MxIndex = -1 * (ii - 3);
      im1 = (float)ii;
    }
    AveRead = AveRead + (float)LDR[ii] / 7.;
  }

  // Step 2)  Calculate error from weighted average, based off the readings around the maximum value
  CriteriaForMax = 2;  // max should be at least twice as big as the other values
  if (MxRead > CriteriaForMax * AveRead)  // only when the max is big enough
  {
    if (im1 != 0 && im1 != 6)  // max not on either ends
    {
      im0 = im1 - 1;  // index for left
      im2 = im1 + 1;  // index for right
      if (LDR[im0] + LDR[im1] + LDR[im2] == 0)  // if the denominator calculates to 0, jump out and do not update error
        return;
      WeightedAve = ((float)(LDR[im0] * im0 + LDR[im1] * im1 + LDR[im2] * im2)) / ((float)(LDR[im0] + LDR[im1] + LDR[im2]));
      error = -1 * (WeightedAve - 3);
    }
    else if (im1 == 0)  // max on left end
    {
      im2 = im1 + 1;
      if (LDR[im1] + LDR[im2] == 0)  // if the denominator calculates to 0, jump out and do not update error
        return;
      WeightedAve = ((float)(LDR[im1] * im1 + LDR[im2] * im2)) / ((float)(LDR[im1] + LDR[im2]));
      error = -1 * (WeightedAve - 3);
    }
    else if (im1 == 6)  // max on right end
    {
      im0 = im1 - 1;
      if (LDR[im0] + LDR[im1] == 0)  // if the denominator calculates to 0, jump out and do not update error
        return;
      WeightedAve = ((float)(LDR[im0] * im0 + LDR[im1] * im1)) / ((float)(LDR[im0] + LDR[im1]));
      error = -1 * (WeightedAve - 3);
    }
  }
}  // end CalcError()


// ********************************************************************************************** //
// function to make a turn ( a basic P controller)
void PID_Turn()
{
  // *Read values are between 0 and 100, scale to become PID Constants
  kP = (float)kPRead / 1.;    // each of these scaling factors can change depending on how influential you want them to be
  kI = (float)kIRead / 1000.;  // the potentiometers will also scale them
  kD = (float)kDRead / 100.;
  // error holds values from -3 to 3

  Turn = error * kP + sumerror * kI + (error - lasterror) * kD; //PID!!!!!
```

```
  sumerror = sumerror + error;
  if (sumerror > 5) {
    sumerror = 5; // prevents integrator wind-up
  }
  else if (sumerror < -5) {
    sumerror = -5;
  }

  lasterror = error;

}  // end PID_Turn()


// ********************************************************************************************** //
// function to print values of interest
void Print()
{
  Serial.print(SpRead); Serial.print(" ");  // Initial Speed addition from potentiometer
  Serial.print(kP); Serial.print(" ");      // PID values from potentiometers after scaling
  Serial.print(kI); Serial.print(" ");
  Serial.print(kD); Serial.print("    ");

  Serial.print(LDR[0]); Serial.print(" ");  // Each photo resistor value is shown
  Serial.print(LDR[1]); Serial.print(" ");
  Serial.print(LDR[2]); Serial.print(" ");
  Serial.print(LDR[3]); Serial.print(" ");
  Serial.print(LDR[4]); Serial.print(" ");
  Serial.print(LDR[5]); Serial.print(" ");
  Serial.print(LDR[6]); Serial.print("    ");

  Serial.print(MxRead); Serial.print(" ");    // the maximum value from the photo resistors is shown again
  Serial.print(MxIndex); Serial.print("    "); // this is the index of that maximum (0 through 6) (aka which element in LDR)
  Serial.print(error); Serial.print("    ");  // this will show the calculated error (-3 through 3)

  Serial.print(M1SpeedtoMotor); Serial.print(" "); // This prints the arduino output to each motor so you can see what the values (0-255)
  Serial.println(M2SpeedtoMotor);                  //  that are sent to the motors would be without actually needing to power/run the motors

  // delay(100); //just here to slow down the output for easier reading if wanted
  // ensure delay is commented when actually running your robot or this will slow down sampling too much
}  // end Print()
```