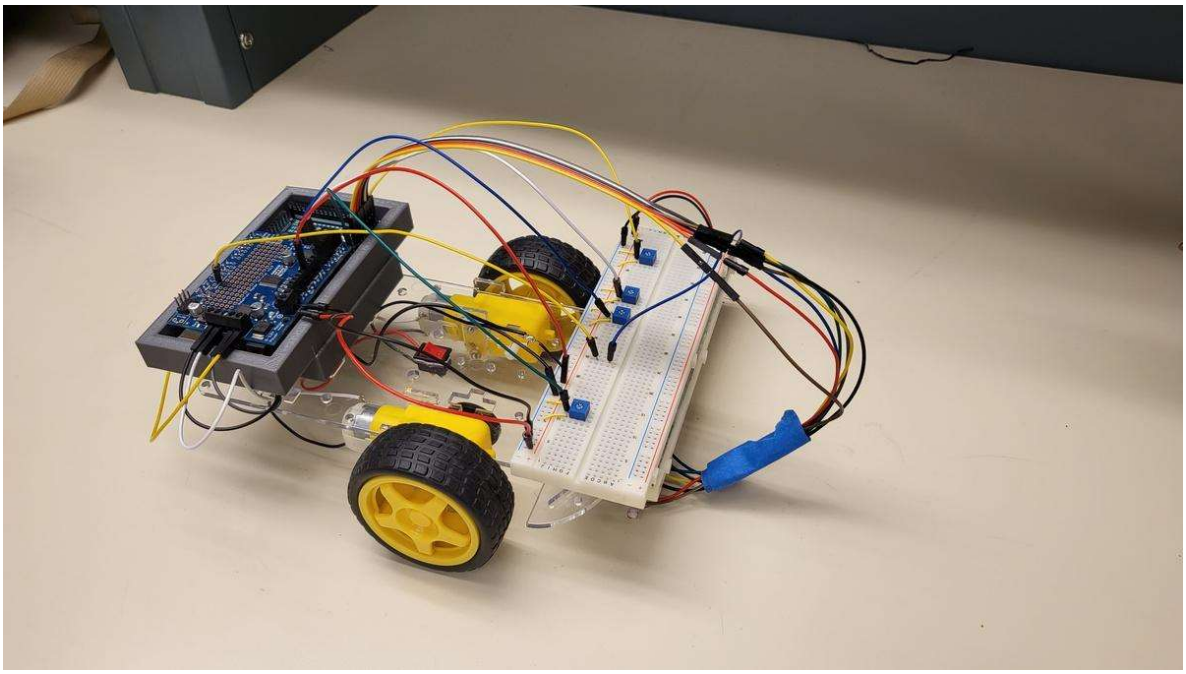
Final Project: Robotics



Electrical and Computer Engineering 201

Embedded Systems and Control:
Line Following Robot

Objective

The objective of this final project is to create a line-following robot. Upon completion, you will have a better understanding of sensors, actuators, programming, systems, and controls. You will have the opportunity to compete against your classmates and push your creativity in controller and system design.

What You Will Need

Materials:

- 1 Arduino Mega
- 1 USB Cable (A to B)
- 1 Cart Chassis Kit
- 2 DC Motors
- 2 Encoder Discs
- 2 Wheels
- 1 Caster
- 1 Switch
- 4 Fasteners
- 4-30mm Screws
- 8-6mm Screws
- 4 Spacer Bars
- 8 Hex Nuts
- 1 Adafruit Motor Shield
- Set of Female/Male headers
- 2 Resistors (330 Ohms)
- 7 LEDs
- 2 Mini Breadboards
- 4 10K Potentiometers
- 7 Resistors (10k Ohms)
- 7 Photoresistors
- 4 AA Batteries
- Jumper Wires
- Solid-core Colored Wire (for soldering)
- Solder
- Double-sided tape
- Cardboard or Cardstock

Equipment:

- Computer or laptop (USB A port or hub)
- Solder station **with proper ventilation**
- Philips head screwdriver
- Wire strippers
- Potentiometer trimming tool
- Flush wire cutters
- **One** final project kit
- **Two** partner lab kits

Software:

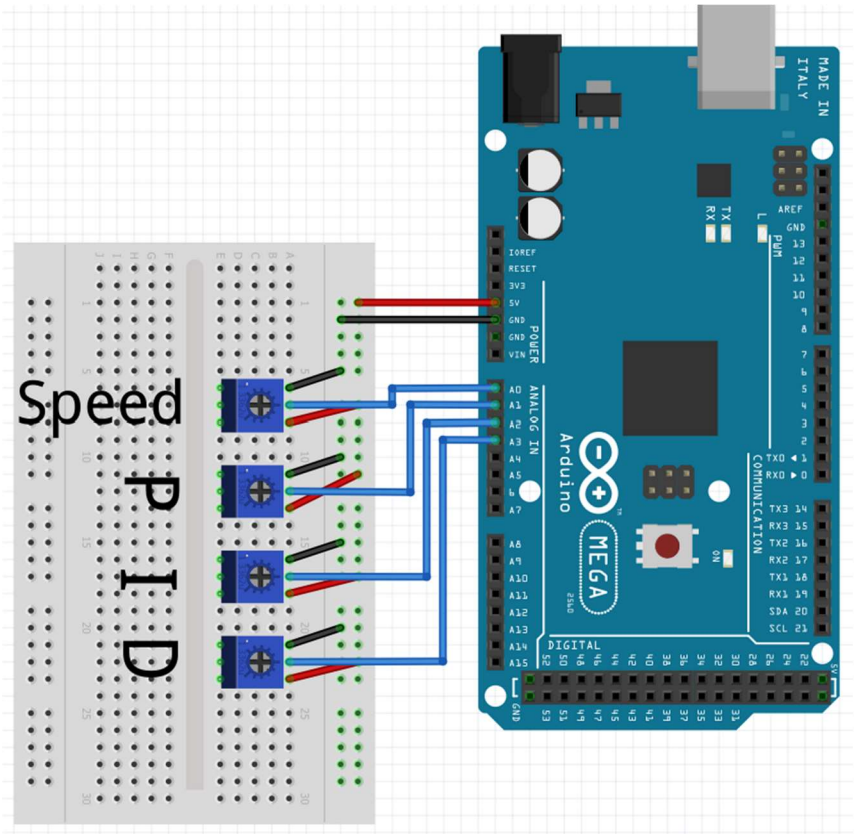
- Arduino Software (IDE)
- Fusion 360 (Computer Aided Design, CAD)
- Eagle or KiCad (Printed Circuit Board Design, PCB)

Challenge #1: Potentiometers

For the first challenge, set up potentiometers on a breadboard and follow the wiring diagram shown below.

This challenge is not setting up your PID controller, it is only setting up a way to interface or change your Speed (S), Proportional (P), Integral (I), and Derivative (D) coefficient values (variables) without the need to re-upload new code to your Arduino each time. See Challenge #8 for details on how these will be used later.

Once you have the board set up, use the Arduino code on Canvas "**10_1_potentiometer_code_outline.ino**" or in Appendix 10.1. Ensure the circuit prints the potentiometer output to the serial monitor. Remember that a potentiometer acts as a variable resistor.



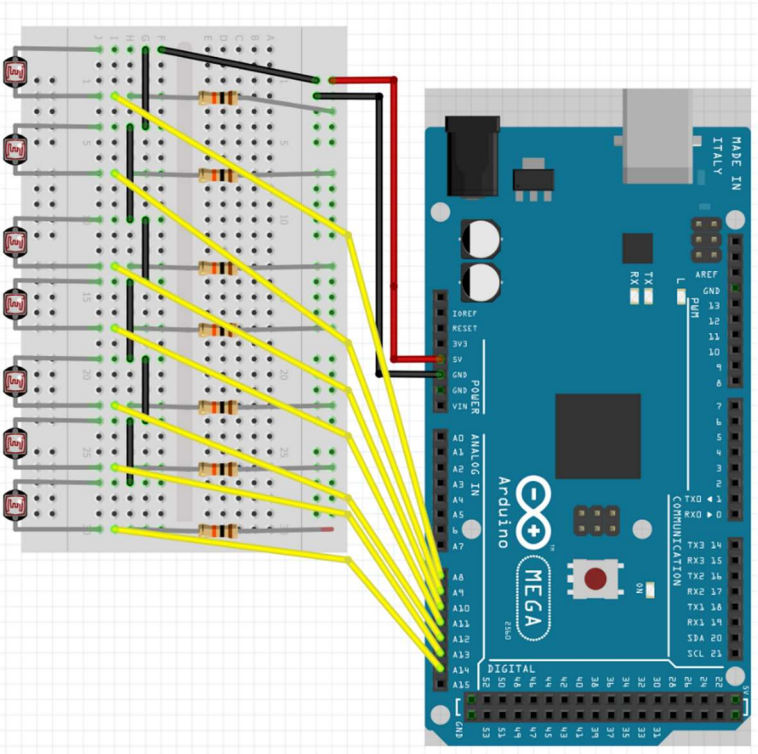
1. Do counterclockwise or clockwise twists of the potentiometers increase the value printed to the serial monitor?

2. How can you change the circuit so that turning the potentiometer clockwise increases or decreases the value printed on the screen? (the opposite of what your potentiometer currently does)

Challenge #2: Photoresistors

Use photoresistors together with regular resistors to build voltage dividers as shown in the wiring diagram. The voltage dividers use 1kΩ resistors. Each voltage divider should then be connected to one of the Arduino analog pins (A8-A14 in the figure).

Once you have the board set up, use the Arduino code on Canvas "**10_2_photoresistor_code_outline.ino**" or in Appendix 10.2 to read the values from each of your photo resistors and print them to the serial monitor. Test this code to be sure each photo resistor is working properly; they should read similar values.



Try placing the photoresistor array (breadboard) close to different colored surfaces and then try it out on a line.

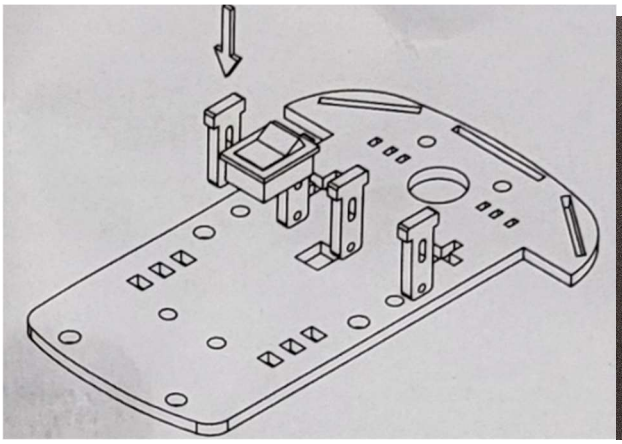
1. What is the best distance for your sensors to be from the surface to properly differentiate when the sensors are hovering over black or white (line or no line)?
2. How do surrounding light and shadows affect the readings?
3. How sensitive are the sensors? Are they able to detect small changes in light intensity?
4. If the sensors are not reading the difference in colors properly, how can you improve the reading?
5. If you add an LED next to the photoresistors so that the LED shines on the surface they are reading, does this help the reading?
6. Is there anything else you can do to improve sensor accuracy?

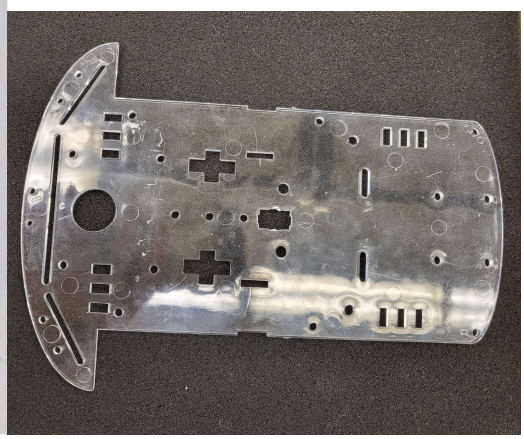
Challenge #3: Chassis

Now it is time to put together the cart chassis. Assembly will require 2 wheels and 2 motors with a caster as a third wheel. You may design and 3D print your own chassis **after** you build and test the given cart assembly.

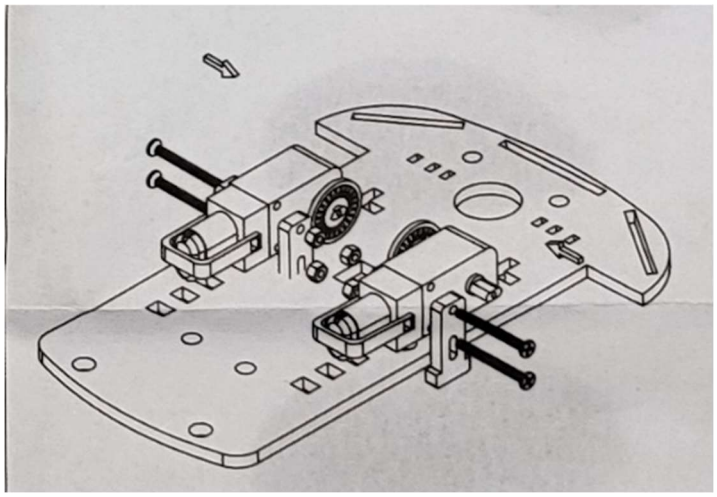
Note: this challenge is difficult. Some parts fit very tightly and require some patience to attach. Try not to break anything!

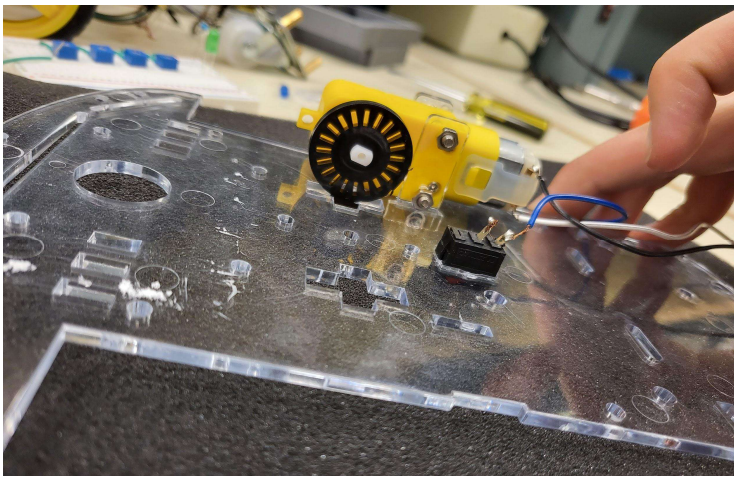
1. Insert the switch into the slot in the middle of the chassis, then insert the plastic fasteners on either side. It may be helpful to attach one pair of fasteners and move on to step 2, then come back after attaching the motor.

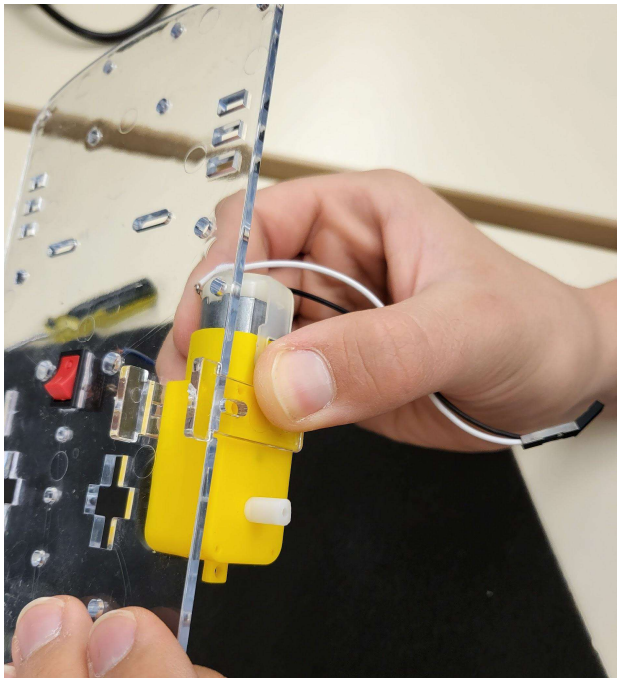


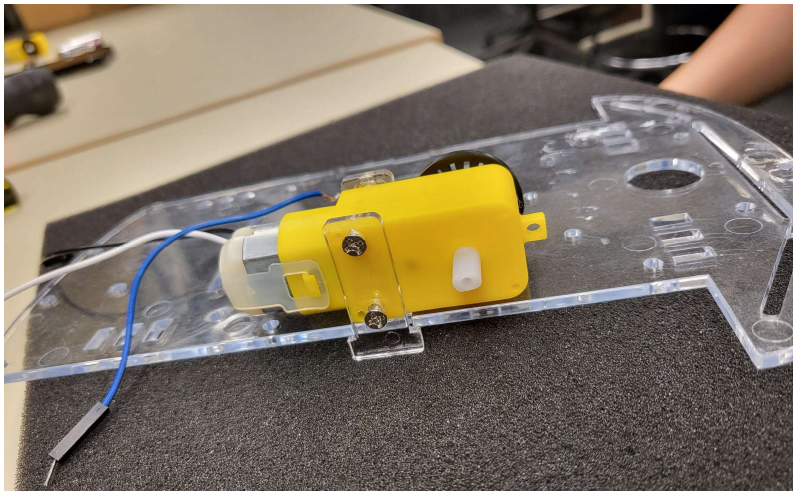


2. Attach the black encoder disc to one side of each motor. Then attach the motor to the chassis by placing nuts on the fastener nearer to the switch and inserting a 30mm screw bar into the fasteners and the motor. Two bars through the motor in each hole in the fastener should fix it in place. Repeat for the other motor.



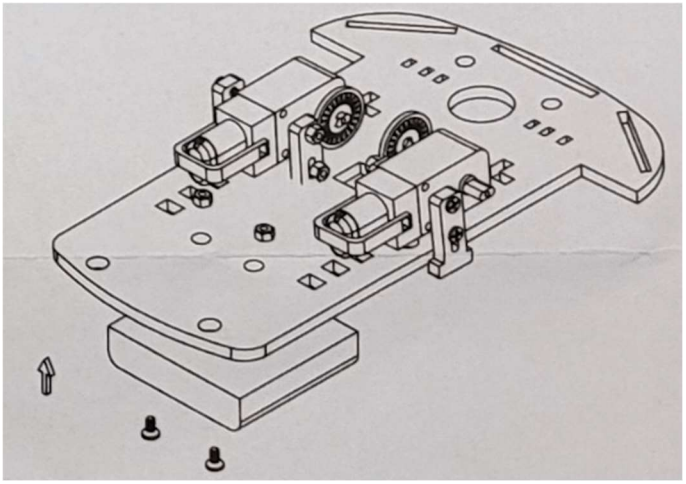


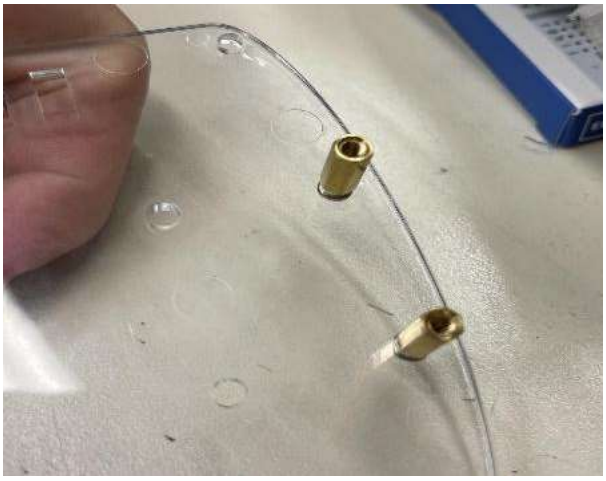


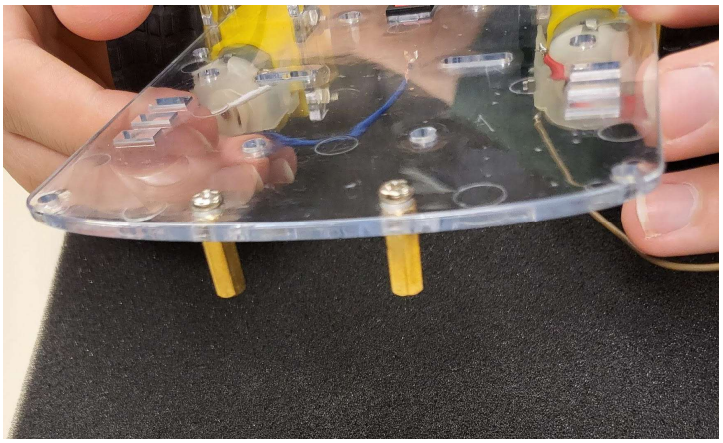


3. Screw two standoffs (the metal spacers) to the front of the chassis through the holes

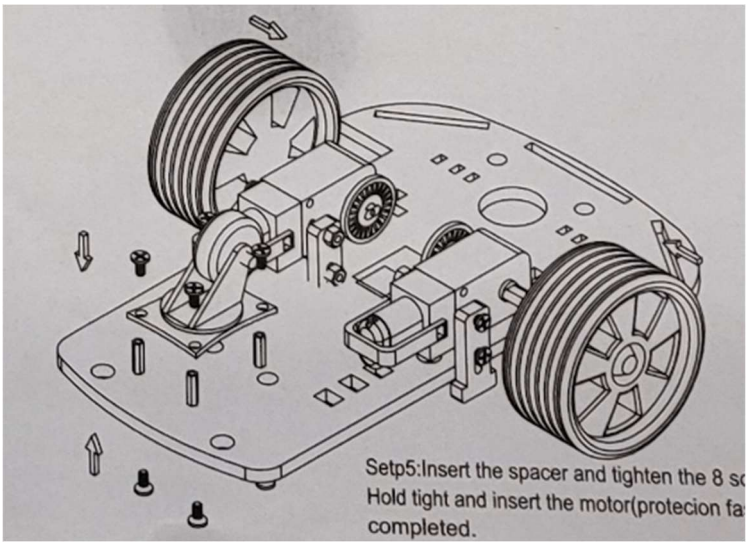
closest to the edge using the 6mm screws. The back edge should look like this, with the standoffs on the same side of the chassis as the motors.

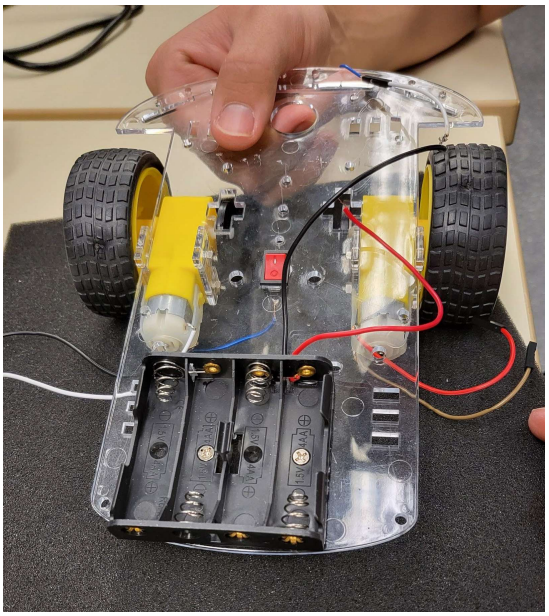


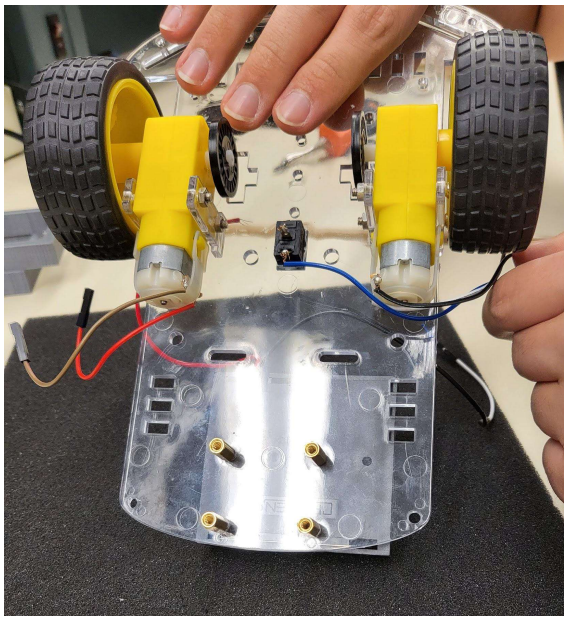




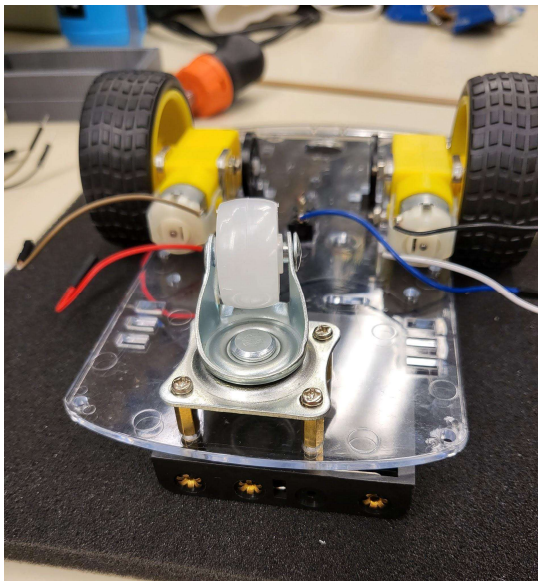
4. Next, attach the battery container to the chassis. Attach two screws through the case, and use the other two standoffs to fix them in place. Shown below is a schematic and a picture of what the result should look like. Your cart parts **should not** have wires soldered to them yet (even though the examples show soldered connections).

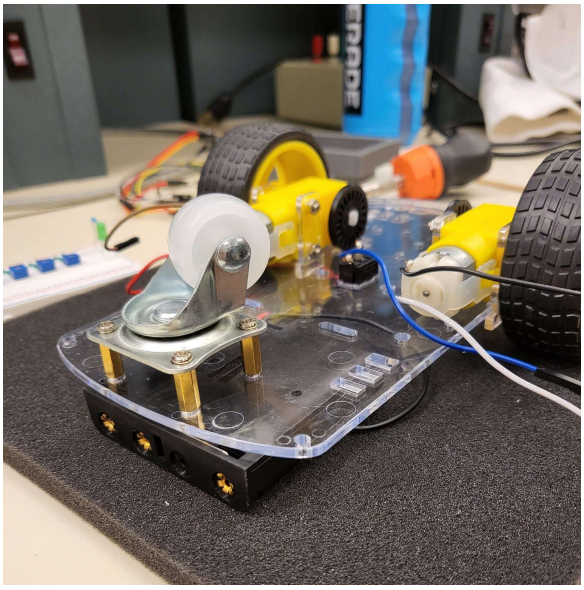

Setp5:Insert the spacer and tighten the 8 sc
Hold tight and insert the motor(protecion fa
completed.

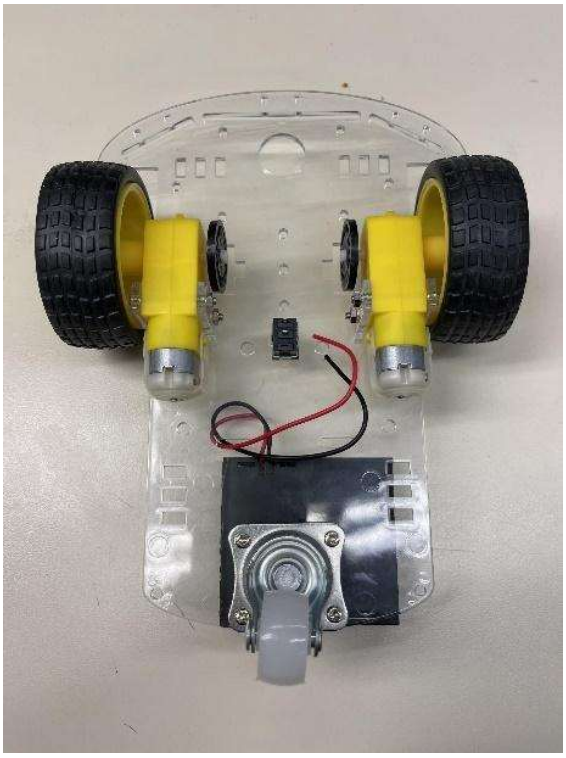


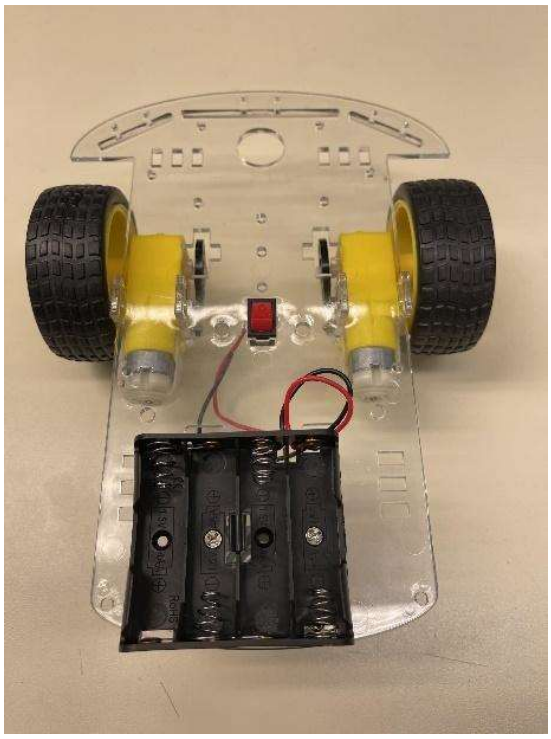


5. Attach the caster to the standoffs using the remaining screws. Your cart should now look like this:









TA CHECK OFF:

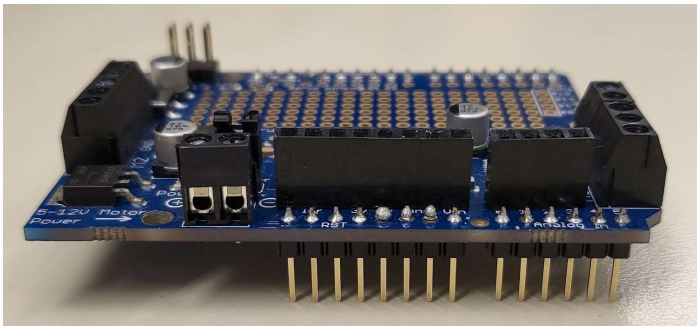
Show your TA: Demonstrate the cart BEFORE supplying any power to avoid shorting components!

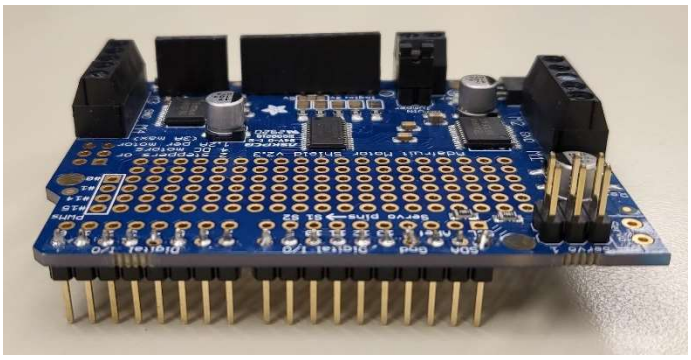
Challenge #4: Motor Driver and Arduino Mega

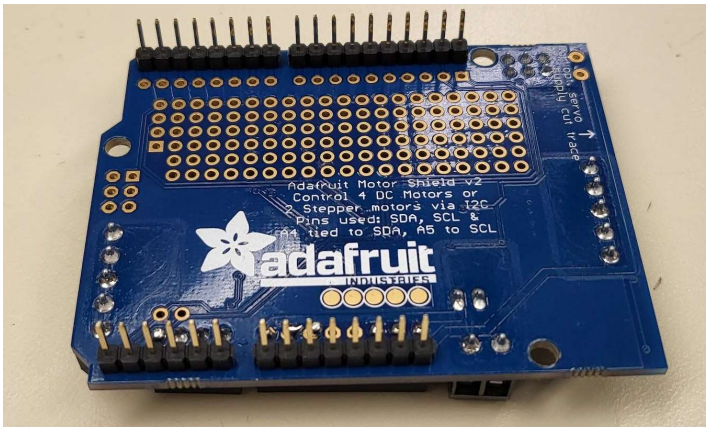
Prepare the motor shield by soldering the header pins onto the motor shield. *It helps to solder your first one or two pins of each strip while it is attached to an Arduino or a breadboard for alignment purposes.*

Female header pins should be added to the inner row of the connections at the bottom of the motor shield (power and analog pins) for easy connectivity to other devices using jumper wires.

Put **male header pins** on the outer edge of the motor shield (as seen in the figures) which should have the longer ends facing downwards to connect to the Arduino.







TA CHECK OFF:

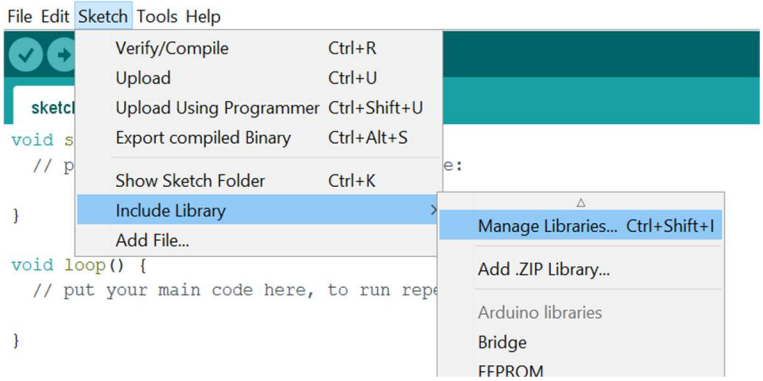
Show your TA: Demonstrate the soldering before you connect power to avoid shorting components!

- Next, **add the motorshield library** to the Arduino by following the instructions below.

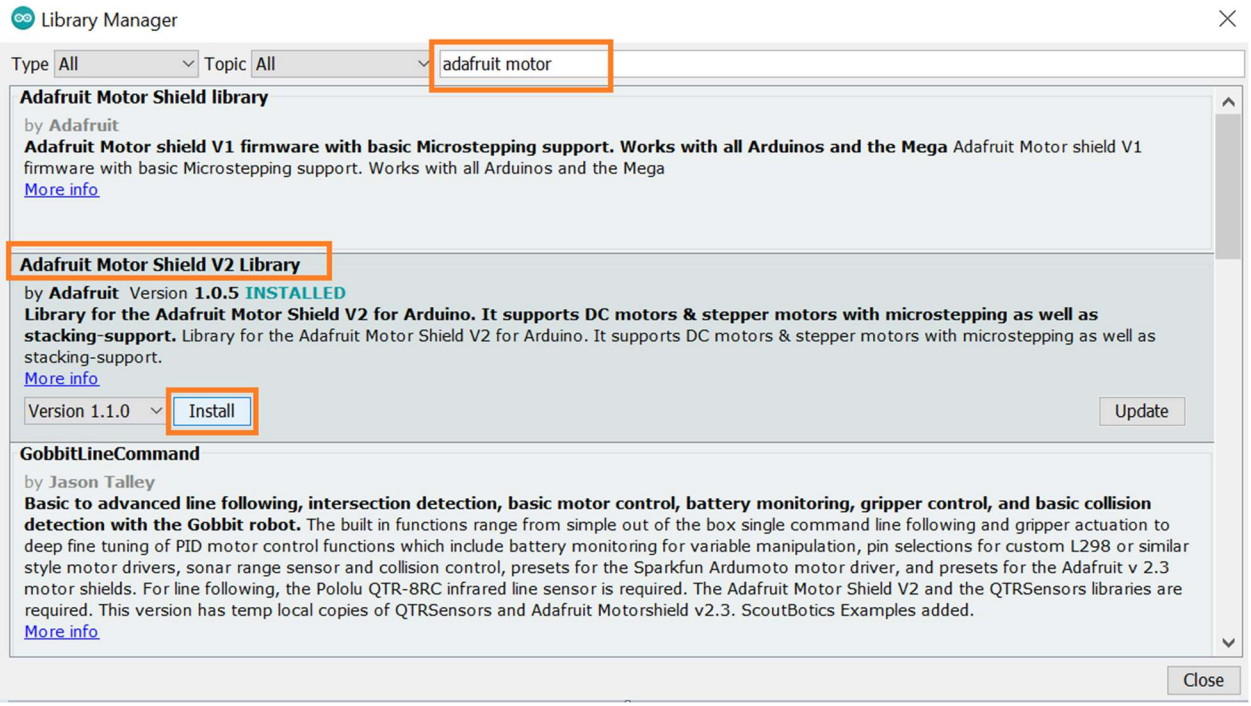
  Download and install the Adafruit motor shield library for Arduino.

  1. From the IDE open up the **Library manager**…

     Go to "**Sketch**" -> "**Include Library**" -> "**Manage Libraries**"



  2. Type in "adafruit motor" to locate the library. Click "**Install**".

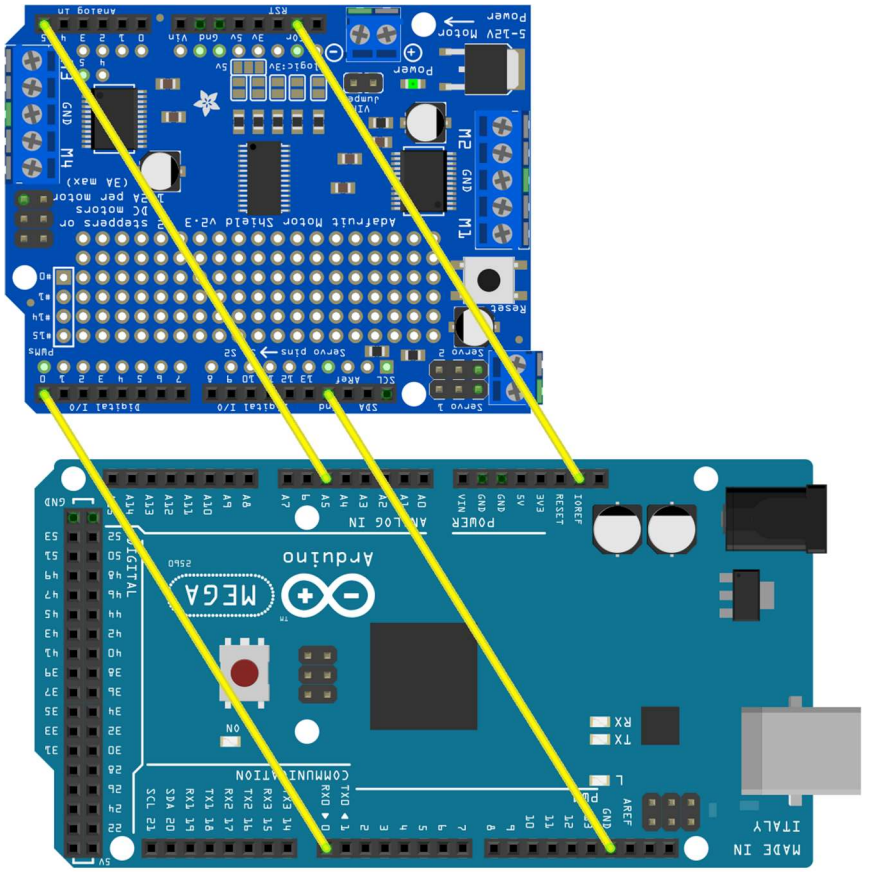


Without the motors connected, but with the motor shield attached to the Arduino.

Use the Arduino code on Canvas "**10_3_motor_driver_code.ino**" or in Appendix 10.3.

Ensure the code compiles and uploads without error.

Note: The wires in the schematic represent the four corner pins of the motor shield and their location on the Arduino board.


Adafruit Motor Shield v2.3
Arduino
MEGA
POWER
ANALOG IN
DIGITAL
COMMUNICATION
MADE IN ITALY
fritzing

Challenge #5: System Integration Part 1

Next, you will need to connect wires to the motors and solder them.

First, cut **four** strands of roughly 6” wire and strip a very small amount off **one** end. Loop the stripped end of each wire through both metal loops on the side of each motor.

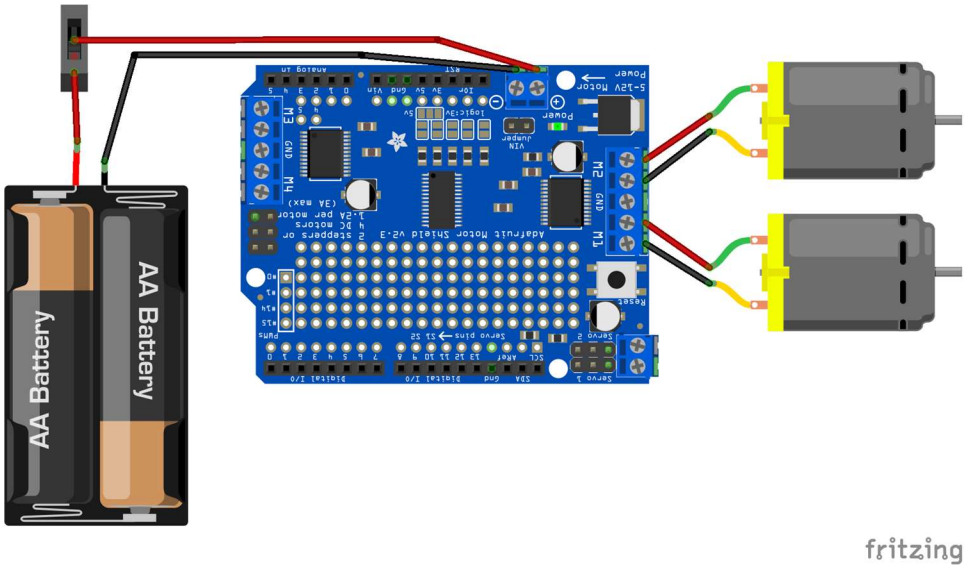
Second, solder all four connections to hold the wires in place. You may find that using an Arduino jumper wire for this works well.

When you are done, both motors should have wires connected to each end, as shown below. The type of header is up to you, but female wires may be easier to work with down the road.

Next, repeat a similar process for the switch on the cart. A diagram is provided below.

1. Solder a jumper wire from one end of the switch to the positive lead of the power supply on the cart.
2. Solder a jumper wire onto the other end of the switch so that the other end of the jumper can be plugged into the positive power supply port of the motor shield.
3. Solder a jumper wire from the negative lead of the power supply to the negative power supply port of the motor shield.

At this point, you should be able to flip the switch to turn the Arduino and motor shield on without having to connect the Arduino to your computer.



Looking at the diagram above, connect your motors to M1 and M2 on your motor shield.

- Ensure both power sources are available to connect, one for the Arduino Mega and one for the motor shield.
- **Do not** connect the power sources and have your computer plugged into Arduino (through the USB) at the same time.

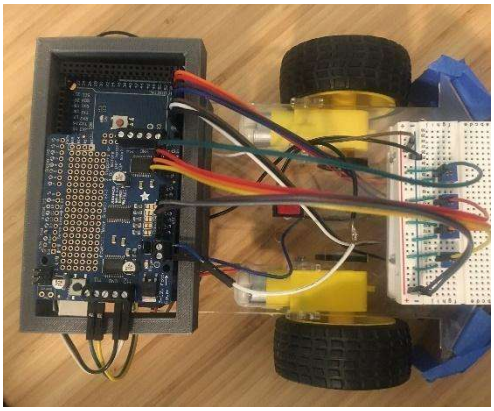
Next, test the connection using the code in Appendix 10.3 or write your own. The Appendix 10.3 code should move the cart forward for 3 seconds, pause, and then backwards for 3 seconds.

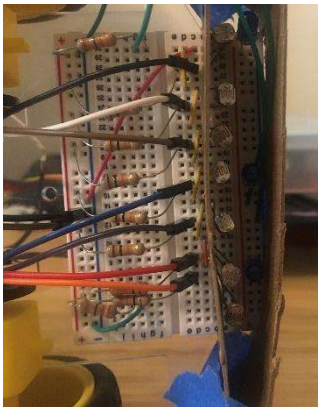
After reading through the code, try another maneuver such as going in a circle or making a figure 8. Be careful as the cart is fast. **Do not** test on a table or elevated surface. Instead, test the cart in an open area free of obstacles.

1. Does each knob work? How do you know?
2. Does each photoresistor work and sense the difference between a black and white surface? How do you know?
3. How do you know each motor is connected correctly and turns when it should?
4. Does the cart drive forward and backward?

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. One option is to change the polarity of your wires coming from your motor shield from M1 and M2 by switching the red and black wires. Another option is to switch which motors are connected to M1 and M2. 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 #5: System Integration Part 2

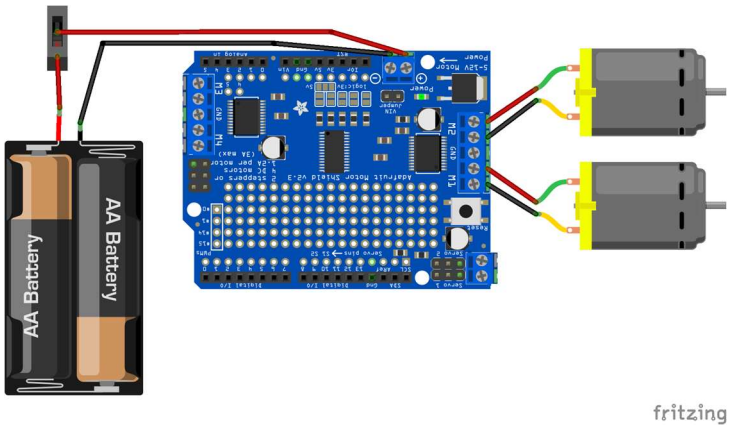




There are many ways to put your robot together from here. An example is provided above. Remember to consider the height of the photoresistors from the ground for good sensitivity measuring various surfaces. It may be beneficial to use thick paper to keep the photoresistors aligned. Test different distances to make sure the photoresistors are reading properly.

Tape helps with nearly everything including holding your breadboard in place, holding the Arduino Mega to a specific spot on your cart, keeping wires organized, etc. You may also try the adhesive backing that comes on some breadboards to stick the breadboard to the chassis. The adhesive backing only works once, so make sure about your placement before removing the backing!

Wire the cart like you did for Challenges 1-3, then connect the negative lead of the battery pack to the negative power lead of the motor shield. For the positive lead, solder the wires such that the switch is between the battery pack and the Arduino. When you flip the switch and are disconnected from your computer, the Arduino should power on and off. The same diagram from part 1 is provided for convenience.



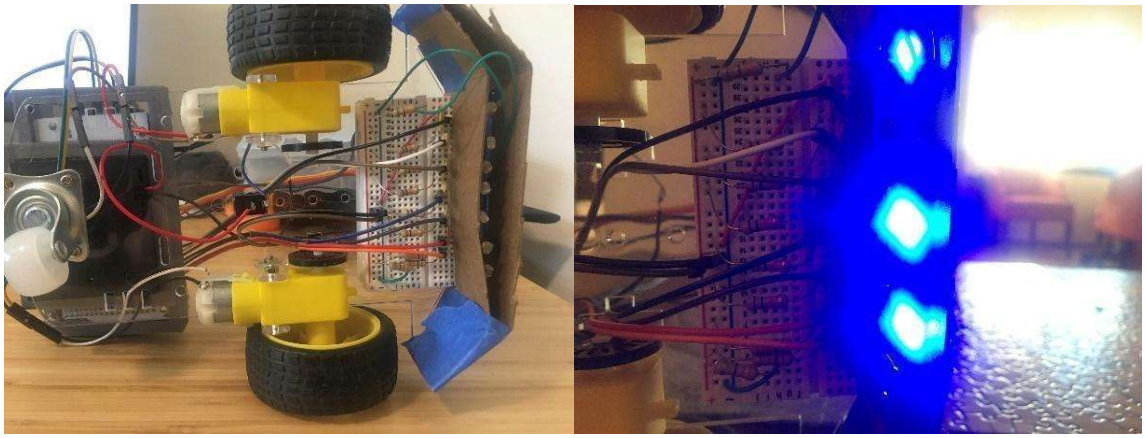
Run each of the programs from Challenges #1 - #3 (10.1-10.3) individually. The questions for

challenge #5 are provided below *again*. Only answer the questions once on the assignment.

1. Does each knob work?
2. Does each photoresistor work and sense the difference between a black and white surface?
3. How do you know each motor is connected correctly and turns when it should?
4. Does the cart drive forward and backward?

Challenge #6: Hardware (complete at least 3) Light Shield and Light Source Additions: (Highly Recommended)

One problem that arises is the robot's sensitivity to ambient light changes and shadows. Add a light shield or additional light source on your robot to help mitigate this problem. Test your addition using the 10.2 code to see whether it helps the photoresistors read more accurately.



Increase the Battery Power:

The given battery pack does not provide enough power to run the motors at a high speed without causing a "brownout." A brownout occurs when the system attempts to draw more power than the batteries can provide. The brownout will cause the Arduino to shut off and restart (likely causing the cart to start calibrating again). Try adding additional batteries to power the motors and Arduino. You can also separate the motor power from the Arduino power to prevent brownouts from restarting the Arduino.

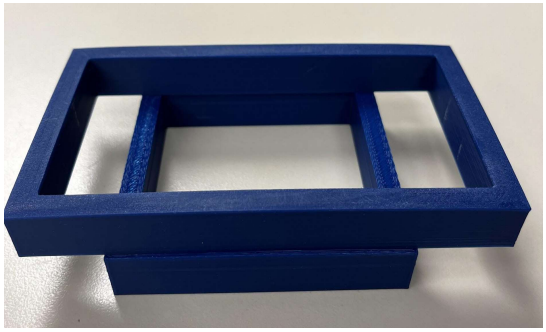
PCB:

Currently, everything is implemented on a breadboard. A more secure and permanent option would be to implement the same circuit on a PCB (Printed Circuit Board). A PCB design tutorial is provided as a separate document if you're interested. Premade PCBs are also directly available from your TA.

3D Printed Chassis or Additional Parts: Printed Chassis or Additional Parts:

Many parts of the chassis are awkward to use, especially when trying to place different

components. Design a chassis you can 3D print and use specially for this robot. Consider what would be convenient to have. Some ideas for integrating into the chassis: different dimensions to add control, a built-in light shield, special spots for the photoresistors, battery case, and the Arduino. A basic model is provided, but you are encouraged to make changes as you see fit.



PID Indicator:

Currently, only knobs are used to set the value on your PID controller. Create your own real time indicator of what your PID values are. Be creative using LEDs, servos, or other circuitry to illustrate these values. Knowing if your knob is all the way high or low is beneficial for debugging. Using pulse width modulation (PWM) to change the intensity of an LED is an easy way to display the status of each knob. Variable voltage dividers will also work in changing the intensity of an LED.

Collision avoidance:

It is unfortunately very common that these robots will run into a wall, another robot, or fall off a table. Using sensors or code additions, have your Arduino detect one of these and avoid a disaster. How could you implement detection within the code? Could you implement it without changing the code?

Propose your own:

Propose your own hardware addition using design tools, sensors, and/or actuators. Must be approved by a TA or Professor.

Challenge #7: Calibration

Calibrating your sensors is important because sensor readings can change with changes in sensor position or alignment, changes in surrounding lighting, or collisions.

This challenge will walk you through the process of calibrating your device using code found on Canvas "**10_4_calibration_code.ino**" or in Appendix 10.4.

The goal of calibration is to take all possible readings from each sensor (shown at the top of the two figures), and then map those readings to a uniform measure. Mapping makes it easier to determine where the line to be followed is.

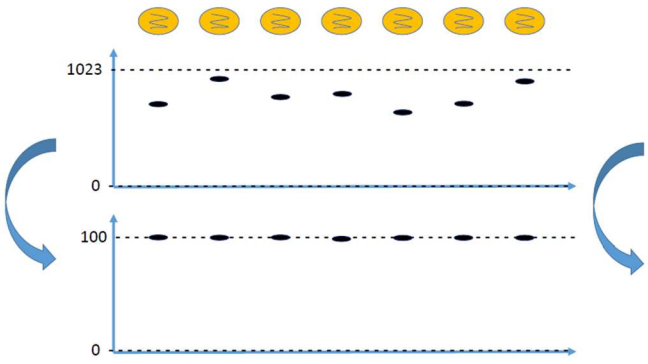


Figure 1: Mapped reading for a **black line**

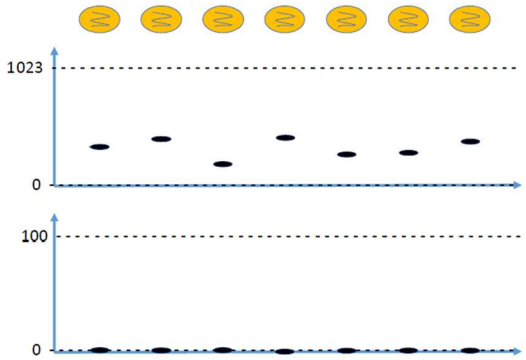


Figure 2: Mapped reading for a **white line**

Note: before implementing the new code found on Canvas "**10_4_calibration_code.ino**" or in Appendix 10.4, 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. For example, using digital pin 31 since pin 13 is hiding below the Adafruit Motor Shield. The LED is used to help you know what step of the calibration you should be performing.

Read through the code and comments found on Canvas "**10_4_calibration_code.ino**" or in Appendix 10.4. Test the 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 *light surface.*

2. After a few seconds, the LED will *blink quickly.* 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 a light surface.

3. The LED will then *blink slowly* again, giving you time to move the photoresistors over a dark surface or line. Ensure that each photoresistor is fully over the dark surface.

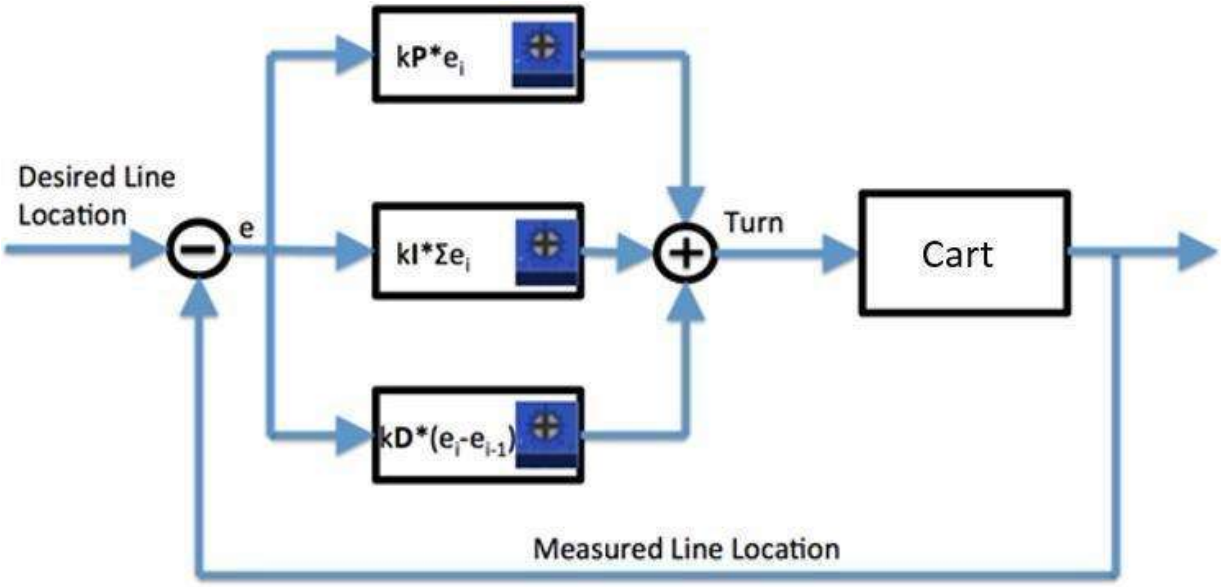
4. Again, after a few seconds, the LED will *blink quickly.* Again, it is important not to touch the cart or cast any shadows over it because it is finding the nominal photoresistor reading of each pin for what correlates with the dark surface.

5. Now, the sensors should be calibrated, and the Arduino should be able to distinguish between dark and light. The readings should be more uniform when moving between 0 and 100. You can see these readings on the serial monitor.

6. Error, or distance from the centerline of the cart, is then calculated through a weighted average. The weighted average is calculated in the PID control section.

7. After calibration, what do the photoresistors read when a black surface is placed in front of them? What about when a white surface is placed in front of them? (hint: use the serial monitor)

Challenge #8: PID Control



Apply what you learned on PID control! In the past four sections, you have been putting together each of the main components, and now you will organize your code to connect everything together. The first step is to quantify your error, e. This is what your photoresistors are for, and the error can be found by comparing your measured value to your desired value.

$$error = (measured\ value - desired\ value)$$

You need to operate on this error. Error can be broken down into 3 main operations: proportional, integral, and derivative control.

Proportional Control (P): takes your current error value and scales it by your proportional gain, kP. This scaling transforms your error into a turn rate for correcting your error. The higher the error, the higher the turn rate. The cart's turn rate is controlled through your left and right motors. If your error indicates that you are to the right of the line, you will need to increase the speed of your right motor and or decrease the speed of your left motor. The next figure explains proportional control graphically.

Increasing "P" will increase the sensitivity of your cart's sensors.

Integral Control (I): takes into account a sum of your past errors. This can be implemented into your code with:

$$sumerror = (sumerror + error)$$

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

Increasing "I" will help your cart get rid of small deviations from the line.

Derivative Control (D): can predict an increase or decrease in your error and prepare your turn rate in advance. This is helpful for sudden and sharp turns. By taking your current error and subtracting your previous error, you will be able to see how much your error has changed during one time step.

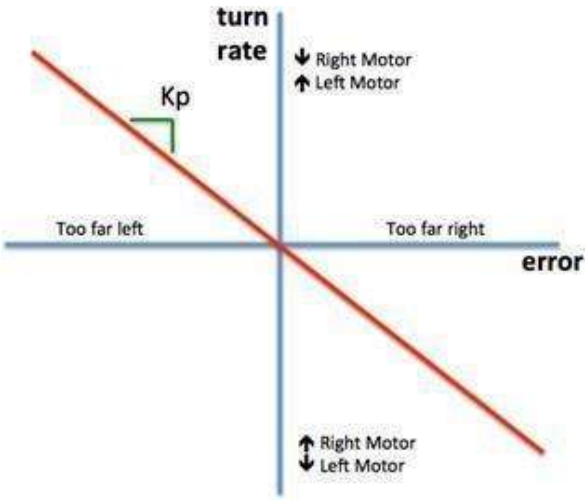
$$deriverror = (error - previous\ error)$$

Unless corrected, 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 right or left motor speed to counter sharp changes. Increasing "D" will help your cart follow a line. Increasing "D" by too much will cause the car to overcorrect, and lose the line.

Now, we can add these terms together to calculate our turn rate for controlling our cart.

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

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

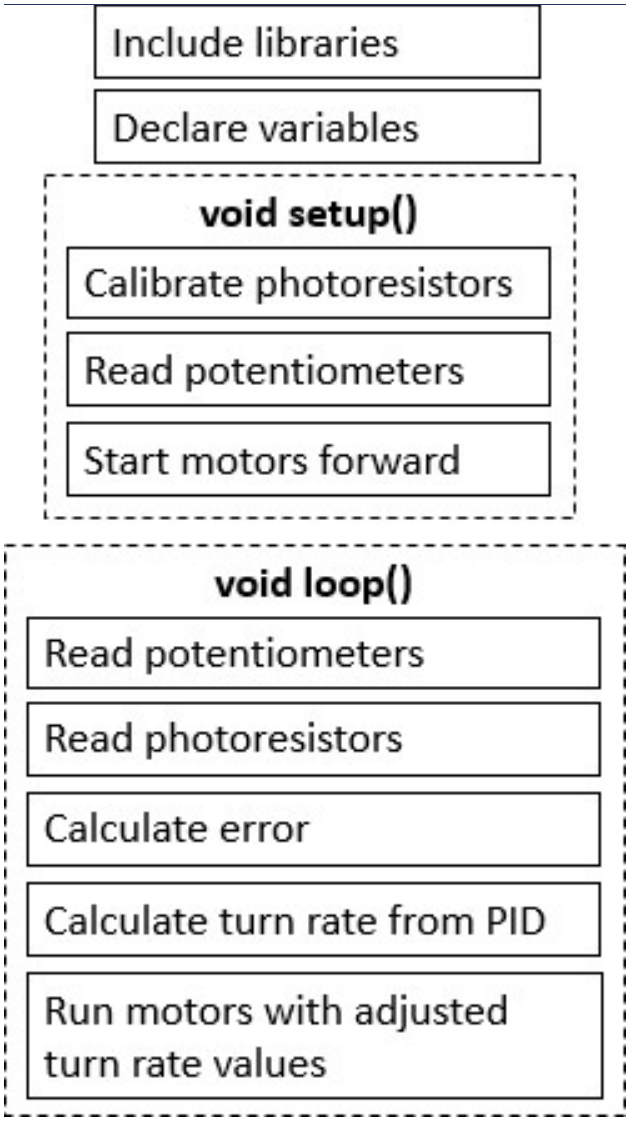


See *code flow* for a rough outline to better understand the code.

Use the Arduino code on Canvas "**10_5_line_follower_code.ino**" or in Appendix 10.5.

Code Flow: Code outline can be found in Appendix 10.5

Note: you must change the values of P, I, and D in the code. The values given are phony values that will not produce a functional cart! Experiment with your team to find values that work for your cart.



TA CHECK OFF:

Show your TA: The PID constants you decided upon in the 10.5 code

Challenge #9: Competition

1. Line Courses

Each course will have different criteria for score including (1) how sharp the robot can turn while following a line, (2) time to complete certain portions of the course, and (3) a mystery condition. With the ability to control your speed and PID knobs between challenges, your robot will compete against your classmates' robots.

2. Innovation and Creativity Challenge

Using any of the sensors or actuators you have learned about throughout the quarter or any others that are mentioned in previous challenges, you will improve your robot's control, usefulness, or increase the level of interactivity and fun (you will get bonus cool points!).

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 Outline */
/* 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
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 Outline */
/* 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 Code Outline */
// 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); // Motors can be switched here (1) <--> (2)
Adafruit_DCMotor *Motor2 = AFMS.getMotor(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 Code ECE 5 Lab 4 */
// Variables and Libaries for Motor
// Variables for Light Sensors
int LDR_Pin[7] = {A8,A9,A10,A11,A12,A13,A14}; // Arrays are used top simplify the code
int LDR[7]; // these are variables that have multiple elements to each variable name
// LDR_Pin hold 7 values and A8 is the 0th element and A11 is the 4th element
// Calibration Variables
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.};
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 = 40;
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 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;
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.;
```

```
}
CriteriaForMax = 2; // max should be at least twice as big as the other values
if (MxRead > CriteriaForMax*AveRead)
{
if (im1!=0 && im1!=6)
{
im0 = im1-1;
im2 = im1+1;
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)
{
im2 = im1+1;
WeightedAve = ((float)(LDR[im1]*im1 + LDR[im2]*im2))/((float)(LDR[im1]+LDR[im2]));
error = -1*(WeightedAve - 3);
}
else if (im1 == 6)
{
im0 = im1-1;
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(" ");
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.println(error); // this will show the calculated error (-3 through 3)
delay(100); //just here to slow down the output for easier reading if wanted
} // end Print()
```

10.5 Line Follower Code

```
/* ********************************************************************************************** */
// * 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
#include <Wire.h>
#include <Adafruit_MotorShield.h>
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *Motor1 = AFMS.getMotor(1); // you may switch 1 with 2 if needed (if you see motors responding to error
in the opposite way they should be.
Adafruit_DCMotor *Motor2 = AFMS.getMotor(2);
int M1Sp = 20, M2Sp = 20; // this is the nominal speed for the motors when not using potentiometer
int M1SpeedtoMotor, M2SpeedtoMotor;
// Variables for Potentiometer
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
int LDR_Pin[7] = {A8,A9,A10,A11,A12,A13,A14}; // Many arrays are used in this code to simplify it
int LDR[7]; // these are variables that have multiple elements to each variable name
// LDR_Pin hold 7 values and A8 is the 0th element and A11 is the 4th element
// Calibration Variables
int led_Pin = 31; /// This is a led set up to indicate what part of the calibration you are on.
float Mn[7]; /// You could use pin 13 instead which is a built in LED to Arduino
float Mx[7];
float LDRf[7] = {0.,0.,0.,0.,0.,0.,0.};
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 = 40;
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()
{
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;
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.;
}
CriteriaForMax = 2; // max should be at least twice as big as the other values
if (MxRead > CriteriaForMax*AveRead)
{
if (im1!=0 && im1!=6)
{
im0 = im1-1;
im2 = im1+1;
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)
{
im2 = im1+1;
WeightedAve = ((float)(LDR[im1]*im1 + LDR[im2]*im2))/((float)(LDR[im1]+LDR[im2]));
error = -1*(WeightedAve - 3);

}
else if (im1 == 6)
{
im0 = im1-1;
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;
if (Turn < 0) { M1P = Turn; M2P = -Turn;} //One motor becomes slower and the other faster
else if (Turn > 0) {M1P = Turn ; M2P = -Turn;}
else {M1P = 0 ; M2P = 0;}
} // 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()
```