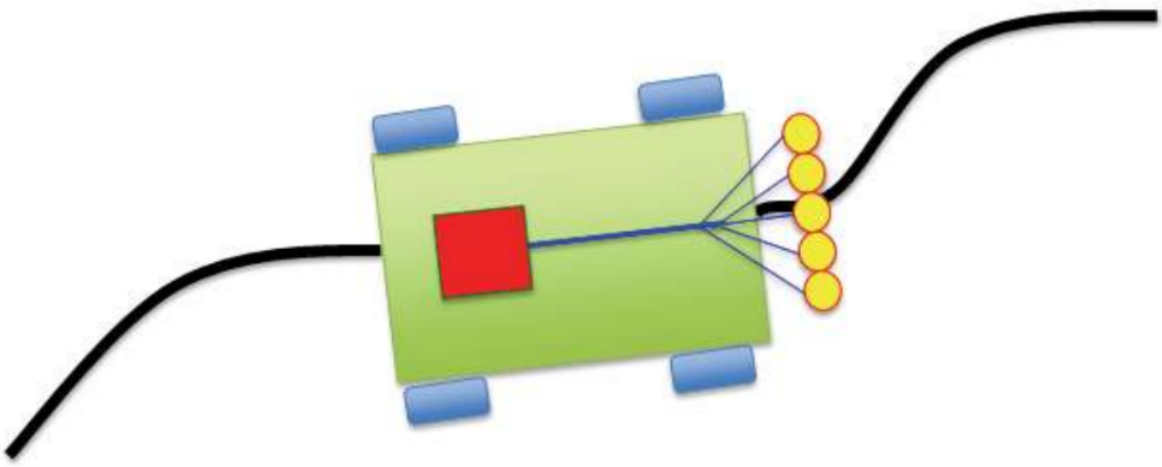
Robotics



Electrical and Computer Engineering 5

Embedded systems and Control:
Line Following Robot

Developed by Prof. Karcher Morris, Dr. John Eldon, and MingWei Yeoh

Code can be found on Canvas

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.

We will start from building and testing part of the system, and this is a good time to divide up the work across the team. Then, we will connect every part together and start running the robot.

Outline

1A) Potentiometers
1B) Photoresistors
1C) Motor driver and Arduino Mega
1D) CAD The Chassis
2) Assemble the robot
3) Follow a line
4) Competition
    Inspiration

What You Will Need

Materials:

- 1 Arduino Mega
- 1 USB Cable A-B
- 1 Cart Chassis – 3D Printed
- 1 Caster
- 1 L298N Motor Driver
- LEDs
- 2 DC Motors
- 2 Wheels
- 2 Breadboards
- Double A and 9V battery packs
- 4 Potentiometers
- 7 Resistors (10k Ohms)
- 7 Photoresistor
- 6 AA Batteries

Machinery:

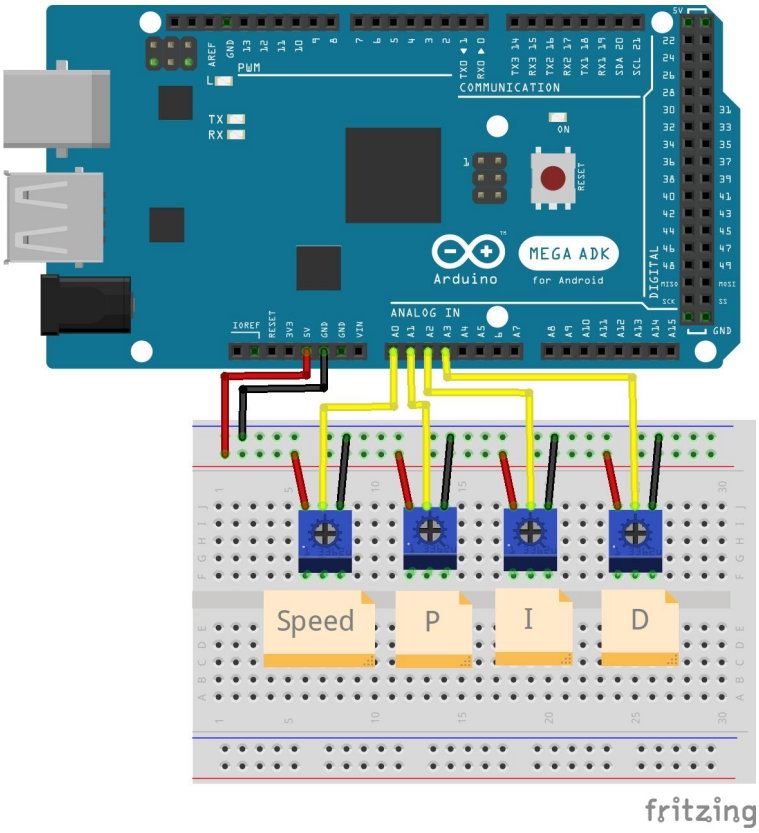
- Computer / Laptop
- Solder Station
- Screw Driver

Software:

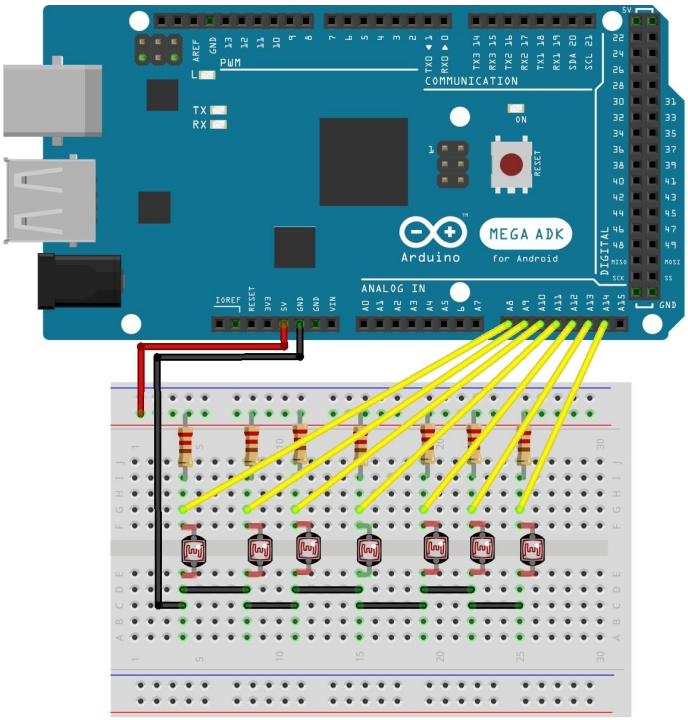
- Arduino Software (IDE)

Challenge #1A: Potentiometers

For the first challenge, set up potentiometers on a breadboard 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 potentiometer code in canvas. Make sure to **fix the FIX MEs.** Test this to ensure it prints the potentiometer output to the serial monitor. Remember that a potentiometer acts as a variable resistor. **Do counterclockwise 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 #1B: Photoresistors



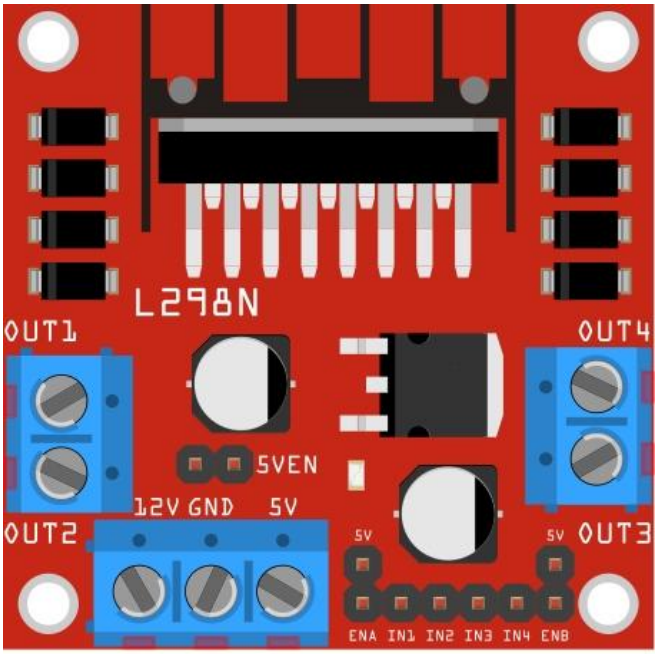
Develop multiple voltage dividers with photoresistors instead of potentiometers. The voltage dividers should use **10kΩ** resistors for optimized performance.

Once you have the board set up, use the Photoresistor code. This reads the values from each of your photo resistors and prints them to the serial monitor. Test this to be sure each photo resistor is working properly. **Be sure to fix the FIX MEs in the code.**

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!)?**

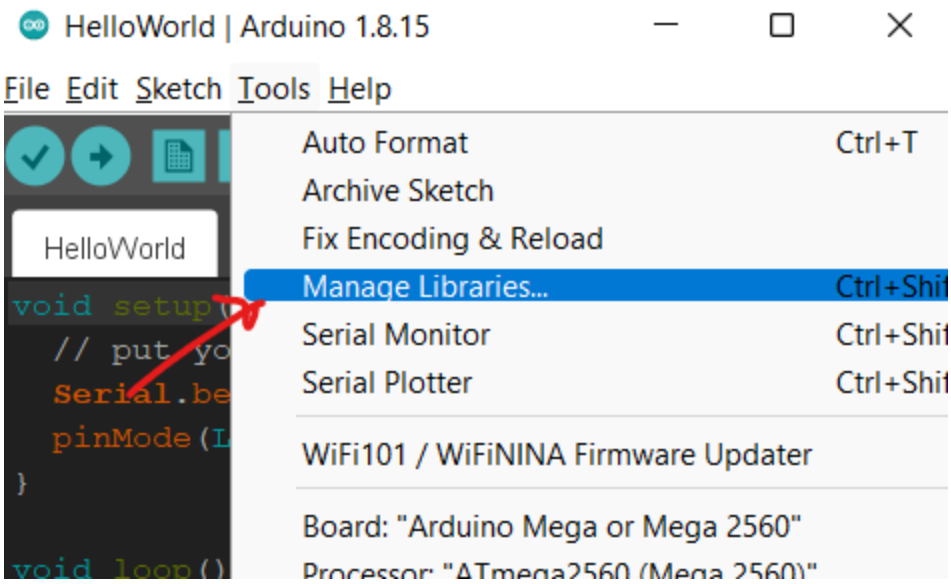
Challenge #1C: Prepping the Motors

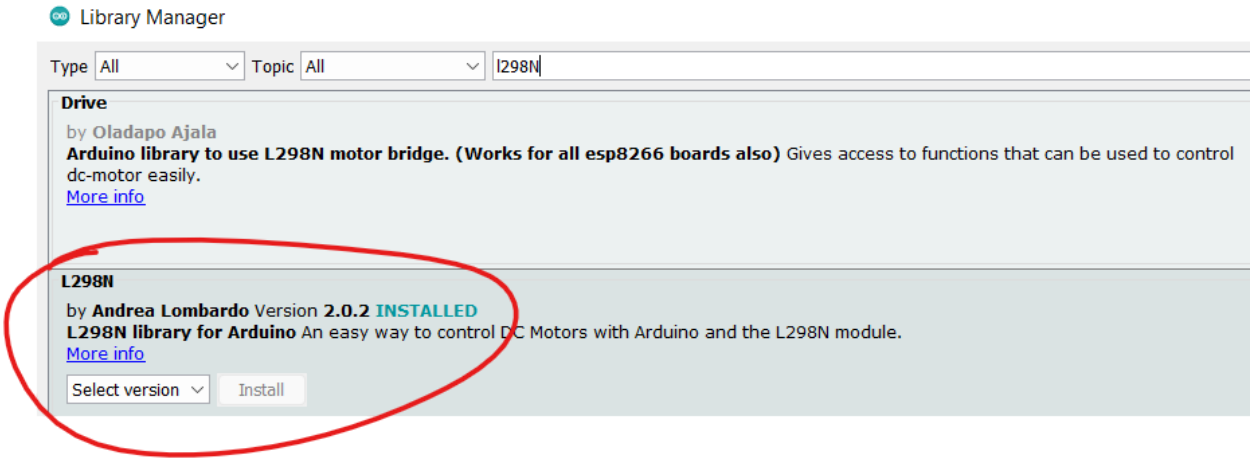
We will be using the L298N Motor driver to drive both of our DC motors. These motor drivers are very easy to control.



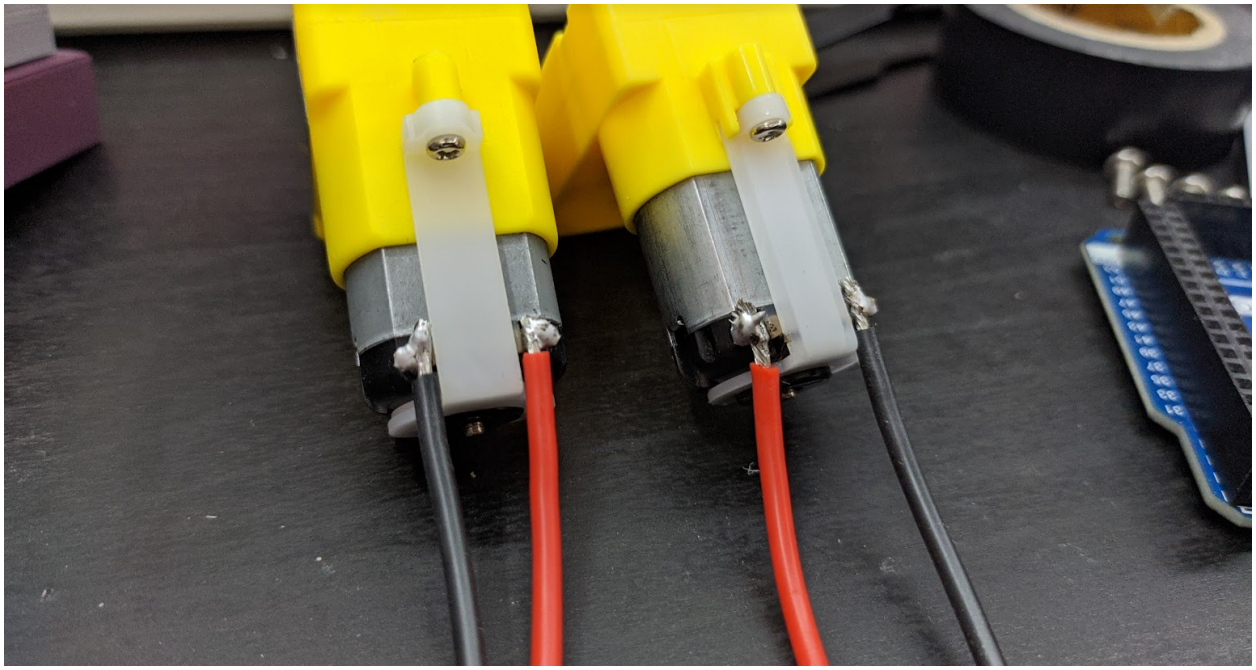
IN1 & IN2 correspond to the direction that the motor is driven (OUT1 & OUT2). If **IN1** is high (connected to 5v) the motor spins one way. If **IN2** is high then the motor spins the other direction. **ENA** takes in a PWM signal (like a servo!) to vary the speed that the motor goes. When you buy these from Amazon, there is a jumper pin between **5V and ENA** to simply drive the motors in full forward or reverse mode without the need for PWM control. You can remove and discard this jumper as we won't be running these motors at full speed all the time. For precise control, we will be varying the speed that our motors go. The same can be said for IN3, IN4, and ENB, except that they are controlling the other motor (OUT3 & OUT4).

To control it with code we will need to **download the L298N Motor driver library by Andrea Lombardo with the Library Manager on the Arduino software**

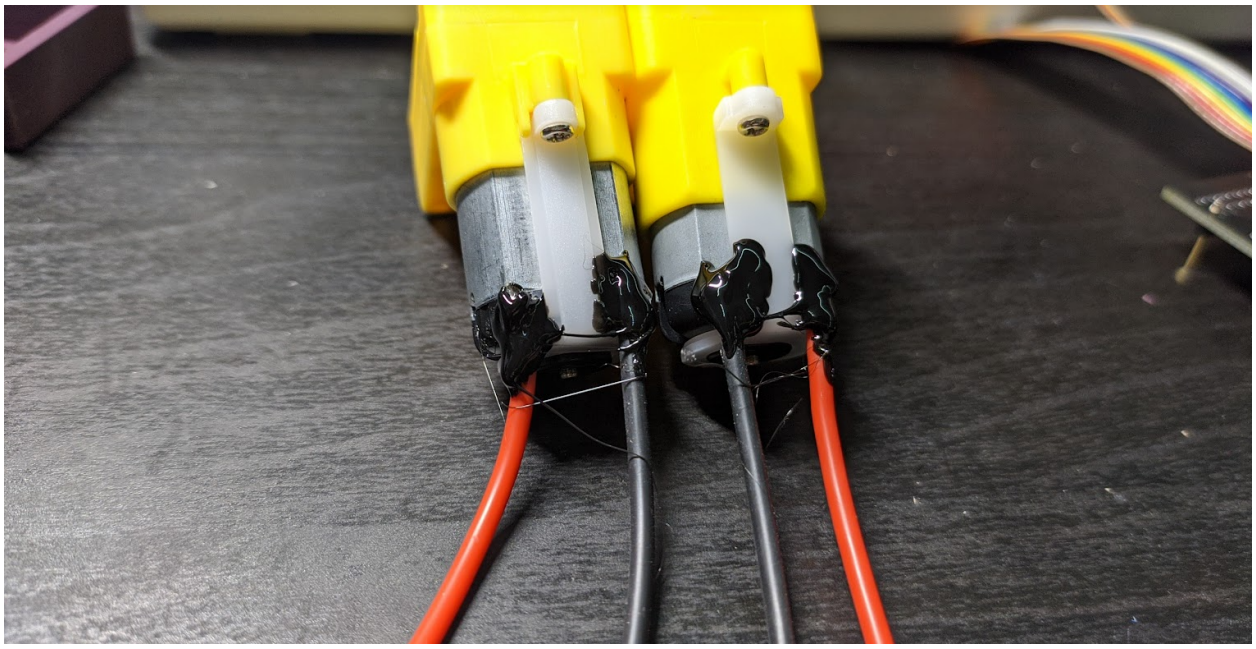




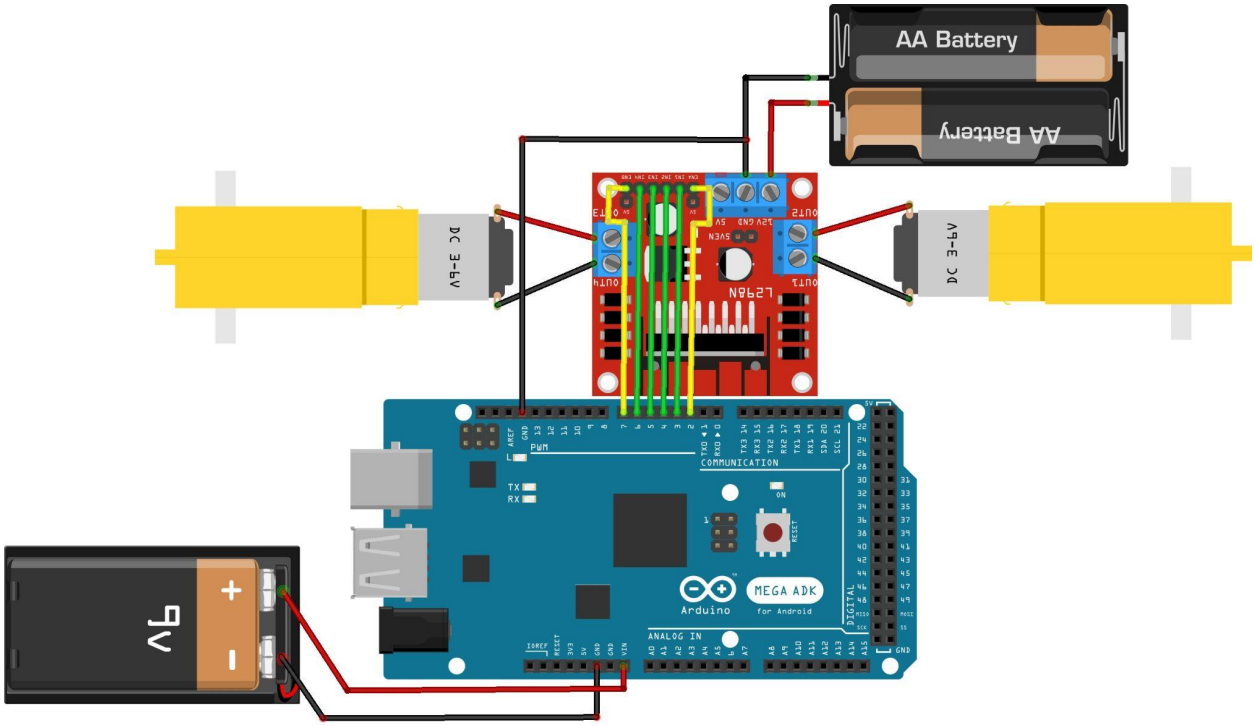
Once you have done that, we should set up our motors. We need to solder wires onto our motor so that we can control them!



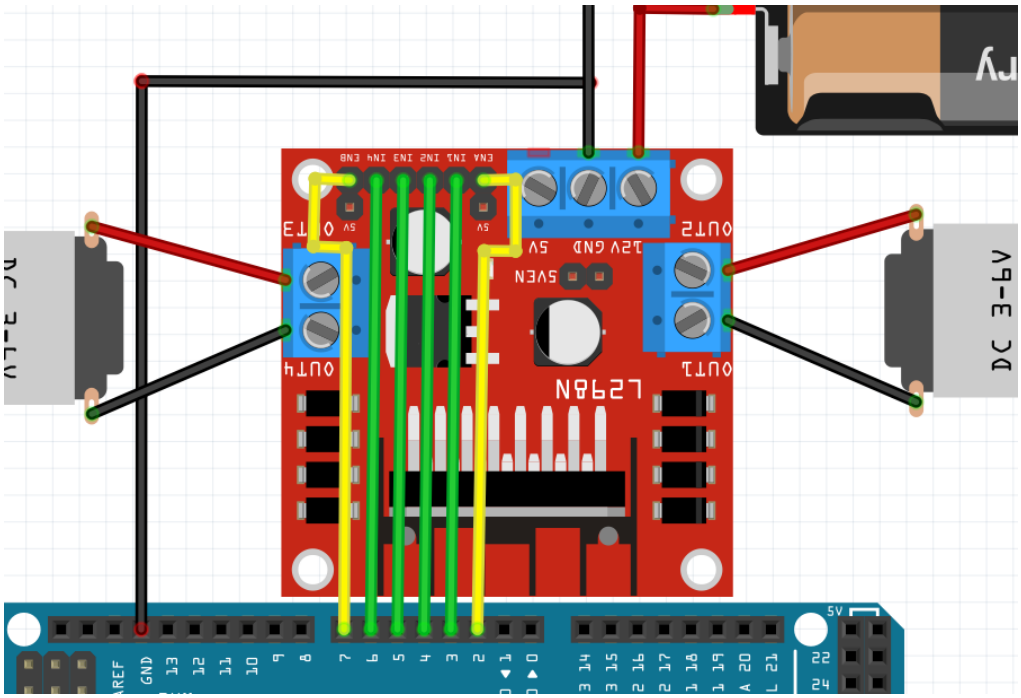
After you have soldered wires, place a **BIG GLOB OF HOT GLUE** on your solder joint and the wire. The goal is to add **strain relief** to the super thin input tabs to your motor. It would be very sad if on the day of competition these little tabs broke off and the motors stopped working :(



Now let's connect the motorshield and run some motors! You don't need to have this attached to the chassis yet.

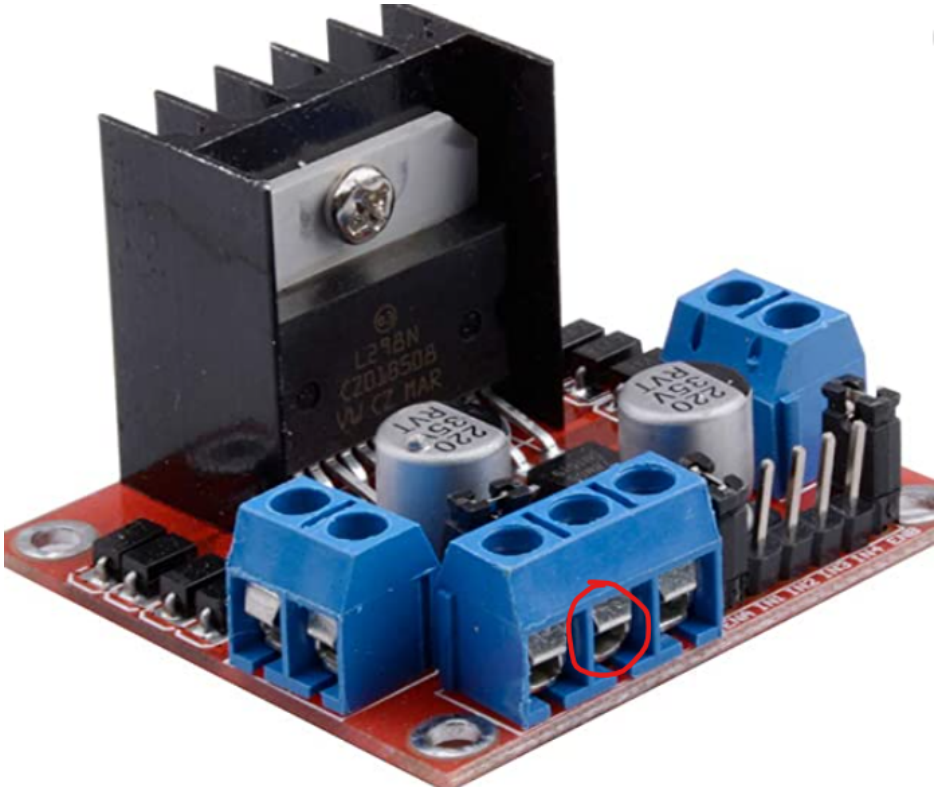


fritzing



Looking at the diagram above, connect your motors to OUT1, OUT2, OUT3, and OUT4 on your motor driver. Ensure both power sources are available to connect, one for the Arduino Mega and one for the motor shield. **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.

It is very important that all the **grounds are connected to each other**!!! The ground from your double A batteries and motor driver must be somehow connected to the ground of the Arduino as well. I recommend twisting the ground wires from the double A battery and the arduino together and then putting those inside the motor driver's ground terminal.



Code it! You can use the Motor_driver_code or build your own. Make the robot move forward for 3 seconds and backward for 3 seconds. After reading through the code, you can 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 the motor driver code 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 Motor***

Shield from M1 and M2 by inserting the black wire where the red wire is and vice versa.

Wiring summary:

- Switch motor direction by swapping black and red wires
- You must CONNECT ALL GROUNDS TOGETHER!!
- The 5V that come off of the motor driver has a max current of 36mA DO NOT POWER ANYTHING OFF OF THE MOTOR DRIVER's 5V OUTPUT
- When I used pin 13 as one of my control pins, it did not do what was expected so I would suggest not using that pin.

Challenge #1D: CAD the Chassis

Now we want to start working on our chassis. Design 3D printed **improvements** to your robot in Onshape.

After you click on the link, please hit "Make a copy" near the top so that edits can be made.

Make a copy to edit

It may be based on the 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!

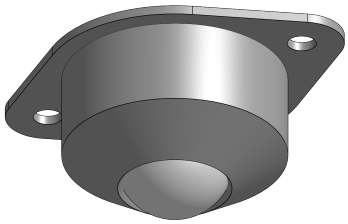
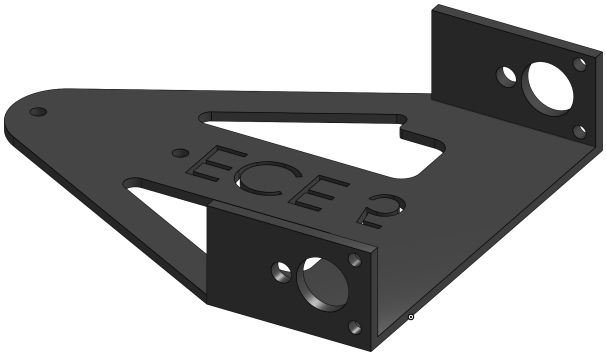


Image of caster wheel



Preprovided part in Onshape

Examples of possible modifications:

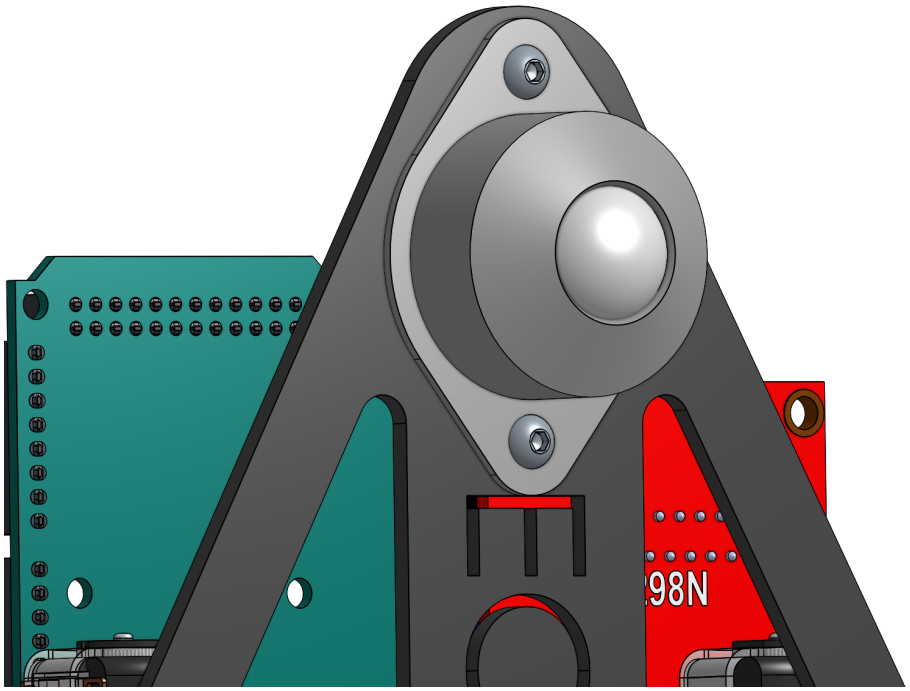
- A 3D print to hold a breadboard
- Mounting for the Arduino Mega
- A part to hold the photoresistors in front of the robot
- Light shield

Modifications **do not** need to be overly complicated nor do they require utilization of the components in Onshape. The components are useful if you are designing with them in mind however.

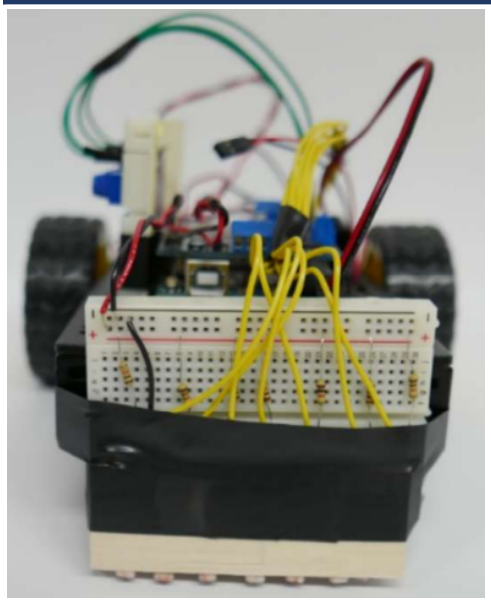
Chassis Assembly: Attaching Things

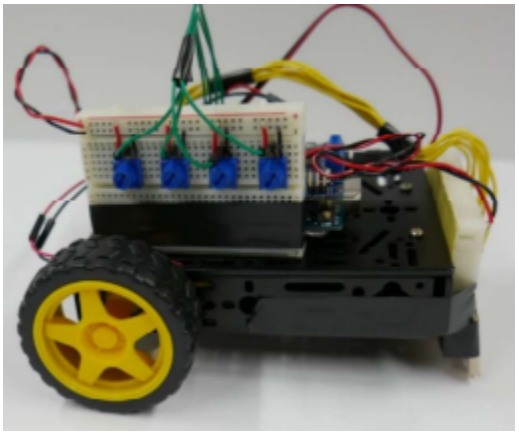
In order to attach your yellow motor to the chassis, use tape.

To attach the castor wheel on the underside of the robot, use the M3x6 Phillip Head screws given to you. Align everything together and then thread the screws into the plastic. This **will** take some effort but be sure to not over tighten!



Challenge #2: System Integration





There are many ways to attach the line sensors.

- Remember to consider the height of those photoresistors from the ground for good sensitivity measuring white vs. black surfaces.
- It may also be good to use a popsicle stick 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.
- A light shield is commonly used to ensure the photoresistors do not have outside interference from its environment.

Run each of the programs from Challenges 1A - 1C 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?

It cannot be stressed how important it is to isolate any problems **before** running the entire line following robot program.

Challenge #3A: 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 robot when you run the program. 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.

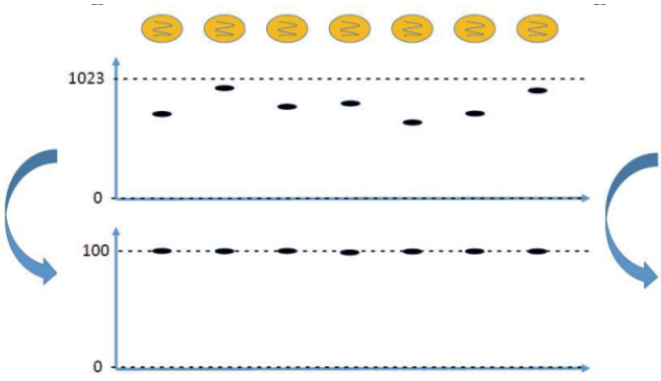


Figure 1: Mapped reading for a black line

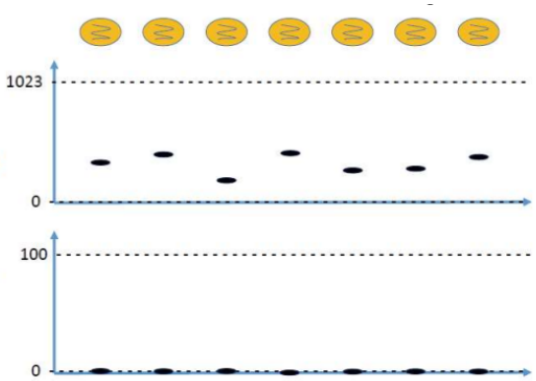


Figure 2: Mapped reading for a white line

Before implementing the new code, you may want to add an **indicator LED** to make it clearer what calibration step the robot is on. If you have LEDs to illuminate your photoresistors you can simply turn those into indicator LEDs as well for some cool underglow. Please put any indicator LEDs to this array in the code:

```
64
65  int led_Pins[] = {41,43,45,47,49,51,53};  // LEDs to indicate what part of calibration you're on and to illuminate the photoresistors
66
```

Open the Line Follower Code.ino file.

Read through the code table of contents and comments for the Line Follower Robot Code. Test this code by uploading to the Arduino and following these calibration steps:

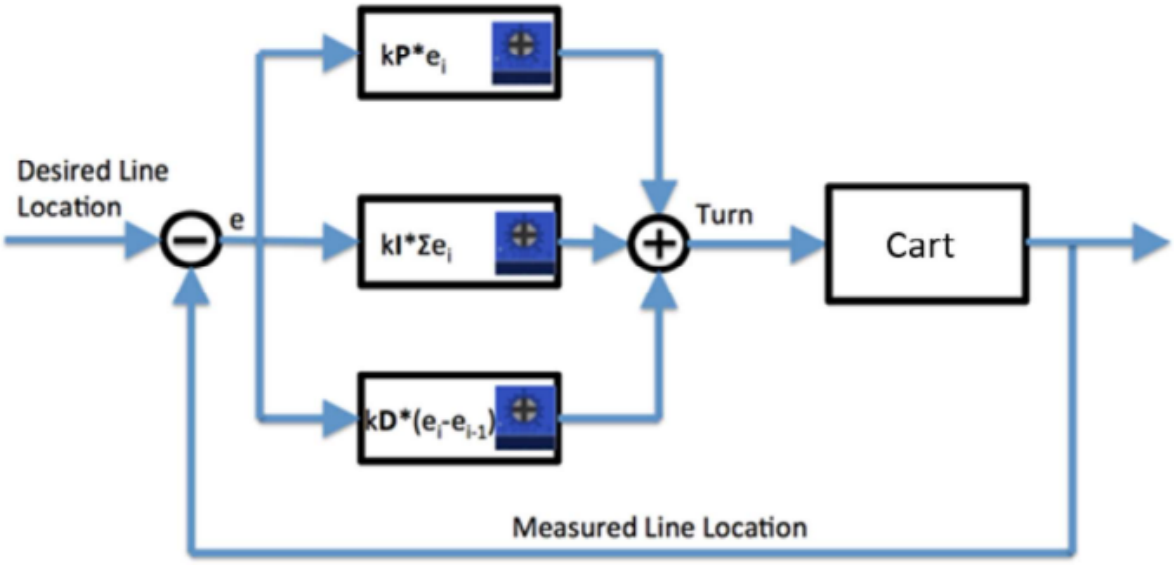
1) Before you turn the robot on, place the robot's sensors over a white surface
2) Turn the robot on. The indicator light will blink on and then off once.
3) Once the indicator lights are off, the robot has calibrated the white color
4) Move the robot to the black surface

5) The LED lights will give you three warning blinks, on the last one it will calibrate for the black color. Whilst initializing the indicator lights will turn off.
6) When the light turns back on and remains solid, the robot is ready to go!

You can also calibrate the robot with the Serial monitor open and the robot will describe to you what it is doing.

Challenge #3B: PID Control

I recommend you watch my favorite video on PID control as a refresher.



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 car'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)

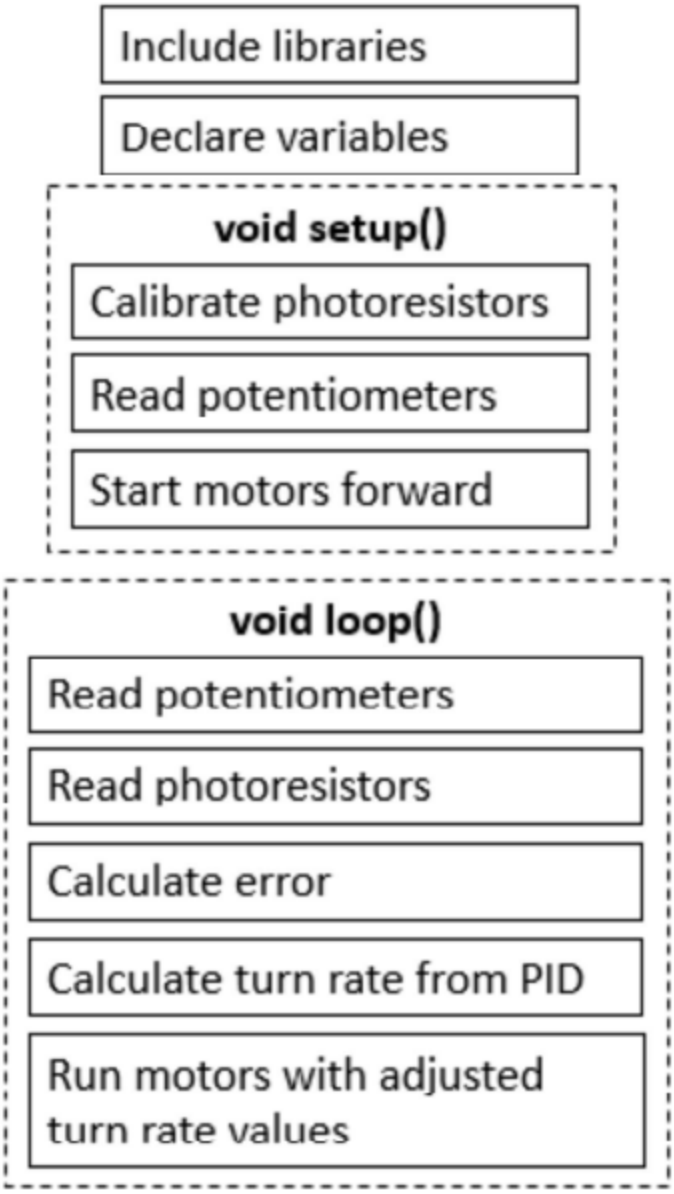
Odds are high that, unless corrected for, the rate at which your error changes will remain the same from time step to time step. By recognizing this worsening (or improvement) of your error, you will be able to increase (or decrease) the amount of additional right or left motor speed necessary to counter 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 #4: Competition

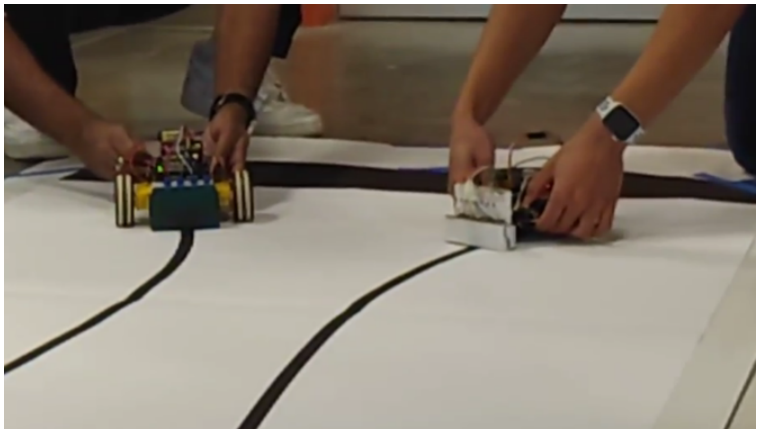
1) 3 Different Courses

Each course will have different criteria for scoring including (1) distance of line followed, (2) accuracy of sensors, and (3) speed of robot. With the ability to control your speed and PID knobs, your robot will compete against your classmate's robots.

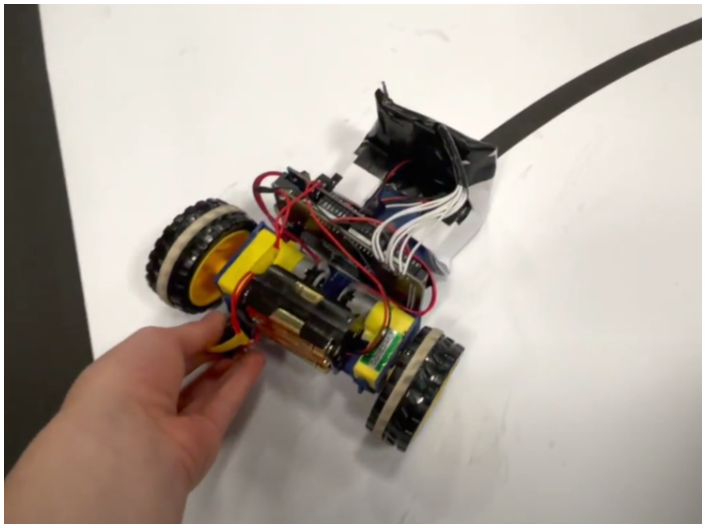
2) Innovation/Creativity challenge

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.

Inspiration

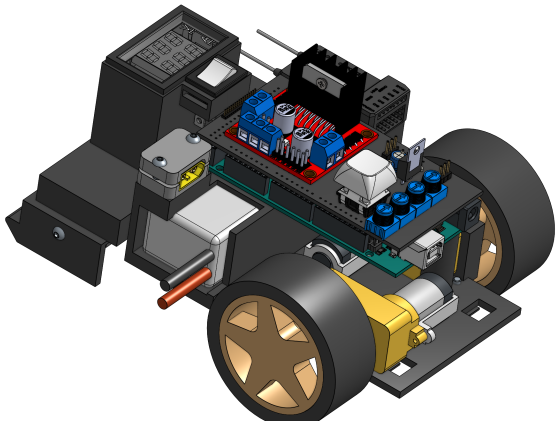


Drag Race 1



Adin's robot recovering from falling off of the line.

Ming's Robot



Ming's Robot for the 2022 model year