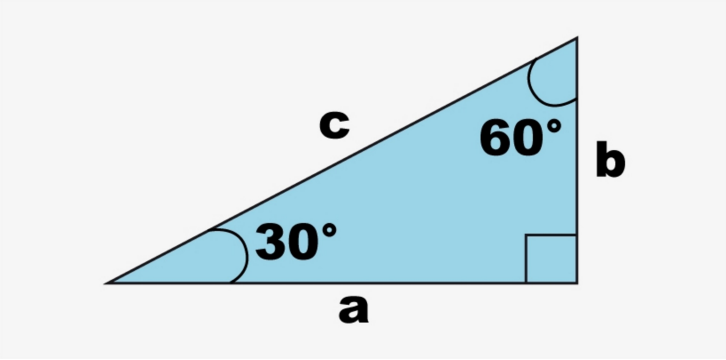
Complete Solar Tracker Instructions

Solar Tracking System Module for Solar Panel Optimization
Welcome! Congratulations on joining the ECEN 191 class and choosing this module in order to get credit. In this module you will be doing an exploration into solar panel efficiency. You will be designing a device that helps a solar panel attain maximum efficiency by aligning the solar panel with the sun using an Arduino, servos, and sensors. You will learn about how solar panels generate electricity, how servo motors work and how to control them, you will build and learn how to test a simple Arduino circuit, and you will submit a PCB design to the shop so that your project is nicely printed on a circuit board to show off to your family. Let's get started!

Lab 1: How Solar Panels Work
In this lab you will familiarize yourself with the inner and outer workings of solar panels, and understand why tracking the sun would be desirable.
To get a background on solar panels, read "What is Solar Energy" from the following article:
https://us.sunpower.com/solar-array-definition#:~:text=When%20photons%20hit%20a%20solar,electrons%20loose%20from%20their%20atoms.&text=When%20electrons%20flow%20through%20such,to%20form%20a%20solar%20array

Then read "How do Solar Panels Work" from the same article.

Now that you have a better understanding of not just how an individual panel works but also what is required in order to create a system of panels that can feed energy into your household electronics or the grid in general, there's just a few other things I want you to know and think about in relation to this project. Despite being so cool, there are some prohibitive reasons why very few roofs have solar panel arrays. Solar panels are expensive, they take up a lot of space, your roof may not be optimally located with a convenient south-facing roof, solar panels themselves can be extremely inefficient (this brochure (1) cites 20% efficiency as an extremely high efficiency, the rest of the solar energy just goes into heat or light reflection), and many solar panels are produced in places like China where much of the carbon emissions savings by installing a solar panel may be offset by emissions due to the way in which the product is manufactured (2).
Suppose we have bought a solar panel or solar panel system already, and we want to make it as efficient as possible. One of the factors in efficiency is the angle at which the sun strikes a solar panel. This article gives a great summary of how and why that works, which you can read, it's not very long:
https://www.altenergymag.com/article/2005/08/solar-energy-potential-at-different-latitudes/120/#:~:text=We%20get%20almost%20as%20much,at%20the%20Equator%3B%2098%25.
How much does the inclination of a solar panel affect the efficiency? Estimates vary.
For instance, using a triangle like this one there the sun is at the cb vertex and the solar panel is perpendicular to line a at the ac vertex, the length of c represents the strength of the radiation of the sun at a 30o angle to the panel and line a represents the actual amount of radiation that the panel can absorb due to the angle. If you increase the 30o angle then less sunlight will strike the panel as explained in that article.



With a 30o angle between the face of the panel and the sun's rays, you are losing 13.4% of energy that you could be capturing. Since the sun rises in the east and sets in the west, for part of the day the angle is much greater than 30o and you are going to lose a lot of energy. What if we could keep the panel aligned with the sun all day long?
This chart from wikipedia (3) gives efficiencies at different time of day. The "hours" column indicates the time away from noon that the sun will be at that angle from the panel. E.g. 1 hour before or after noon the sun will be about 15o away from the optimum angle.

i	Lost	*i*	hours[10]	Lost

0°	0%	15°	1	3.4%
1°	0.015%	30°	2	13.4%
3°	0.14%	45°	3	30%
8°	1%	60°	4	>50%[11]
23.4°[12]	8.3%	75°	5	>75%[11]

Looking at the chart, we see that if the solar panel is within 10-15o then we lose virtually no power, but doubling that angle to 30o radically increases the losses at a non-linear rate.
That wraps up this Lab! You might try getting an estimate of how much energy is being lost on a typical day by a traditional solar panel that doesn't move by adding up the losses multiplied by the amount of time that the percentage is lost for and dividing that number by the total number of hours to get a percentage of time/energy lost. Over a ten hour period centered on noon you lose like around 33% of the energy! That's like pointing the solar panel at the sun for 2/3 of the day and then just disconnecting it for 1/3 and throwing away those gains.
There were some things we didn't take into consideration, like the fact that the sun isn't as intense during the earlier hours of the day, but you get the general principle.

1. https://us.sunpower.com/sites/default/files/media-library/white-papers/wp-understanding-different-types-solar-and-mounting-solutions.pdf
2. https://spectrum.ieee.org/green-tech/solar/solar-energy-isnt-always-as-green-as-you-think
3. https://en.wikipedia.org/wiki/Solar_tracker#cite_note-10

Lab 2: Gathering materials to build the solar tracker

You will need the following materials, which you can get from the ECEN shop:
1 6v solar panel
2 180o servos
4 photoresistors
4 10k resistors
Arduino Uno
Breadboard
20 male to male breadboard wires
10 Male to female wire connectors
Outlet to usb adapter
Usb mini cord
small screws and bolts to hold the 3d printed tracker together, or some other mechanism

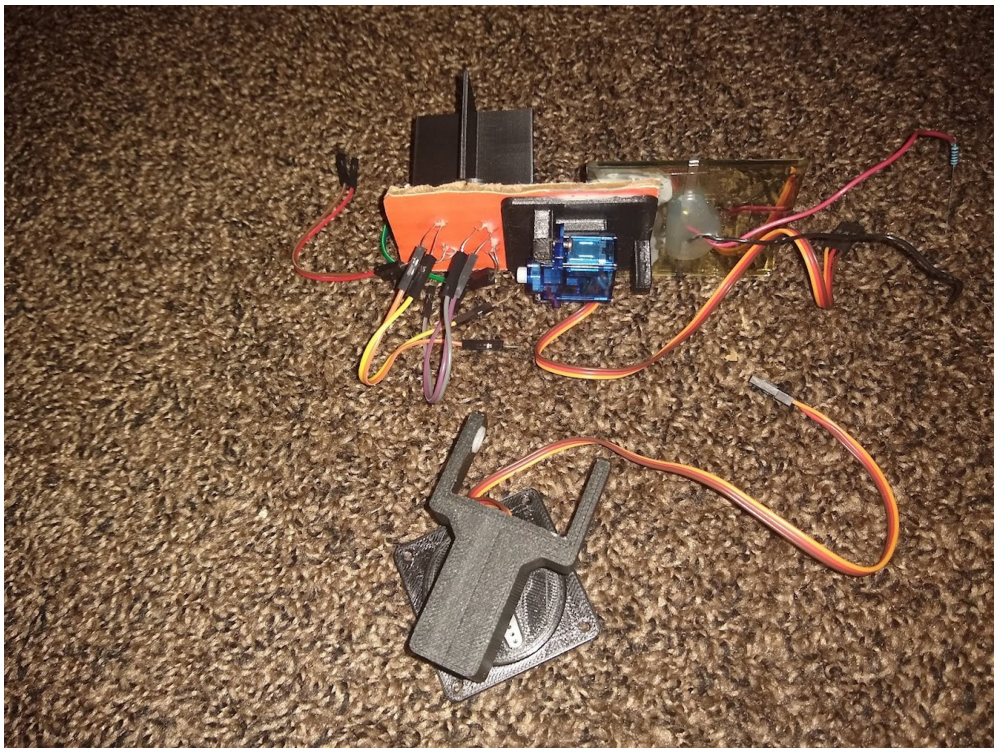
You will also need to order the SolarTracker.stl and stl files in the mini_pan_tilt_servo_g9 folder on the SolarTracker github repo [https://github.com/BYU-ELC/191_SolarTracker/tree/main/Mini_Pan_Tilt_-_Servo_G9] to get 3d printed from the ECEN shop by following the instructions on this webpage: [https://ece.byu.edu/project-requests]

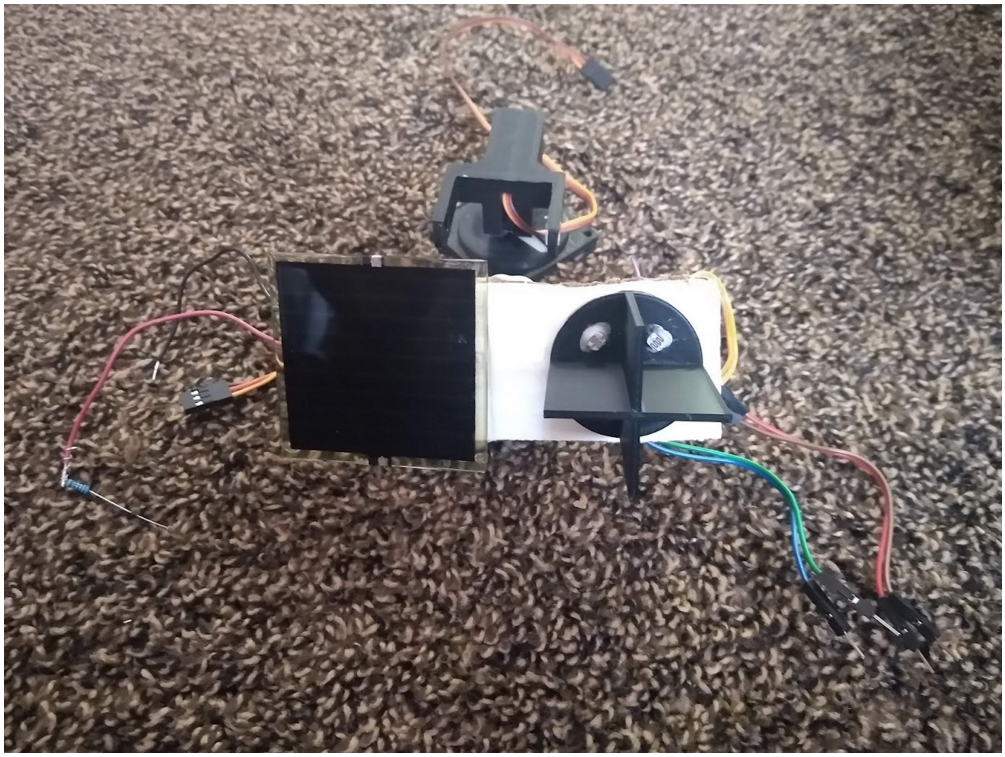
You'll also need a flat sheet of cardboard or plastic or something more stable about 6"x6" to mount the tracking system on, as well as a smaller rectangular piece of carboard about 3"x6" that you will glue to your mount and stick the solar panel to.

You will either need some small screws and/or a strong glue (I actually just used Elmer's glue but I wouldn't recommend that) to stick some of the 3d printed pieces together.

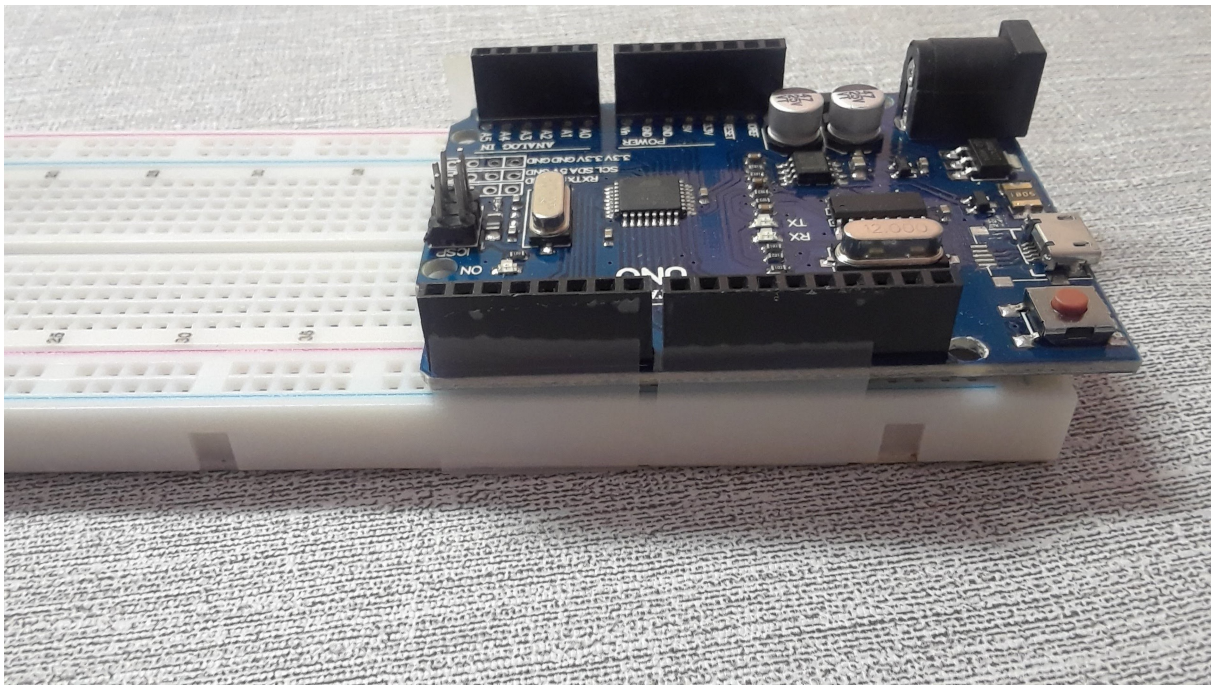
General assembly:
Power source: you will need exposed wires you can connect to your breadboard to power everything. Take the USB mini cable and cut off the mini adapter on the end. Strip the wires (if you want a demonstration and help, the shop or a knowledgeable friend can help) leaving an inch or two exposed, to be inserted into your breadboard. You might consider using two of the male to female wire connectors. Just make sure as little of the wire is exposed as possible so you don't accidentally shock yourself.

This video shows a similar setup and may be useful in interpreting instructions: DIY Solar Tracker || How much solar energy can it save?
Solar Tracker Module: take the pieces from the mini pan tilt servo 3d print and assemble them, using the pictures as reference and using screws or a strong glue to hold them in place. Secure the servo's in the appropriate places and voila you have a working servo module that can be programmed to rotate as you will. Take the 3d printed solartracker (circle with the cross shape) and push photoresistors through the holes, gluing them in place. Poke holes in your smaller piece of cardboard and glue the solartracker piece to the cardboard. Glue that to the solar tracker module horizontally and glue the solar panel to that as in these photo:

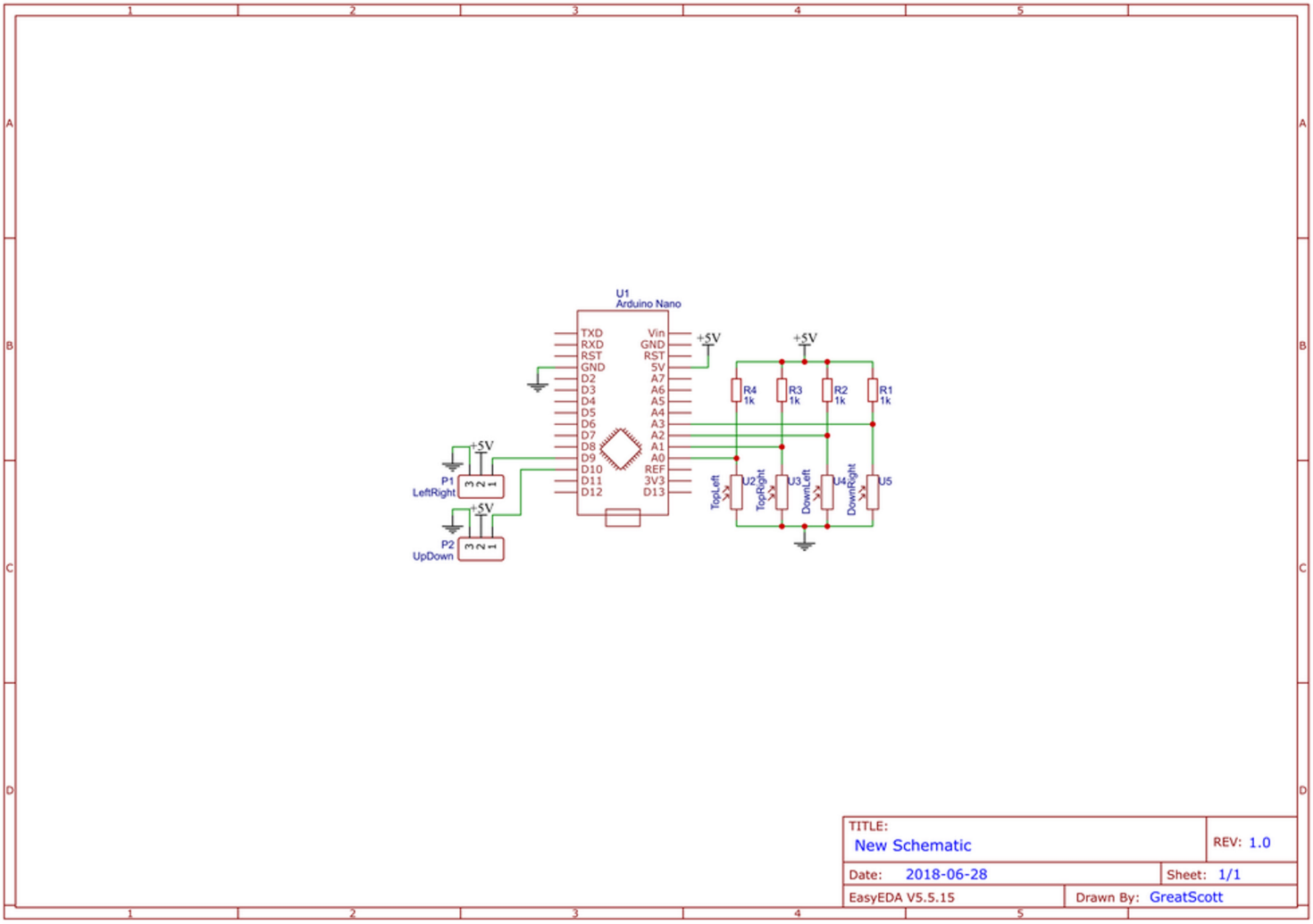




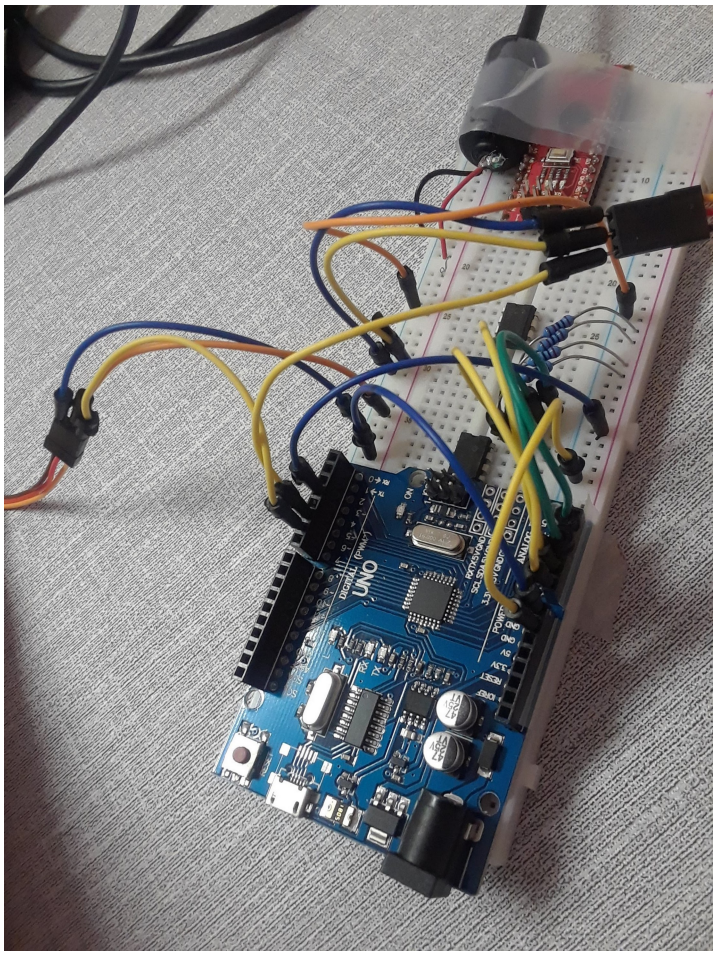
Breadboard and Arduino setup:
Secure your arduino uno to the breadboard. There are better ways, but I used tape along the sides of the plastic headers on the Uno and ran the tape underneath the breadboard like this:



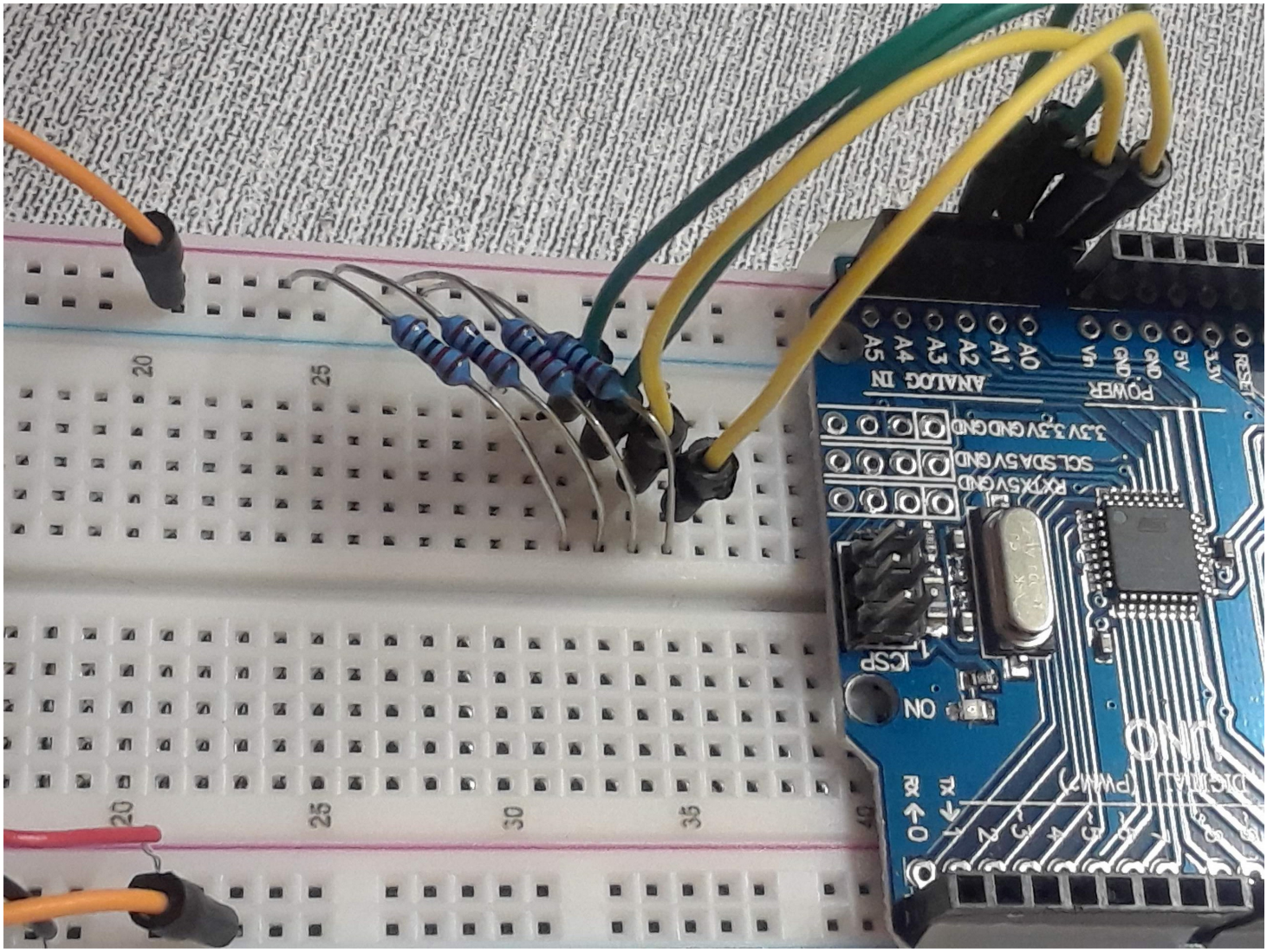
If you have never used a breadboard or played with circuits before, there are some things you should know. Breadboards are extremely useful for prototyping circuits. If you had to print a new circuit board every time you made a change to your circuit your life as an EE/CE would be harder. Thankfully, breadboards make it easy to rebuild your circuit by changing connections between wires wherever needed, allowing quick and easy prototyping for low frequency and low current circuits.
Got to this website: https://www.instructables.com/How-to-use-a-breadboard/
Scroll down to step 2 to see an image of where things are connected. As you can see, every long column is connected internally up and down. They are on the sides of the breadboard and two (one on each side) are labeled "-" and two are labeled "+". The - indicates the "common ground" of your circuit, kind of like the "end" of your circuit, also the part of the circuit that represents 0 voltage in reference to other parts of the circuit, and the + indicates the power side of your circuit, where you input a high voltage. The horizontal rows consist of 5 pins all connected, and you can connect different rows of pins in order to get your circuit built. The other pictures may help illuminate how the physical connections work beneath the plastic. This breadboard is your friend. If you are interested, a simple circuit can be seen in step 4 of that Instructable site. Trace the circuit, starting from power and looping through to ground.
With our breadboard circuit, place one of your jumper cables so both ground rails on the side are connected. Choose any two pins, one on each ground, and simply insert one end of the cable into each. Now that our grounds are connected, if you need to connect something to ground you can plug it into either side as desired. We won't connect the power rails together, as they will actually be used to power different things that need different voltages.
Here is a schematic of the circuit we will be creating:



The Arduino (in our case it's the uno, not the nano) is in the center, with the servos on the left and a sensing circuit on the right. The sensing circuit has photoresistors and resistors, with a wire in between them going to an input on the Arduino so we can read sensor values.
We'll come back to how it functions later, but for now let's build it. Your end product will look something like this:



Designate one of your power rails as the main power rail, for the servos and Arduino, and designate the other for the resistors/resistor network. Take the four 10k Ohm resistors and plug them into the resistor power rail and plug the other end of the resistors into any horizontal strips, with one per strip. Connect each of these strips to the A0-A3 pins on the Arduino, one strip connected per Arduino input pin. These are input pins on the Arduino for receiving data and are located next to the words "ANALOG IN" on the Arduino.



Take 4 male to female wires and plug one male end into each of the strips with a resistor on it. Plug one end of each photoresistor wire (it doesn't matter which end of the photoresistor) into the female end of the connector. Take 4 more male to female connectors, attaching the female end to the remaining photoresistor wires and plugging the male end into the ground rail of the breadboard.
Take a male to male wire and connect the ground of the Arduino Uno to the ground of the breadboard. This pin is labeled GND by the word POWER. There are two pins labeled GND, and it does not matter which you use. Take another wire and connect the Vin pin of the Arduino Uno to the power rail on the breadboard you designated for powering the Arduino and Servos. Connect the D3 pin on the Arduino Uno (by "DIGITAL (PWM~)") with the power rail designated for the resistors.
Plug three male to male wires into the pins on your two servos. Connect the red servo wires to the power rail on the breadboard designated for the Servos/Arduino, the orange wires to D5 and D6 on the Arduino, and the black wires to ground on the breadboard.

You now have a circuit capable of fulfilling our goal!
Keep the usb cable that you stripped unconnected from the wall for now.

Download the Arduino software you'll need to program your arduino from this website:
https://www.arduino.cc/en/software
After it downloads, open it. Plug your Arduino into a computer port using a USB micro cable. In the IDE (the Arduino software), select "tools", click "board", and choose "Arduino Uno" so that the IDE can communicate properly with your specific Arduino. Also in "tools", click "port" and there should be one port available, a COM port where your Arduino is plugged in. Select it. If you ever use a different USB port you will need to correct which one the IDE communicates with here. If there are multiple ports listed you may need to figure out which one is connected to the Arduino. In Windows, go to the search bar at the bottom of your screen and search "device manager". Go to the list of COM ports. Note the ports and see which one appears/disappears when you plug in/unplug the Arduino. The list of devices should refresh each time you plug/unplug the Arduino, and you'll see which one is the Arduino. It probably has the name "CH340".
In the Arduino IDE you will write code that tells the Arduino what to do in a later lab.

Lab 3: Ohm's Law, Photoresistors and Voltage Dividers

Let's dive into the sensing part of our Solar Tracker that will tell the Arduino where the source of light is. Photoresistors are the key component in this. Unlike solar cells, photoresistors don't actually sense light directly by turning it into electrical current like solar cells do. They function like traditional resistors, resisting the flow of current, except that the amount that a photoresistor restricts electrical flow is inversely related to the amount of light hitting the resistor. The more light hits the photoresistor the less it will restrict current.

The Arduino doesn't measure current directly, but it's input pins can measure voltage, and we can relate voltage and current in our resistor circuit by Ohm's Law, which states that V = I*R (Voltage = Current*Resistance), or the difference in voltage between two parts in a circuit is equal to the current flowing through that part in the circuit times the resistance of that part of the circuit. We won't get any useful information by putting the sensing wire in front of the 10K Ohm resistor because we already know that the voltage drop across the resistor and photoresistor combined is 5V (the Arduino out pin supplies 5V, and ground is always 0, so 5 - 0 = 5V) but if we put the sensing wire between the resistors like we did in Lab 2 we can measure how much the voltage drops from that part of the circuit to ground, over the photoresistors.

This will change based on the photoresistor values which is based on the amount of light hitting the photoresistors, and now we have a relationship between the values coming into the Arduino and the amount of light on the photoresistors. This technique is called a "voltage divider". If the 10K Ohm resistor and the photoresistor are equal in value then the voltage measured in the middle by the Arduino will be half of the total voltage. If the photoresistor has less resistance because there's a lot of light on it, then Ohm's Law tells us that very little voltage will drop across it, and we will get a low input value. Similarly, if there is no light on the photoresistor the value of the resistance will be very high, much higher than the 10K Ohm resistor, and almost all of the 5 Volts will drop across it.

The Arduino will measure this 5 Volts as a big input value. In ECEN 240 you will go further in depth into this but an equation that tells the exact voltage drop across the second of two resistors is given by V*R2/(R1+R2), further confirming our conclusions about how light on the photoresistor and our Arduino's input value are related.

We are now ready to program our Arduino to be able to read photoresistor values! If you are familiar with coding then you probably don't need much of the explanations in this or the other programs that are used in this module. If you've never coded before I tried to explain things so that it's clear what the code is doing.

Open the Read_Resistors_Solar.ino file from the Github repository in the IDE. This is based off a basic program that comes with the IDE, and you can find more programs like it in "File" -> "Examples". If you have never coded anything, fear not, we did it all for you. Let's walk through what the code is telling the Arduino to do.

"int sensor1 = A0;" creates a variable named sensor1 and assigns it to the A0 pin on the arduino, Where values will be read from.

In "void setup() {":

"Serial.begin(9600)" allows for the Arduino to send information back to the IDE and feed it our resistor values so that the IDE will display them for us.

"pinMode(3, OUTPUT)" takes the out pin 3 and assigns it to be an output pin so we can feed 5 Volts across the resistors through that pin.

In Loop:

The loop routine, as the name suggests, runs once sequentially and then loops back to the beginning of "loop()" and runs again. Forever, or until you stop the Arduino.

"digitalWrite(3,High)" takes our digital output pin, 3, and starts feeding 5 Volts to it. Now, with "int sensorValue1 = analogRead(sensor1)" we can take the voltage values coming into sensor1 and capture them with the sensorValue1 variable.

The "Serial.print()" commands arrange our sensor values on the output monitor where we'll read them.

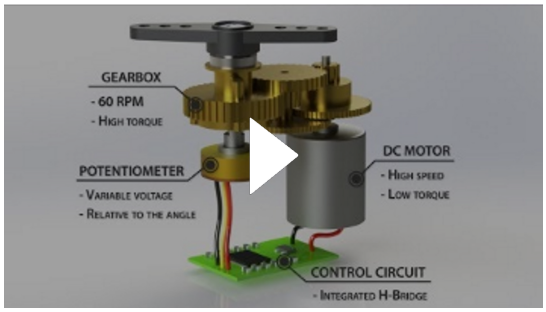
"delay(50)" tells the program to wait 50 milliseconds before restarting the loop.

Go to the top of the IDE and click "upload". It looks like a right arrow. If nothing in the IDE changed colors, that's usually good. Go to "tools" and click "serial plotter" or "serial monitor", depending on if you want to see the value printed out line by line or if you want to see a live plot of the values. Check out both and notice how the values change as you shine your phone's flashlight on individual photoresistors. Do the values go up or down when you shine light on them? In a way, you could think of the photoresistors and measuring the amount of darkness on the photoresistor, since it increases inversely with the amount of light. The number also never goes beyond 1024. This is because the Arduino takes the reading on the analog in port and then scales it between 0 and 1024.

From here we try and get the values to equal each other, or as close as possible. This is where the module will be pointing almost right at the sun, because the cross will obstruct one or more of the photoresistors if it's not point right and the sun and this will alert us by giving a large value on the monitor.

Lab 4: Servos
View this video to familiarize yourself with how servos work.
There are a lot of different types of motors. Big ones that take 240 V like your clothes dryer, small ones like those hobby motors that also appear inside servos, DC motors, AC motors, etc. The motors we are using are small, precise, and have a lot of torque considering their small size. They also don't turn in a full circle, they actually just turn through 180 degrees, which is sufficient for what we are trying to do, given the sun doesn't pass through more than 180o of sky. You can get "continuous" servos that can turn around as much as you want. The nice thing about our servos is that we can tell the servo to turn to a specific degree between 0 and 180 and don't have to mess with timing/on values to try and get it to the right place. It will just go wherever we tell it. These servos are rated for use around 5V and 1Amp of current. That's about 5W of power if we wanted to continuously move a load with the servo. That's also a lot more current than the Arduino can supply (its pins supply about 20ma of current, or 1/25 of the required current). You can try just hooking it up straight to the Arduino, but it will be very weak and may not move at all. That's why we hooked up the servos to the power rail that is supplied by the USB power outlet adapter, which is rated at 2A of current and 5V. Perfect for our 2 servos and it should be able to power the board at the same time. The wire from the Arduino to the servo has the function of telling the servo how far to turn, but it does not actually supply any power to help it rotate.

Now, take your stripped wires from the USB mini cord and insert them into the Arduino/Servo power rail and a ground rail (***making sure that it is not plugged into the USB power outlet adapter while you are doing this, but plug it in afterwards. Your servo motors may respond a little bit***). Avoid touching bare wires after you plug it in, because if you accidentally complete a circuit now you will have a fairly large current coursing through you. **VERY BAD**. Be careful. Make sure the Arduino is plugged into the computer still, and open the Servo_Control.ino program using your Arduino IDE. (You may wonder at this point, because the power rail for the Arduino/USB is now live AND you are powering the Arduino from your USB port. Where does the Arduino get it's power? Is this bad for the Arduino? After doing some research the different sites I looked at said that this should be fine, and the Arduino probably gets it's power from the power rail only, not the USB. From there it only gets programming data. Either way, connecting the power sources "in parallel" like this isn't a problem.)
With the Servo_Control program we are going to do some quick exercises to learn about how to control the Servo and adjust the Servo or the program so that it works correctly.
Watch this video, at least up to 6:30, though the whole thing is interesting if you want to control a lot of servos: How Servo Motors Work & How To Control Servos using Arduino



At 2:38 in the video we are shown the timing and frequency of the pulses we will need to send from the Arduino to the servo in order to get it to a specific angle. Remember, the outlet powers the servo while the Arduino just uses pulses to communicate the angle at which the servo is to turn to. While this pulsing is something we could do, it would be much simpler to use the Arduino "servo.h" library which has already implemented functions that do what we want for us without doing the specific programming on our part. 4:39 shows this library, and it's what we'll use to drive our servo. There are a ton of Arduino libraries available online. They make coding much easier because often someone has already done whatever you are trying to do and the code is often easy to implement since all of the hard stuff is already written. From the servo.h library, you can use a ".write(angle)" command and using the instructions in the library the Arduino will automatically send the appropriate pulses to move the servo to the angle indicated.
Let's go to our Aruino IDE. Open the Servo_Control program. Let's walk through a simple program to control the servos that was made using the instructions and examples on the Arduino website where you can find documentation about the servo.h library.
"Servo VServo" creates an servo object that the Arduino can interact with.
"int VServoPin = 5" assigns our servo that controls up and down movement of the solar tracker to pin 5. This might be switched depending on how you connected it up, and you may need to swap the values for VServoPin and HServoPin at some point.
"int MaxVAngle = 130" is a variable that will keep the vertical servo from turning more than the value we put here, for now it's 130o.
"int MinVAngle = 80" does the same thing, but will be the minimum angle the servo turns through.
"int Vangle" creates a variable used to track the angle the servo should be turned to at any particular

moment.
The same thing repeats for the horizontal turning servo.

"int MotorWait = 50" gives a delay of 50 milliseconds in between each adjustment of the motor angle
In "void setup()" "VServo.attach(VServoPin)" runs some code in the Servo library so it treats the VServoPin (pin #5) as a servo.
In our loop, "delay(1000)" will give a 1 second delay at the start of each loop.
In the rest of our for loop, I wrote a program that will pan one or both of our motors back and forth along the range of angles given by MaxVAngle and MinVAngle or MaxHAngle and MinHAngle.

"while(VAngle <MaxVAngle)" says that we will execute the code between the brackets while the VAngle value is less than the MaxVAngle value. Once it's greater, the code moves on to the rest of the program. The code in the brackets writes the angle specified by VAngle to the vertical servo, waits for the amount of time specified by MotorWait, then increases the value of VAngle by 1 so that the servo will pan upwards just a little bit the next time we tell the Arduino to write the angle to the servo.
The while loop will start over and keep running until the initial condition (Vangle < MaxVAngle) is untrue. Thus, the Arduino will write a slightly higher angle value to the servo over and over until it reaches its max angle.

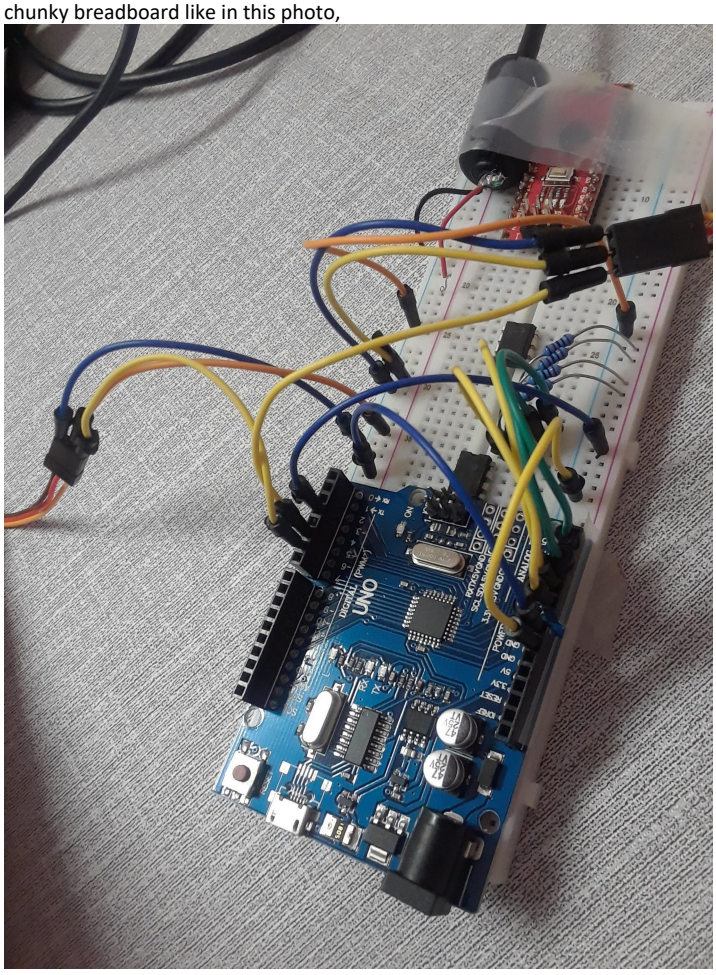
The same thing happens in the "while(Hangle < MaxHAngle) loop, but it will pan horizontally instead to the max horizontal angle.
The last two while loops have similar function as the first two while loops, but they decrease the angle horizontally and vertically.
Plug the outlet adapter and the USB cable for your Arduino/Servo power rail into the wall and upload the program to the arduino and observe how the motors pan the solar tracker. You will probably get some jerky movements of the motors at first, this is probably fine.
Try changing the max and min values and observing what happens. You may need to adjust your servos if they are oriented the wrong way, for example, if the vertical servo keeps trying to run the platform into the mount, adjust the minimum angle and probably the maximum angle to be bigger. You can also readjust the angle the platform is tilted at when it is first attached to the servo.
Find a comfortable range of values for your two servos and keep track of this for later. You want the vertical servo to be able to tilt the platform at any vertical angle the sun would appear at in the sky. The horizontal servo should be able to turn almost a full 180o because the sun rises in the east and sets in the west.

You can modify this program to do whatever you would like. You can pan both motors at once by uncommenting some of the lines of code, or change the code so it does something completely different.

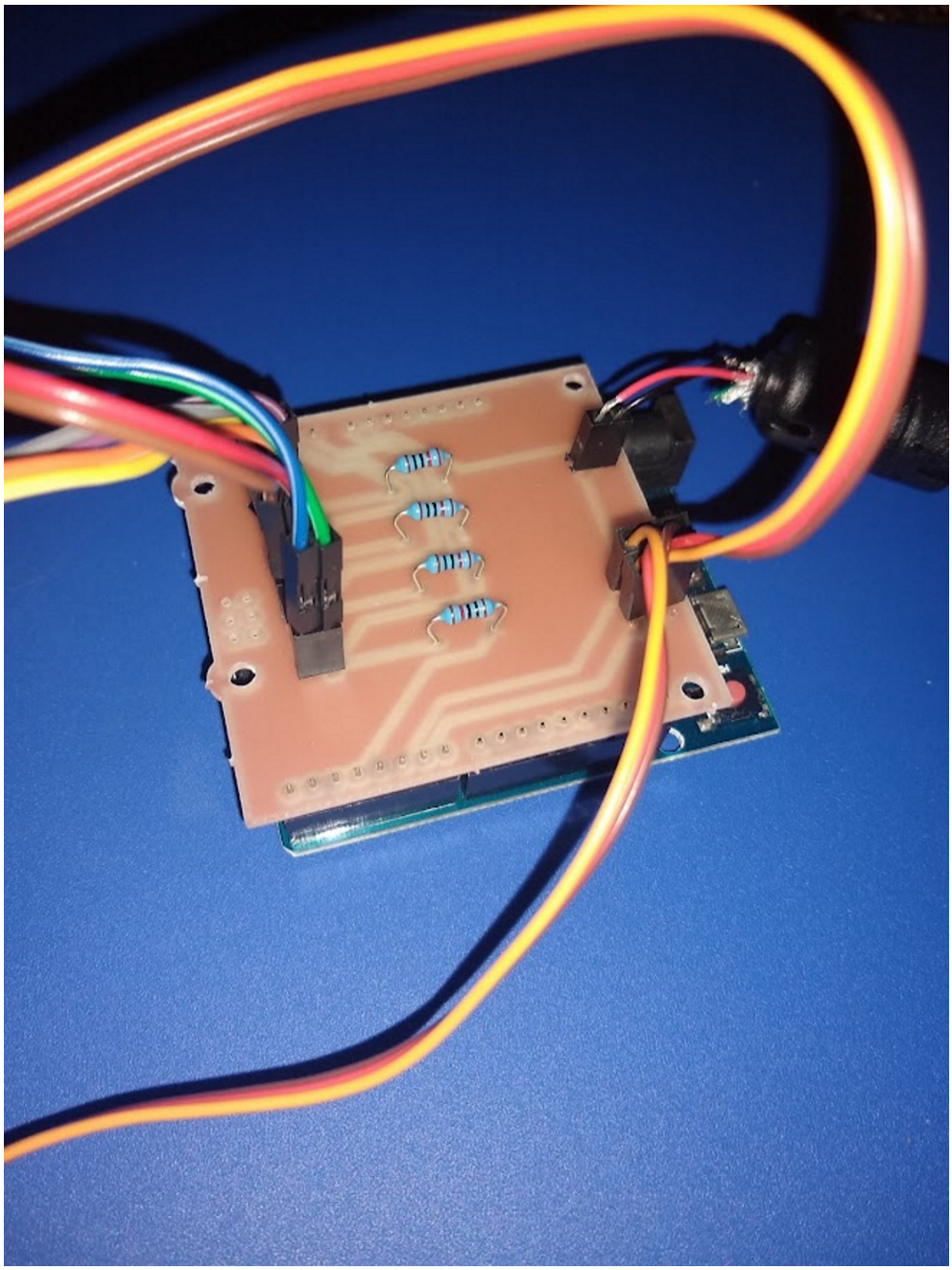
Lab 5: Functioning Solar Tracker
By the end of this lab you'll combine what you learned about sensing light using photoresistors and how to control servos to build a functioning solar tracker.
Open the Light_Tracker.ino file in the Arduino IDE.
Some of this code is the same as in the last lab. Change the max and min angles to your experimentally determined values from lab 4.
After the two initial blocks of code, we have some more variables created. "int threshold" is a number signifying the difference in light on the photoresistors is okay. If we have too small a number it will constantly readjust when it isn't actually going to produce any more energy, and if it is too big than the servos won't adjust the angle to optimize the horizonal and vertical angles enough. Numbers from 6-20 seem to be about right.
"int waittime" does the same thing as MotorWait in the last lab, specifying how many milliseconds to wait in between sampling light levels and choosing whether or not to move the solar tracker. Why might it make sense to put a really big value here so the delay is many seconds or even minutes rather than milliseconds?
"int upleft = A0" makes a variable that will store the amount of light viewed from the upper left of the solar tracker and stores it in A0. You may need to adjust which pin is used for which quadrant of the solar tracker. You can verify that values are being fed to the expected inputs on the arduino by shining your light over a specific photoresistor and see if the voltage drops.
By now even if you've never done any coding before this class you should be able to guess what most of the code is doing. In the loop we create some variables to store the values off the photoresistors (like in lab 3) and print those values out.
The block of code that starts with "//adjust horizontal equilibrium" increases the angle if ULVal (you could think about it like the amount of darkness on the upper left photoresistor) is greater than the value of URVal, making it turn away from darkness and towards greater sources of light. If the difference is less than the threshold, than it doesn't change the angle and simply moves on through the program. This isn't the only way to do it, but I also have the Arduino update the angle of the solar tracker after every time it checks any quadrants. There is also a safety net, if HAngle ever gets bigger than our max angle, it will simply set HAngle to our max angle. And same if HAngle is below our min angle.
"//adjust vertical equilibrium" does the same but for the vertical angle. After adjusting the vertical the

program is complete and will start over, checking the conditions to see if it should adjust the solar tracker's position at all.
Upload the program to your Arduino outside or in a room where there is only one clear source of light. Outside is best. Make sure the Arduino/Servo power rail is plugged into the wall and that the horizontal middle angle of the horizontal servo is pointing to where the sun is at noon-day. It should adjust itself and point the solar panel at the sun or greatest source of light. If you are inside, try using your phone's flashlight to get the tracker to follow your phone around. It might even turn towards far away ceiling lights in the room, though if windows are open reflected light from the sun could give unexpected results.
Change the "threshold" starting value to be something low, like 1 or 2 and re-upload your program to observe the results. Change it to 200 and observe if the precision of the solar tracker is affected. Decided a good threshold value, which may depend on how bright your light source is. Think about whether 10 milliseconds is a good wait time. Think about the best balance between updating enough that you are getting optimum power on the panel and waiting enough between updates that your Arduino isn't moving a ton, wasting power that way.
That's it for this lab!

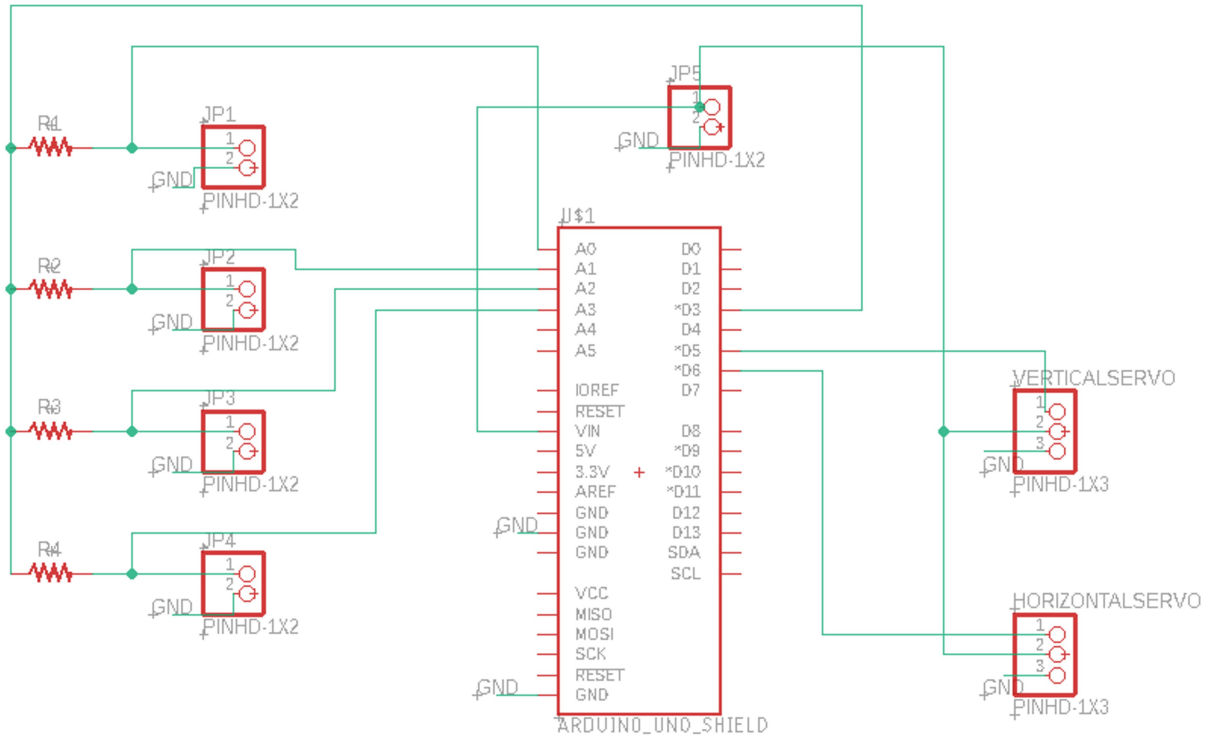
Lab 6: Solar Tracker PCB
Now that you've made a functioning solar tracker, let's make it more durable by taking the breadboard design and implementing it on a PCB (printed circuit board). Instead of all the wires coming out of a chunky breadboard like in this photo,



It will look much nicer, like this:



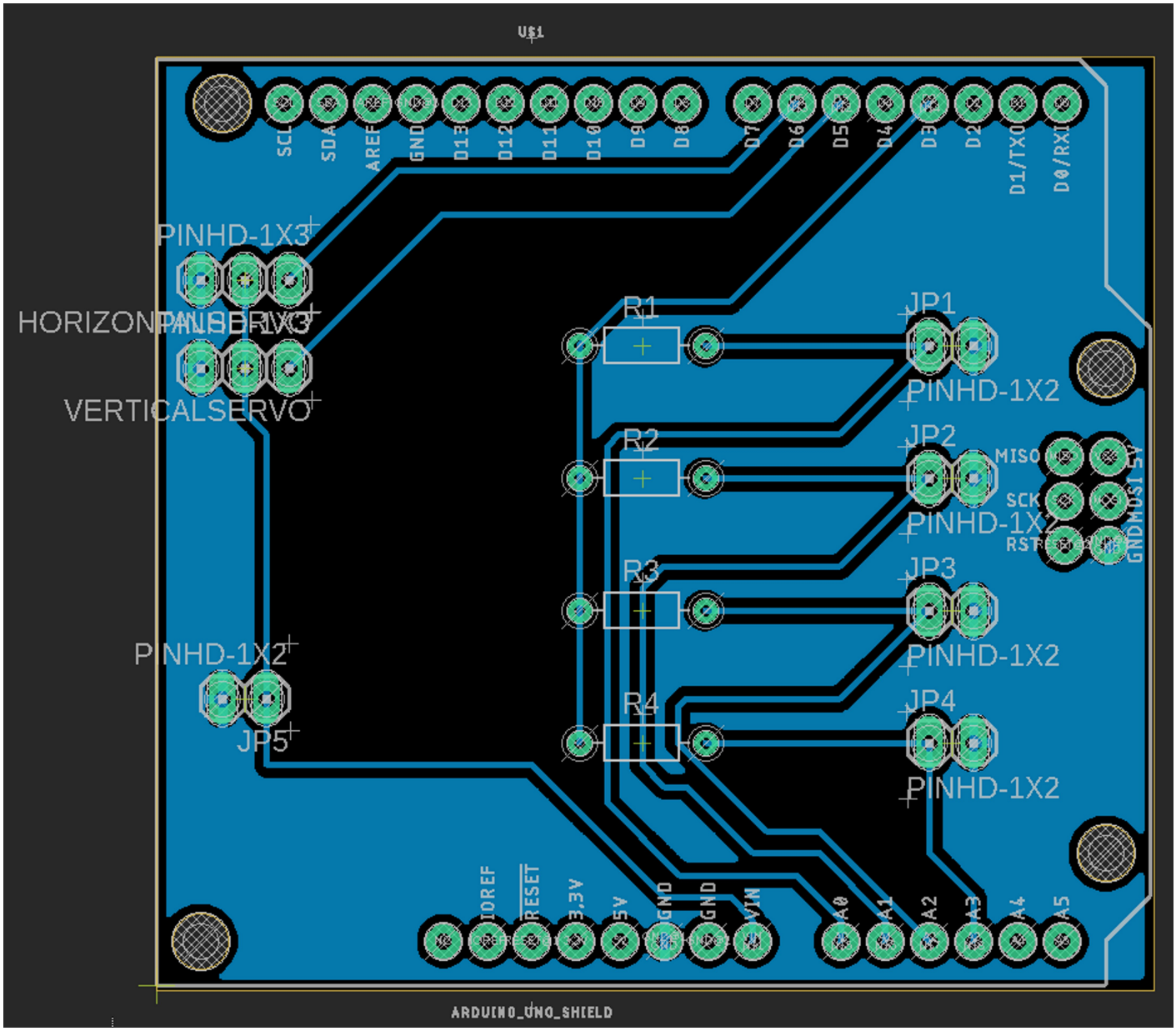
"Eagle" is a program that helps make circuit board files that can be printed out en masse. If you want to make your own circuits on eagle here is a tutorial that you can follow to add components, build the routed traces, check using design rules, and then submit gerber files to a PCB protoyping service like the ECEN shop (ELC) or Oshpark:
https://drive.google.com/file/d/1HpnMLaxNLDTfHNNuHJz5UkOK866xzLBX/view?usp=sharing
You also have free access at BYU to Eagle (and a ton of other programs) at:
https://byuapps.cloud.com/Citrix/StoreWeb/#/apps/all/static/All/
For this lab I just want to show what some of the process looks like.
You start by adding parts to a schematic and connecting them or declaring them to be connected.
Building the solar tracker, I made this:



You can see that it matches the connections on your breadboard. Any lines that have the same name are automatically connected without need for physically drawing lines between them, but I drew some of those lines anyway for visual effect.

Eagle automatically makes a board file where you make an actual 2D representation of what the finished PCB will look like. You can move parts around, build traces between parts, make holes in your board, and even place custom text on your board for more sophisticated PCB printing services.

Here's what our solar tracker board file looks like in eagle after everything is properly connected:



The blue lines and plane indicate metal traces that connect components, and the green circles indicate drilled holes where components are soldered to the board. Empty space indicates no metal will be deposited there (though the big empty spaces will be filled in when we get our PCB from the shop because of the way they manufacture them). You can also see the outline of the Arduino component and where it will connect to our PCB.

Download the "Solar_Tracker_v6.zip" file from the github repo. Go to the ECEN shop's project request page at https://ece.byu.edu/project-requests and click on "request form". Fill out the form, selecting "PCB prototyping". This is a single sided board, for a class, you probably only want one, and there are no special notes for the shop. Submit the form and wait for the confirmation email from them, then reply to the email, attaching the "Solar_Tracker_v6.zip" file. They will send you an email when your board is ready for pickup, which may take a few days.

…

When you go to the shop to pick up your PCB, you may need to buy the following components:

2 8 pin male to female headers
2 6 pin male to female headers
2 3 pin male to male headers
*5 2 pin male to female headers
*you could just solder the components directly into the board instead of using the header pins as a go between for these

Solder each of your headers into the place on the board according to the schematic, and solder your four 10k ohm resistors into place. Take the servo wires and insert them into the appropriate header pins on the PCB. If your board is oriented like the picture above from left to right the pins are GND-

POWER-SIGNAL. The vertical servo should plug into the lower headers and the horizontal into the upper header pins. Connect the photoresistors to the four header pins on the right, the order of wires in a header pin doesn't matter. Don't unplug the jumper cables from the photoresistors so that the photoresistors can reach from the solar tracker. From top to bottom, you need to plug the photoresistors into the headers in the following order: upper left, upper right, lower left, lower right. Insert the Arduino into place on the 6 and 8 header pins, and (while the power is unplugged from the wall) insert the red wire from your Arduino/Servo power cord into the right pin on the remaining blank header pins. Plug the power's ground into the leftmost pin.

Assuming the light tracking program was the last program to be downloaded onto the arduino, you should be able to plug your Arduino/Servo power into the wall adapter and the solar tracker will successfully follow the sun like it did in the last lab.