

Solar Powered RC Buggy With Sun Tracking and Esp32-now



by jonathanr4242

I've made a solar powered esp32-now rc robot. The solar module is mounted on a camera pan-tilt mechanism and tracks the sun by taking measurements from four photoresistors. This is really two projects in one. The first project is to build the esp32-now rc car (or any esp32 type buggy your like) and the second part is making the solar tracker for the sun. The solar module I'm using is quite small, so we need to track the sun to get any decent current from it. And, then we combine the two.

Supplies:

The following is a list of parts I used:

- Robot chassis. I'm using a 4-wheel robot chassis because it has two layers, but a 2-wheel one should work. You just need to have a top layer to mound the solar cells: See for example <https://www.aliexpress.com/item/32633842302.html>
- Solar module: See for example <https://www.aliexpress.com/item/1005003638212228.html>
- TB6612FNG x 2 dual mosfet h-bridge motor drivers. e.g. <https://www.aliexpress.com/item/1005002674647893.html>
- PCB or loads of jumper cables. I got my pcb from [pcbway](#) (I designed the pcb using fritzing, instructions below). Alternatively, you can wire with jumper cables.
- 18650 batteries x 2. For example <https://core-electronics.com.au/polymer-lithium-ion-battery-18650-cell-2600mah.html>
- Battery holder x 2 for 18650 battery.
- TP4056 x 2. See for example <https://www.aliexpress.com/item/1005001562314459.html>
- MT3608 dc-dc boost converter. E.g. <https://www.aliexpress.com/item/1005001636511712.html>
- Soldering equipment: soldering iron, flux and solder wire.
- 3d printer (stl files for the solar cell tracker mount are attached).
- Photoresistors x 4. I'm using the 5539 photoresistors. You can search these on Ali express, or buy them from your local electronics retailer. Best to get a bunch of these because you will need to find the four that have the closest match for outdoor use.
- Camera pan-tilt mount, <https://www.aliexpress.com/item/32750857279.html>. WARNING: Do not buy these with the servos. They always ship with bad servos which are not fit for any use.
- SG90 servos x 2: Try get the ones with the green potentiometer. See image above.
- ESP32 dev board x 2 (1 for the car and one for the remote controller). I'm using the 38-pin dev board, which I've designed my pcb to fit. You can alter pcb design or hack it for a 30-pin board.
- Analogue joystick (typical arduino joystick).
- led x 1
- jumper cables

- hot glue gun and hot glue
- Standoffs. I'm using 40mm and 12mm. See for example <https://www.aliexpress.com/item/32823612285.html>
- The main file for running the car with the trackers is attached below.

<https://youtu.be/O6xnOEPSv2g>

Step 1: Get the MAC Address of the Car Esp32

First thing is to get the MAC address of the esp32. Write this down somewhere because you will need it later. I put a sticker on my esp32 with the MAC address, so I don't forget it for future projects.

The sketch I used is from [random nerd tutorials](#).

```
// Complete Instructions to Get and Change ESP MAC Address: https://RandomNerdTutorials.com/get-change-esp32-esp8266-mac-address-arduino/  
  
#ifdef ESP32  
  #include <WiFi.h>  
#else  
  #include <ESP8266WiFi.h>  
#endif  
  
void setup(){  
  Serial.begin(115200);  
  Serial.println();  
  Serial.print("ESP Board MAC Address: ");  
  Serial.println(WiFi.macAddress());  
}  
  
void loop(){  
  
}
```

```
// Complete Instructions to Get and Change ESP MAC Address

#ifdef ESP32
    #include <WiFi.h>
#else
    #include <ESP8266WiFi.h>
#endif

void setup(){
    Serial.begin(115200);
    Serial.println();
    Serial.print("ESP Board MAC Address: ");
    Serial.println(WiFi.macAddress());
}

void loop(){

}
```



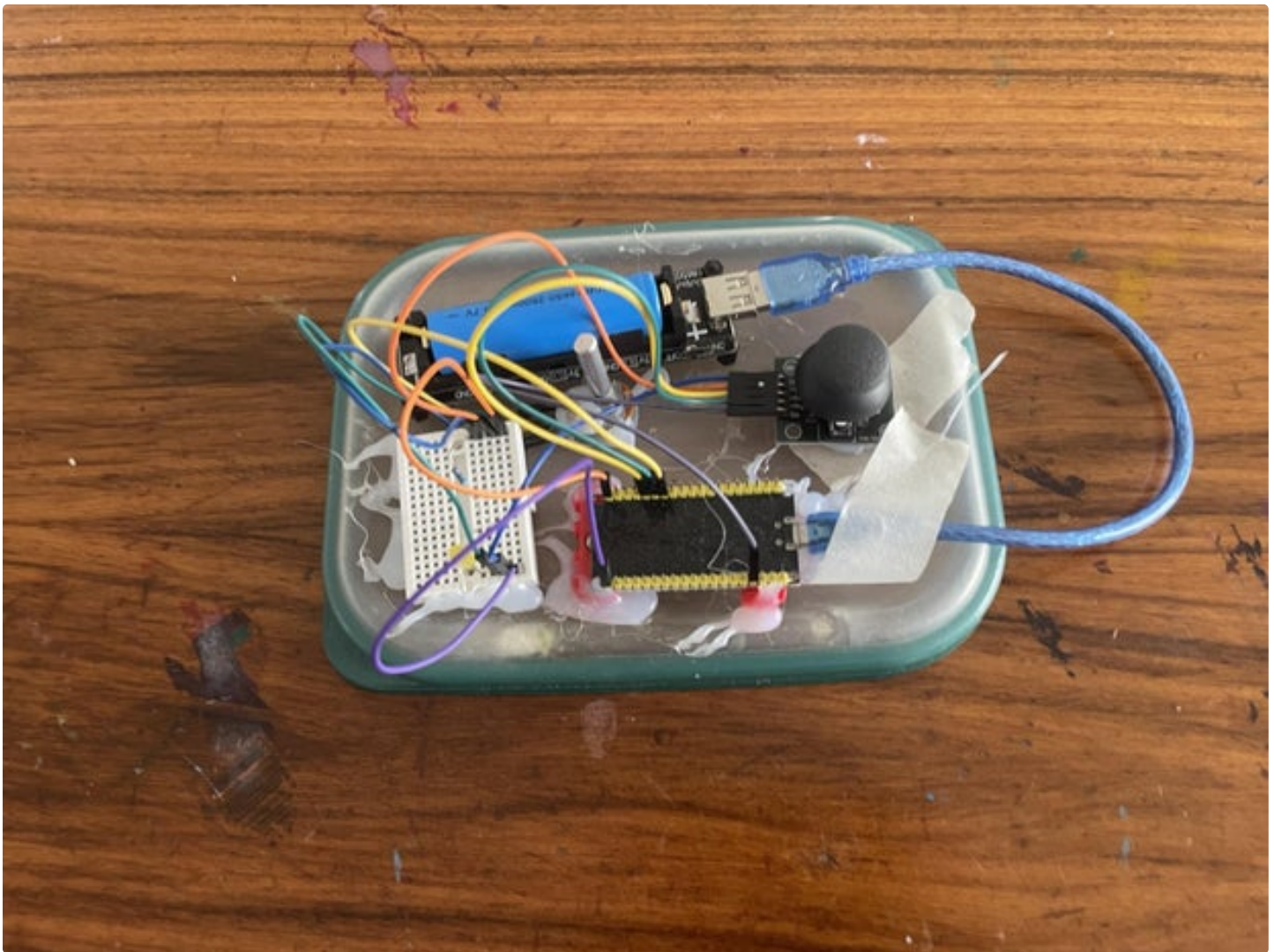
Step 2: Make the Controller

Make the controller. I'm using an arduino analogue joystick, and esp32 and an led. There's also a pot, which isn't used for this particular project. But it's good to have.

The sketch reads the values from the arduino joystick and sends them via esp32-now to the car esp32 dev board. The sketch is attached below.

The pin connections are in the code snippet, where ypin is the y-axis of the joystick, xpin is the x-axis, speed pin is the potentiometer, and breakbutton is the joystick button. Make sure you use the ADC1 pins. See for background <https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/>

```
const int ypin = 35;  
const int xpin = 34;  
const int speedpin = 32;  
const int breakbutton = 15;
```



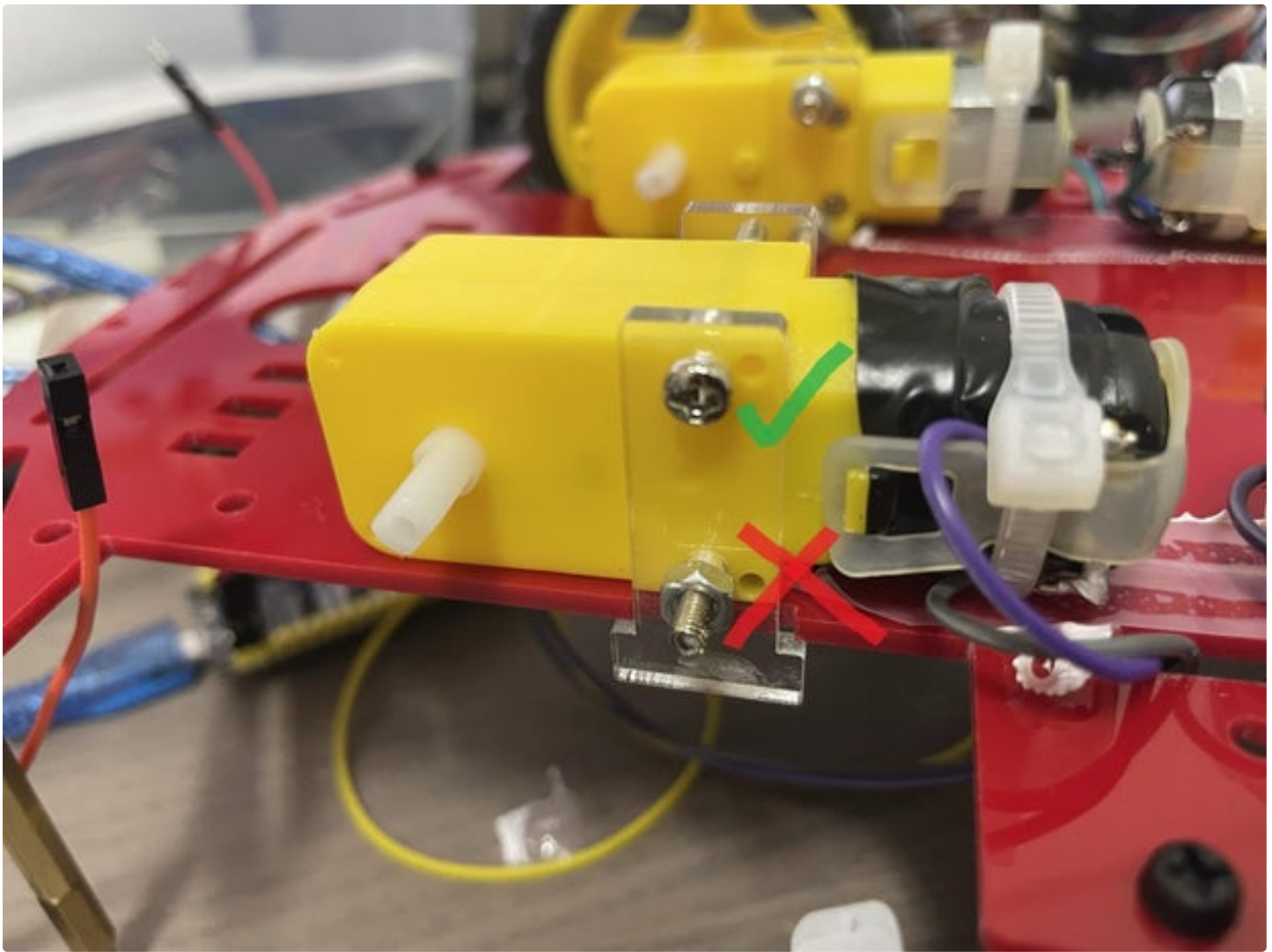
Download

<https://www.instructables.com/FGY/FTQK/LMJ6LC4K/FGYFTQKLMJ6LC4K.ino>

Step 3: Building the Robot Chassis

Build the robot chassis. Most of these will come with ikea like assembly instructions. Make sure to solder on the wires to the connection before securing the motors. Also, make sure no metal bit from the contact is touching the top metal bit of the motors. It's a good idea to secure these with cable ties around the top of the motors so that you don't accidentally pull the contacts off.

Make sure when you screw on the motor holder so that the head of the screw is facing out. In the picture above of the robot, the top screw is correct and the bottom screw is wrong. This is because the sticky out part will catch the wheel and the motor will stall.



Step 4: Wire Up the EPS32-now Car

You can use either a pcb or jumper cables for this step. If you are using a jumper cables it is recommended to use a small breadboard. Make suer that the header pins are soldered so that you can see the pin labels when the TB6612 is attached to a breadboard. This is also how it should be configured for the pcb.

Using a pcb will make the wiring step much easier. I've designed a pcb in fritzing. The fritzing file is attached to this step. And so are the Gerber files. If you want to make changes to the pcb using fritzing, makes sure to select "Routing" -> "Design Rules Check (DRC)" to see if there are any overlapping connections. You can do a 2-layer pcb with fritzing. I got my pcb made up by [pcbway](https://www.pcbway.com/QuickOrderOnline.aspx). Go to <https://www.pcbway.com/QuickOrderOnline.aspx> and there is a button to upload your gerber files as a zip drive.

Next step is to wire up the esp32 and the two motor drivers. If you are using a two wheel drive chassis, you will only need one TB6612 motor controller, but if you are using a four wheel drive chassis (like the one I used), you will need two TB6612s. Wire the corresponding input pins on the two together:

- pwmA (of first driver) -> pwmA (of second driver). I'm using pin 13
- pwmB -> pwmB. Pin 33
- AIN1 -> AIN1. Pin 12
- AIN2 -> AIN2. Pin 27
- BIN1 -> BIN1. Pin 25

- BIN2 -> BIN2. Pin 26

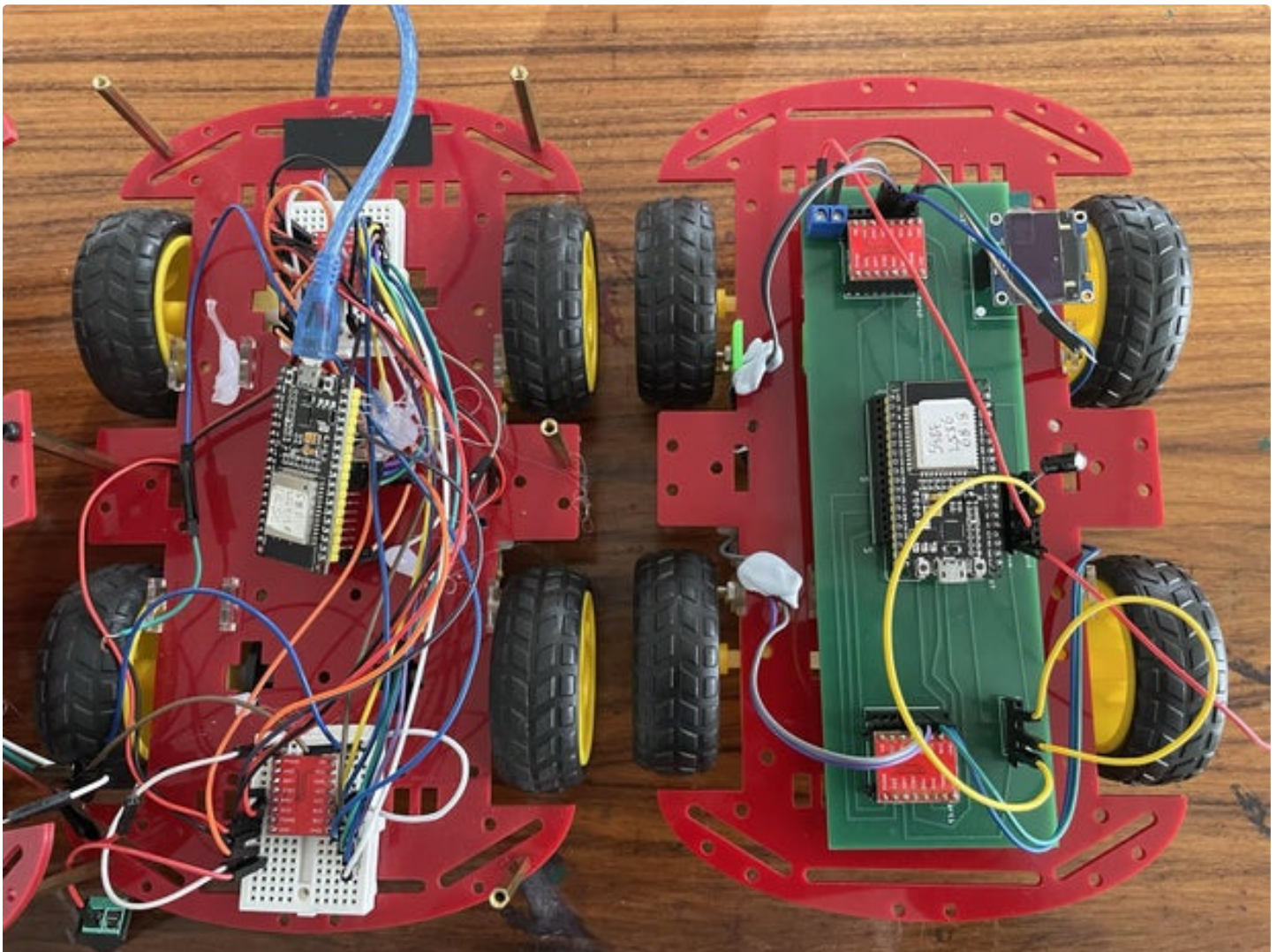
In the code I'm using pin 19 for the STDBY pins, but I've left this unwired in the pcb design in case I want to connect it high. But if I connect it to a digital output, then it gives me the option to set the digital output low, which will mean that the motor controller draws very little current, which is better for charging the batteries with solar cells.

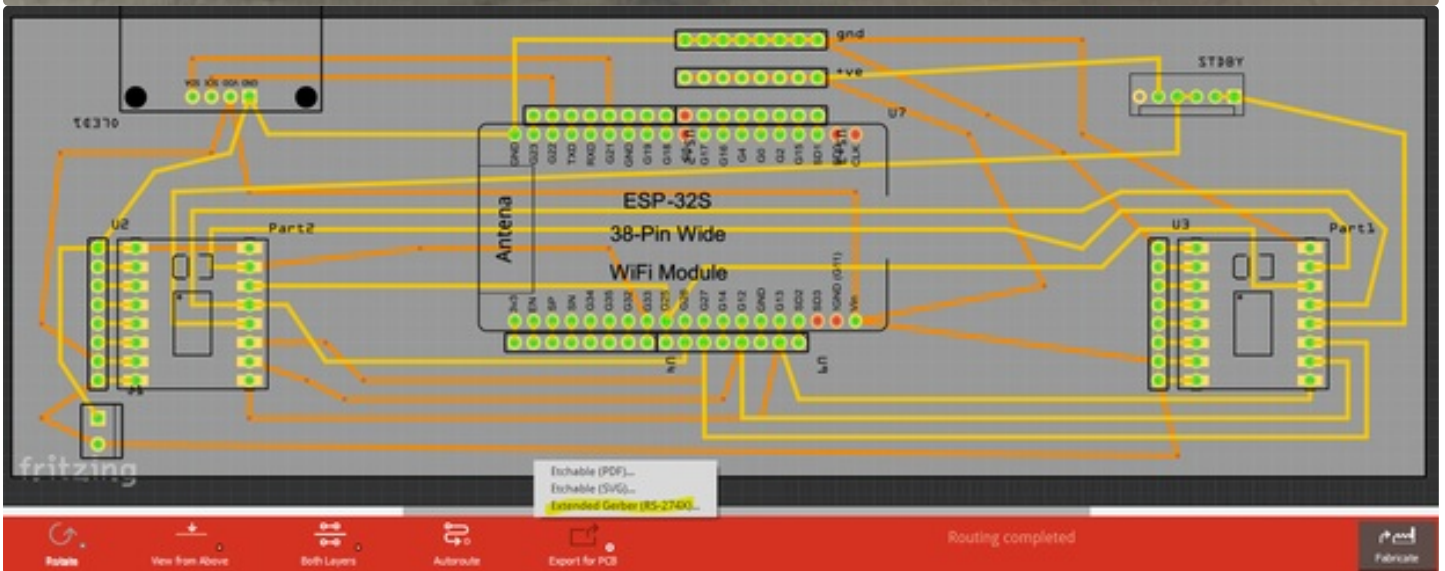
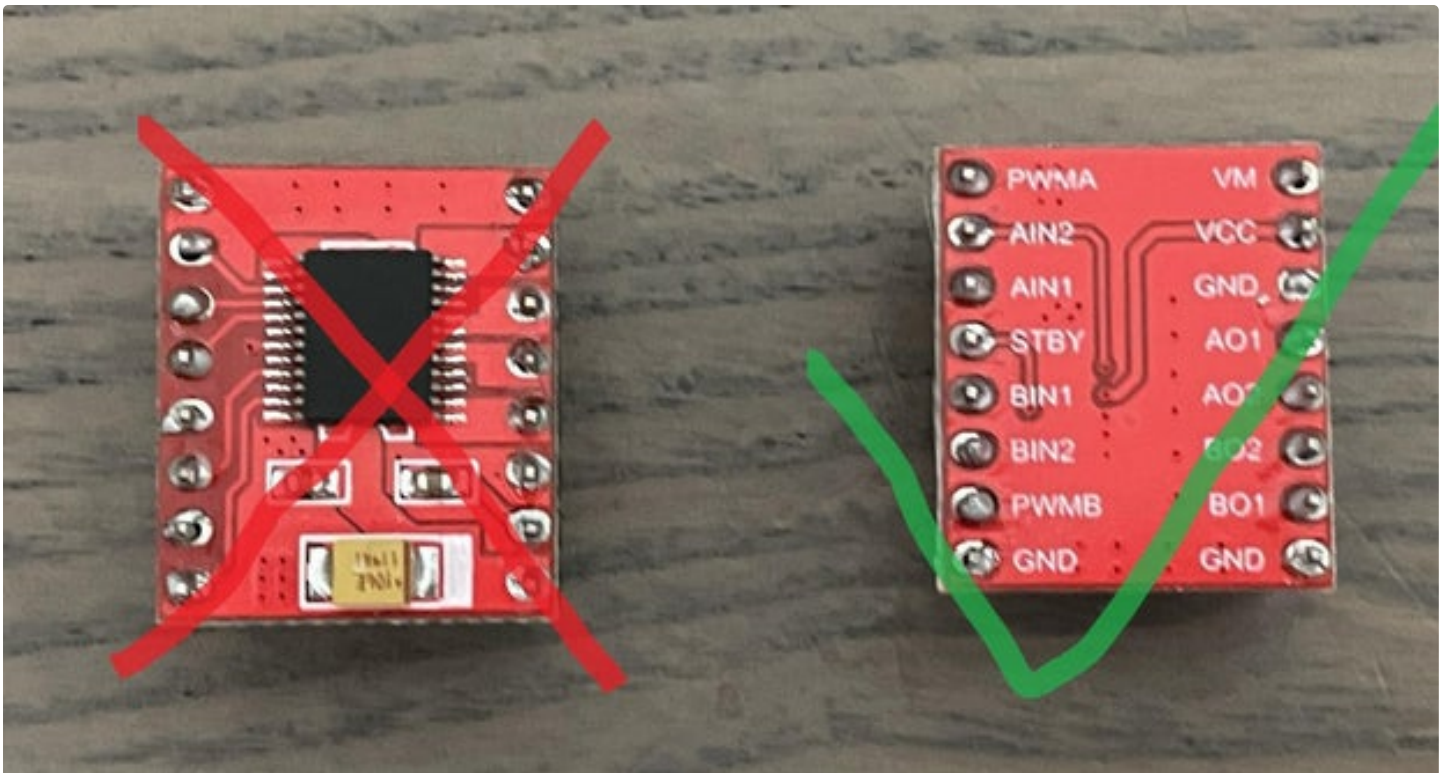
Connect the motors to their respective outputs. So, the back motors are connected to the back driver, and the front motors are connected to the front driver. Note, tying the inputs of the two drivers together will mean that the two left wheels are always going the same direction and the same speed. Similarly for the two right wheels. Also, might need to do some trial and error to get the wheels in the right direction, so you might need to swap AO1 and AO2 for the left motor and swap BO1 and BO2 for the right motor.

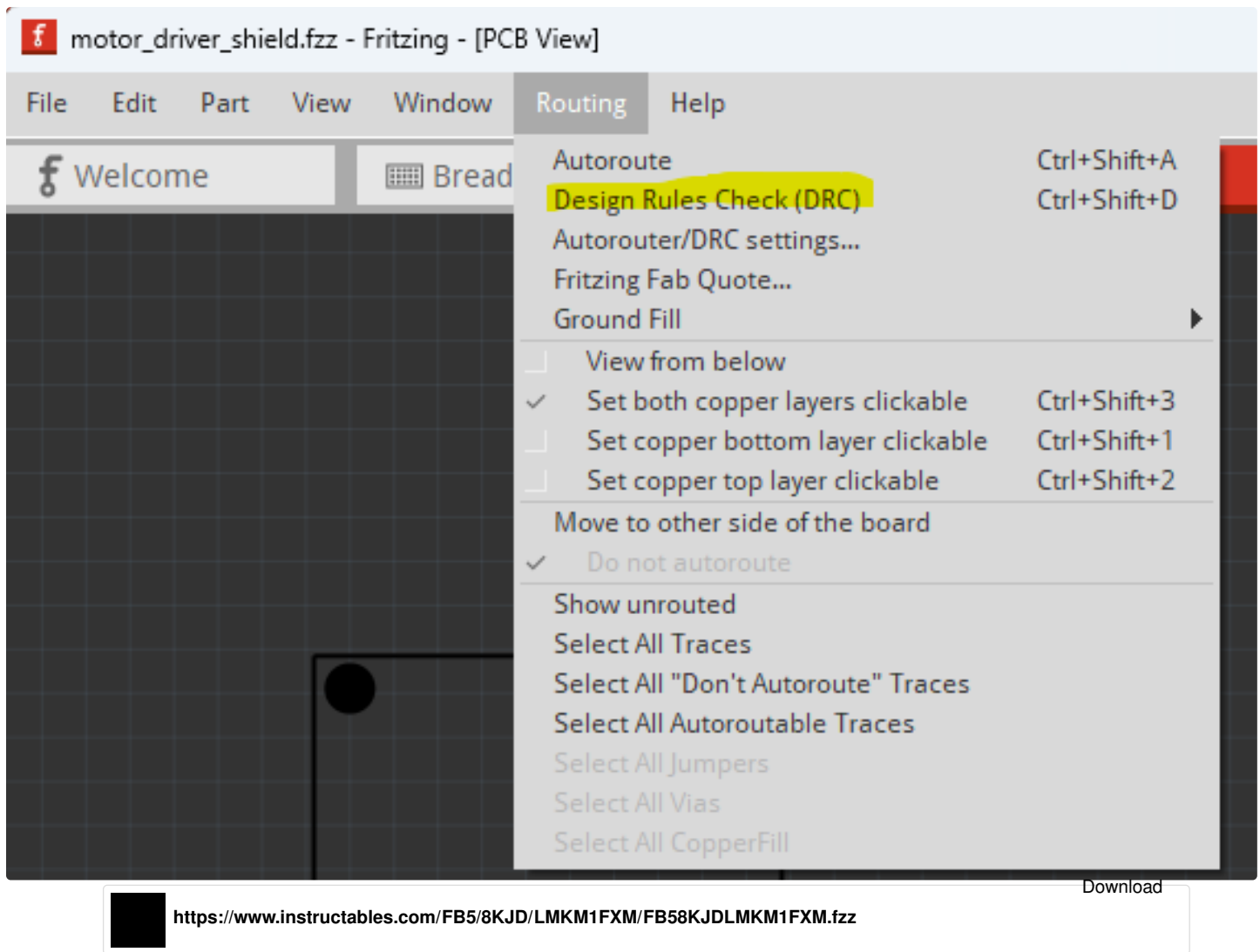
The wiring diagram is shown in the fritzing screen shot.

```
int pwmA_pin = 13; //white
int pwmB_pin = 33; //yellow

int rml1 = 12; // BIN1
int rml2 = 27; // BIN2
int lml1 = 25; // AIN1 H out1 = H | L out1 = L | H short break | L stop purple
int lml2 = 26; // AIN2 L out2 = L | H out2 = H | H short break | L stop brown
int STDBY = 19; //Low is off (current save) //not connected so far
```







Step 5: Upload the Code to the Esp32

Once everything is wired up, you can test the code. This code should also work for the trackers which are wired up in the next step. So once you have this on the esp32, you should not need to re-upload. You can make adjustments to pins and times as you like.

The code is in `car_side_esp32_solar_v2.ino`. This code includes the tracker stuff but you can use it for just the car without the tracker layer connected.

A couple of things to point out:

The two variables defined as `RTC_NOINIT_ATTR` are persistent for when the esp32 goes to sleep and wakes up again.

```
RTC_NOINIT_ATTR int servo_top_persistent;
RTC_NOINIT_ATTR int servo_bottom_persistent; //these are the persistent servo values
```

I'm putting the esp32 to sleep for 10 seconds. This is just for debugging. I would leave it to sleep for longer if I'm using it.

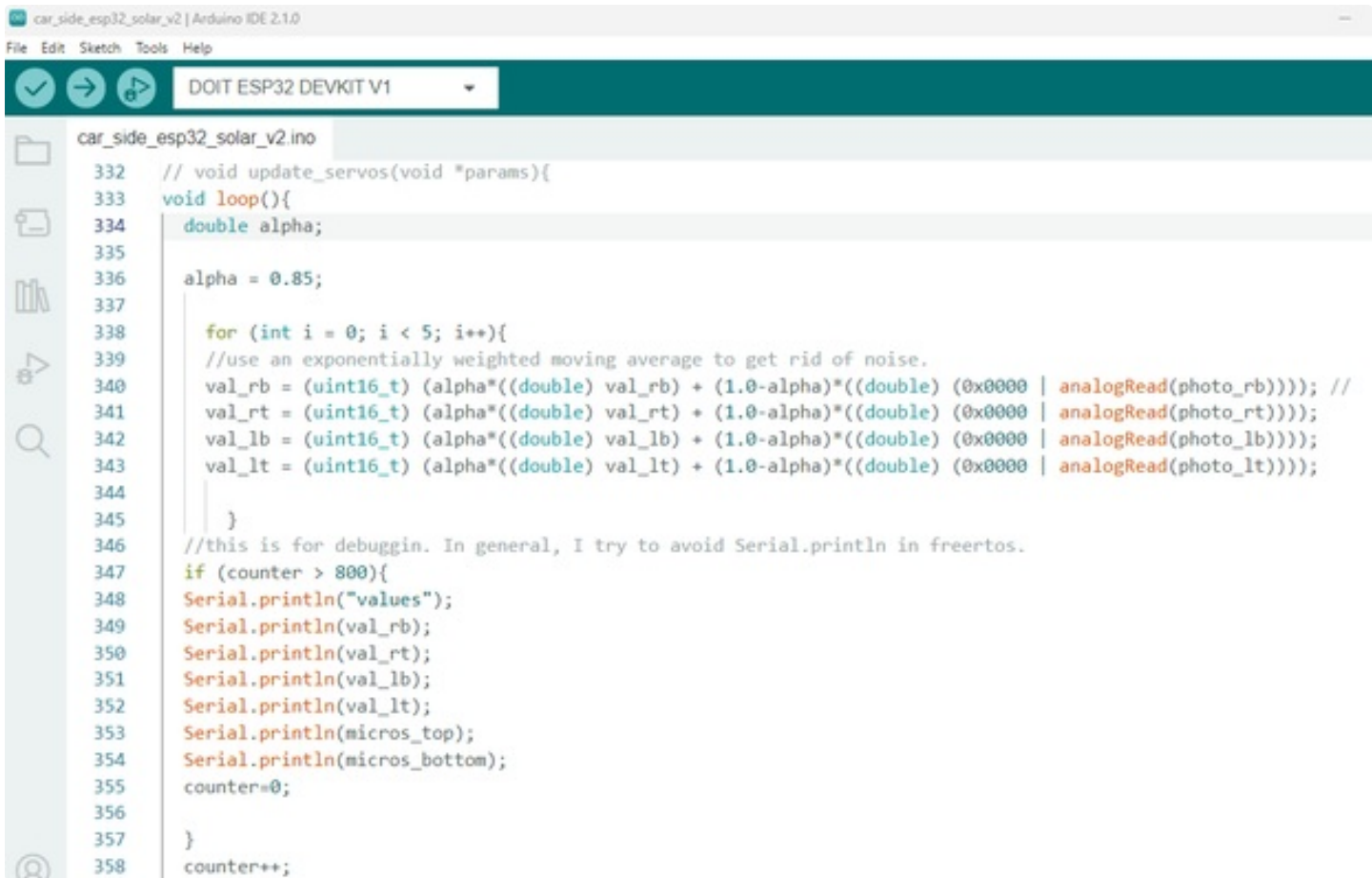
```
#define uS_TO_S_FACTOR 1000000 /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP 10 /* Time ESP32 will go to sleep (in seconds) */
```

The esp32 will fall asleep if it hasn't heard anything from the controller for 8 seconds. This is for debugging and you can change it to suit your needs. Note, if you turn the controller back on and its sending out a signal, when the esp32 wakes up, it won't go back to sleep and you will be able to drive it with the controller. But if the controller is off, it will go back to sleep.

```
if((millis()-t_last_com) > 8000){  
  
    //this will put the robot into deep sleep if it doesn't hear from the remote control in 4s.  
    digitalWrite(STDBY,LOW);  
  
    servo_top_persistent=micros_top;  
    servo_bottom_persistent=micros_bottom;  
  
    Serial.println("going to sleep");  
    esp_deep_sleep_start();  
}
```

I'm using an exponentially weighted moving average when taking measurements from the photo resistors. This will reduce noise

```
double alpha;  
  
alpha = 0.85;  
  
for (int i = 0; i < 5; i++){  
    //use an exponentially weighted moving average to get rid of noise.  
    val_rb = (uint16_t) (alpha*((double) val_rb) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_rb))));  
    val_rt = (uint16_t) (alpha*((double) val_rt) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_rt))));  
    val_lb = (uint16_t) (alpha*((double) val_lb) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_lb))));  
    val_lt = (uint16_t) (alpha*((double) val_lt) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_lt))));  
}
```



```
332 // void update_servos(void *params){
333 void loop(){
334     double alpha;
335
336     alpha = 0.85;
337
338     for (int i = 0; i < 5; i++){
339         //use an exponentially weighted moving average to get rid of noise.
340         val_rb = (uint16_t) (alpha*((double) val_rb) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_rb)))); //
341         val_rt = (uint16_t) (alpha*((double) val_rt) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_rt))));
342         val_lb = (uint16_t) (alpha*((double) val_lb) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_lb))));
343         val_lt = (uint16_t) (alpha*((double) val_lt) + (1.0-alpha)*((double) (0x0000 | analogRead(photo_lt))));
344
345     }
346     //this is for debuggin. In general, I try to avoid Serial.println in freertos.
347     if (counter > 800){
348         Serial.println("values");
349         Serial.println(val_rb);
350         Serial.println(val_rt);
351         Serial.println(val_lb);
352         Serial.println(val_lt);
353         Serial.println(micros_top);
354         Serial.println(micros_bottom);
355         counter=0;
356
357     }
358     counter++;
```

Download

<https://www.instructables.com/F0C/NCGY/LMKM1DFQ/F0CNCGYLMKM1DFQ.ino>

Step 6: Make the Tracker Mount

Print out the tracker mount. The stl file is attached below. I've designed the mount so that it does not shade the solar cell. Maker sure the corners of the tracker are painted black because otherwise light will reflect and confuse the photo resistors.

The idea is that there will be one photo resistor in each corner. The photo resistors will all be detecting maximum sunlight when the tracker is facing directly into the sun. So, if the photo resistor on the right is detecting more sunlight than the photo resistor on the left, the tracker will move right, so that the left can share in some of the sunlight. Similarly, if the photo resistor on the top is detecting more sunlight that the one on the bottom, the tracker will tilt upwards so that the bottom photo resistor can get more light.

Now, getting this to work indoors is easy, but getting it working outside requires one to be a bit more careful.

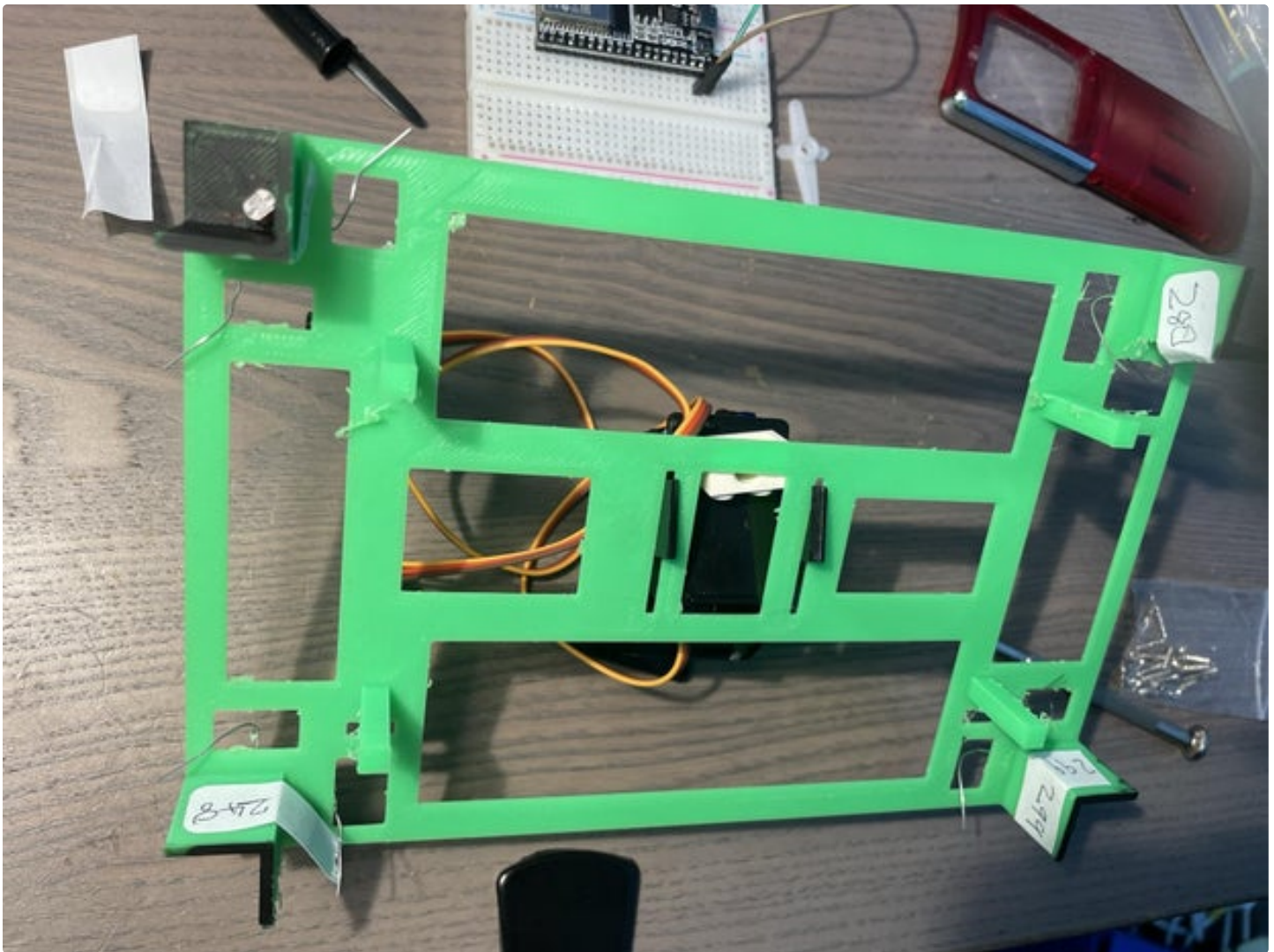
I used 5539 photo resistors. These are rated as 30k to 90k ohm for indoor light and 5M ohm for dark. In sunlight, they behave very differently. So, you must be careful here or else the tracker will not work outside.

1. **Firstly, you need to take about 10 of them (they're 20 for a dollar on ali express) outside and measure the resistance of each one with a multi-meter. I measured them between about 200 and 400 ohms in bright sunlight.**
2. **Next take the four that are closest together in resistance. And out of these four put the two highest ones on opposite corners of the mount. Remember which two these are. We will come back to them later.**

3. **Having these on opposite corners, there will be a strong one facing off against a weak one and a weak one facing off against a strong one. This will add more stability.**
4. Place a photoresistor in each of the four corners. There should be two small holes in each corner for the photoresistor contacts to pass through.
5. On each photo resistor connect a 2.2k ohm resistor to one side. Solder a jumper cable to the other side of the resistor. This will connect to pin 19. I've broken off the side rail of a small bread board, and connected one rail to pin 19 and the other rail to ground. Could also connect to 3.3v, but this means the current through the photo resistors won't switch off when the esp32 goes to sleep during battery charging. I've also placed a small capacitor across the rail (around 100 micro farads).
6. Solder a jumper cable at the junction of the photo resistor and the 2.2k ohm resistor. This will act as a voltage divider, so the voltage goes low when there is more sun (lower resistance on the photo resistor). Each of these will connect to an ADC pin on the esp32. I'm using pins 34, 35, 36 and 39. Make sure you use ADC1 pins because the ADC2 pins won't work with any radio function.
7. Connect the other side of the two diagonal photo resistors with the slightly **lower resistance** to earth (by soldering on a jumper cable).
8. Connect the other side of the two diagonal photo resistors with the slightly **higher resistance** to a small resistance (I'm using 100 ohms, but in retrospect I think 50 ohms might be better) and then to earth. This will further strengthen them.

Now you should have the mount of the tracker all wired up. It should be ready to go on the camera pan-tilt. Note, I purposely designed the mount with a lot of missing squares so that it would be light because the servos aren't very powerful.





| | | |
|--|---|-----------------------|
| | https://www.instructables.com/FU1/J9N8/LMKM1FTH/FU1J9N8LMKM1FTH.fzz | Download |
| | https://www.instructables.com/F0X/IYEG/LMKM1FVY/F0XIYEGLMKM1FVY.stl | View in 3D Download |

Step 7: Pan-tilt Mechanism

Assemble the pan-tilt mechanism. I bought mine from ali express for around 2 AUD. Don't buy the ones that come with the servos because those servos don't really work. Servos are one of those things that's like maple syrup - the cheap ones are never good. Try to centre the servos at 90 degrees before you attach them. Most sg90 servos have between 0 and 180 degree rotation.

I used hot glue to attach the solar cell tracker mount to the pan-tilt mechanism. When you attach the mount, try to have it leaning backwards while servo is flat. This is because the pan-tilt mechanism won't be able to rotate as far backward as it can forward, so we need to compensate because we want the solar cell to tilt in both directions.

The pan-tilt should be attached to the top layer of the car chassis. I drilled four M2 holes to attach this. This will provide a stable base for testing as well as be convenient to attach to the car.

Calibrate the servos (mostly for top servo)

Due to the constraints of the pan-tilt mechanism, you will not be able to rotate the top servo all the way back. It will get

jammed at a certain number of degrees. connect the servos to the esp32, with the bottom servo on pin 4 and the top servo on pin 16. Then upload the sketch "soalr_tracker_servo_calibrate.ino" to the esp32 and you can put in values between 750 microseconds and 2250 microseconds at the serial monitor to test the range of motion. The code pulses the servo at a frequency of 50Hz to get it to turn (which is what the servo library is doing under the hood). We will need the maximum and minimum values for each servo for the main sketch.

You might be wondering why I am pulsing the servo with a pwm signal, and I don't just don't use a Servo object, which provides a servo.attach(), servo.write() functions. There are a couple of reasons for this:

1. Most importantly, I want to move the servos a fraction of a degree: the servo library only allows me to move the servos in whole number of degrees, while pulsing it with microseconds allows me to move it just under one third of a degree at a time. If I move it too far, I can generate instability in the tracker.
2. Also, I can choose which timer I use for the servo. Remember, I will eventually have some dc motors connected which will also be using a pwm signal at a different frequency, and I don't want to get the timers mixed up.

```

#include <Arduino.h>

const int top_servo = 16; // Top servo
const int bottom_servo = 4; //Bottomservo
const int servoFrequency = 50; // PWM frequency in Hz
const int bottom_ch = 4; //timer channels
const int top_ch = 5;

//analogue inputs for photoresistors
const int photo_rb = 39; //right bottom
const int photo_rt = 36; //right top
const int photo_lb = 35; //left bottom
const int photo_lt = 34; //left top

void setup() {
  // Configure LED Controller Channel 4 for Servo 1
  ledcSetup(bottom_ch, servoFrequency, 16);
  ledcAttachPin(bottom_servo, bottom_ch);

  // Configure LED Controller Channel 5 for Servo 2
  ledcSetup(top_ch, servoFrequency, 16);
  ledcAttachPin(top_servo, top_ch);

  Serial.begin(115200); // Initialize serial communication
  Serial.println("Enter desired microseconds (750 to 2250) for Servo 2:");
}

void loop() {
  if (Serial.available() > 0) {
    int microsInput = Serial.parseInt(); // Read the input from serial monitor

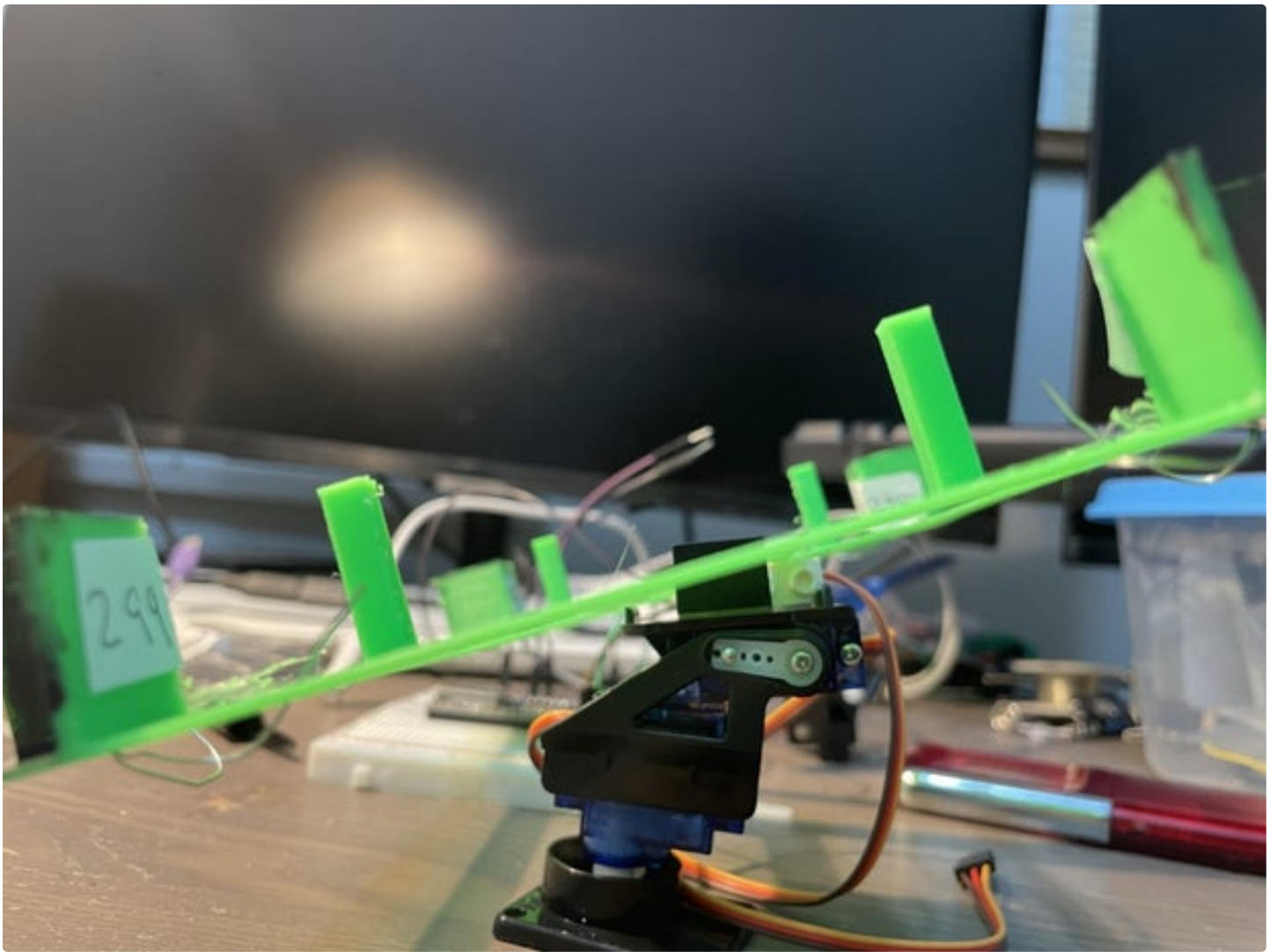
    // Ensure the input is within the valid range
    if (microsInput >= 350 && microsInput <= 2750) {
      writeServoMicros(top_ch, microsInput); // change to bottom_ch to calibrate bottom servo
      Serial.print("Setting Servo 2 to ");
      Serial.print(microsInput);
      Serial.println(" microseconds");
    } else {
      Serial.println("Invalid input. Enter a value between 750 and 2250 microseconds.");
    }
  }
}

void writeServoMicros(int ch, int micros) {
  // Map the desired pulse width (micros) to the PWM range (0-65535)
  int dutyCycle = map(micros, 0, 20000, 0, 65535);

  // Set the PWM duty cycle to move the servo
  ledcWrite(ch, dutyCycle);
}

```

Next you can test the tracker with the main code (attached to step 4). This code will print out to the serial monitor the ADC values of the photo resistors. It is recommended to test this code on the tracker without





Download

<https://www.instructables.com/FUZ/B8SN/LMKM177P/FUZB8SNLMKM177P.ino>

Step 8: Attach the Tracker to the Car and Add the Solar Cell

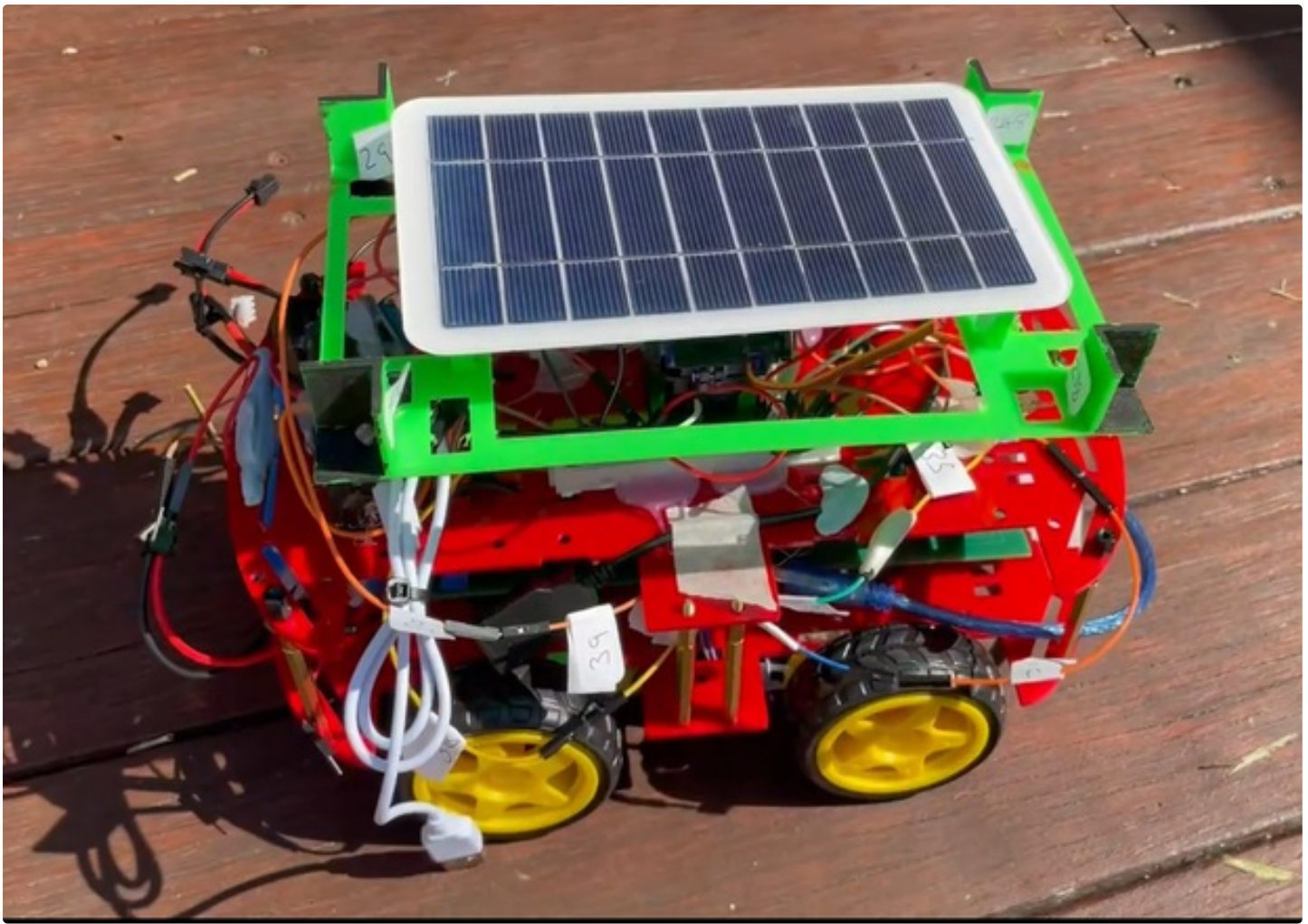
The next step is to attach the tracker to the base of the car. I have drilled four holes in the top layer of the robot chassis to mount the pan-tilt mechanism with M2 screws. Then I have used standoffs to mount the top layer of the chassis onto the

bottom layer. The standoffs that came with the car were not tall enough, so I had to get some taller ones. I use a 40mm and a 12mm stacked.

I attached the solar cell to the tracker mount with some hot glue.







Step 9: Battery Case, Charger and Boost Converter

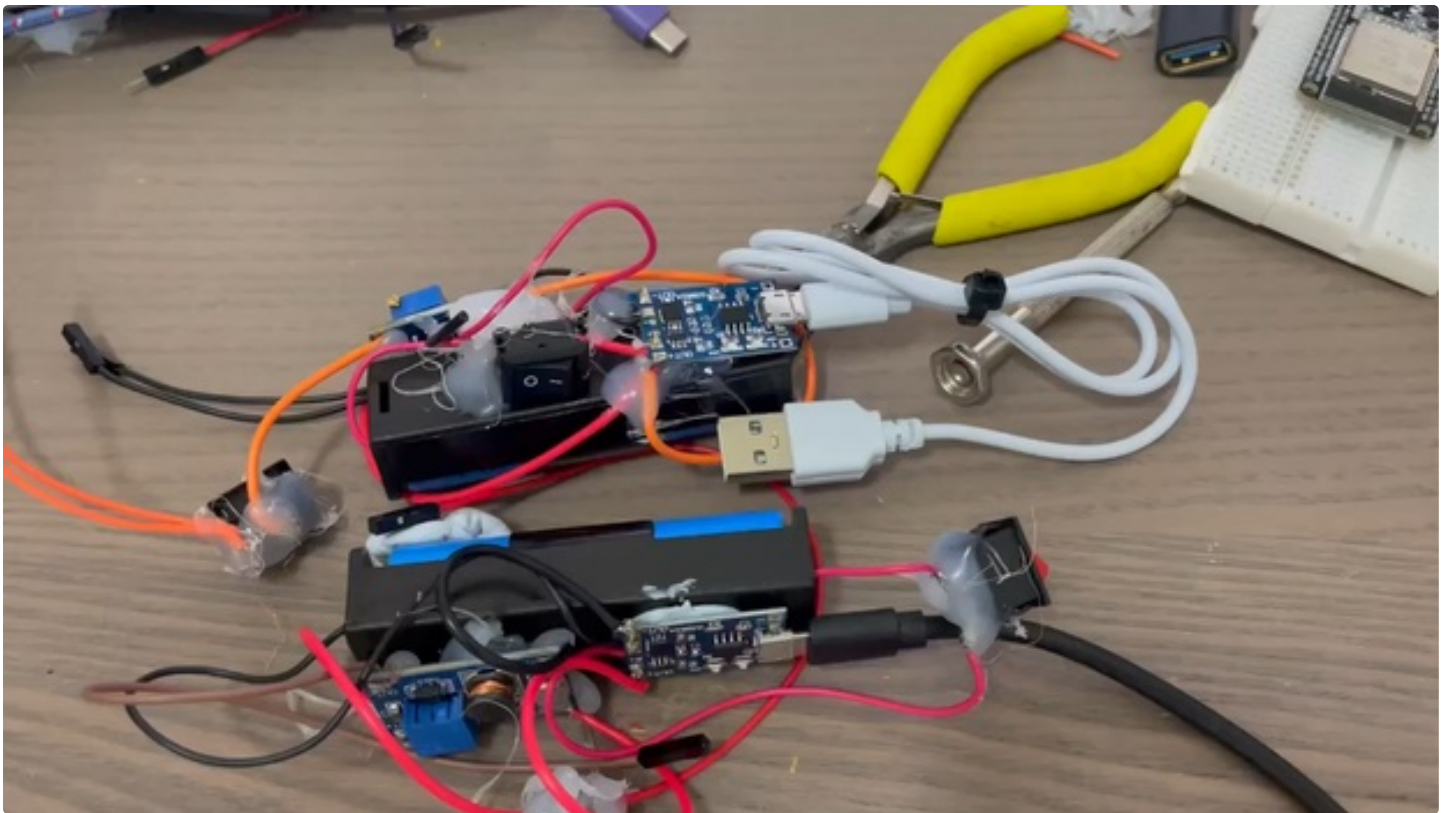
I used two 18650 batteries. One for the esp32 & servos, and another battery for the motors. The esp32/servo battery was at 5v, while the motors battery was at 6v. This means I can only have one battery charging at a time from the solar cell. If you've never used a tp4056, I recommend you doing a quick few minute review.

To each battery:

1. Place inside battery holder.
2. Solder +/- to battery terminals on the tp4056.
3. Then the outputs of the tp4056 are going to the inputs of the MT3608 boost converter.
4. The outputs of the boost converter are powering the esp32 car and the motors. Note, the motors are powered through the TB6612 motor drivers.

Note, if you look at the pictures, there are two side rails from small bread boards on the top layer:

- One is a 3.3v rail (GND and pin 18 held high), which is connected to the top and bottom of the photoresistor configuration.
- The other is the 5V rail which is connected to the servos and the



Step 10: Find a Sunny Spot to Test the Car

When I finished the sun was almost going down. I found a window which still had some sun coming through it and managed to test it. It is best to test with one servo plugged in at a time. Good luck and let me know if you have any difficulties or if there is anything I haven't explained yet.

So long and thanks for all the fish.

