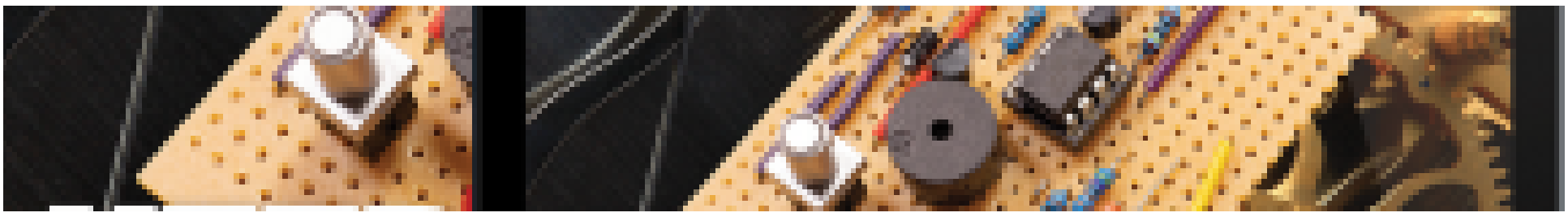


PROJECTS

Seeing Stars

Hacking a Motorised Telescope with Arduino

DIYODE MAGAZINE

Issue 52, November 2021

This article includes additional downloadable resources.
Please log in to access.

We show you how to reverse engineer a cheap GoTo telescope so that you can add automation using an Arduino.

BUILD TIME: A WEEKEND
DIFFICULTY RATING: INTERMEDIATE

In the age where DIY electronic parts are cheaper than ever, it's no surprise that makers are finding more creative ways to add more features and convenience to their favourite household gadgets. While designing and assembling projects from scratch is usually what we do at DIYODE, some find it more fun to improve existing products. Reverse engineering, tinkering, hacking, whatever you want to call it – the process of modifying something that isn't exactly how you want it is one of the benefits of learning electronics!

This project started a couple of months back when we found a cheap telescope on Gumtree (for our overseas readers, an Australian website for second-hand goods) for a modest $200. This telescope, in particular, is a GoTo model. It has a set of motors that, when requested by a hand controller, will 'slew' to any object in the sky. This saves the hassle of trying to manually mess about with knobs and instead will move the telescope to precise positions at the press of a button.

It's great for beginners because it only takes some very basic astronomical knowledge to get started with observing Jupiter's moons, Saturn's rings and the wonderful glow of bright nebulae like the Orion Nebula. However, the telescope we picked up was by no means perfect. The hand controller it came with resembles a mobile phone from the mid-1990s and is comparably frustrating to use.

(/_images/617513f0c672e0b43e7fd15f)

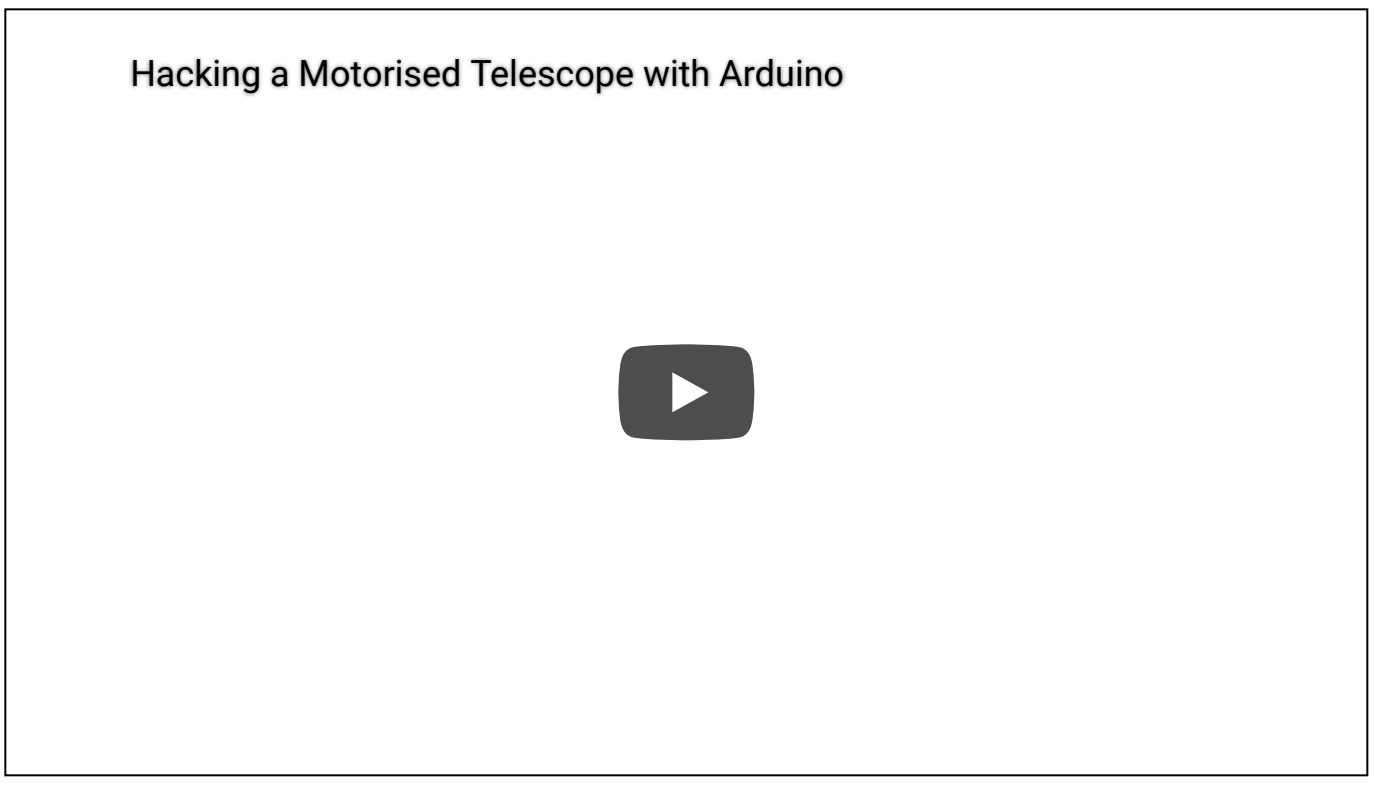
114mm Bushnell Northstar GoTo Telescope

On GoTo telescopes, there are four cardinal direction buttons on the controller that allows the user to move the motors on a desired axis. However, for some reason on this telescope, it ramps the motors to full speed whenever one of the buttons are pressed. This makes it near-on impossible to manually position the telescope to a specific object in the sky if the automated movement is inaccurate. Imagine trying to drive a car if the accelerator was a ON-OFF switch rather than a continuously variable pedal – it's either flooring it or nothing at all. The hand controller also has a small speaker with voice recordings that give information about the observed object. It is a cool concept, but when you want to enjoy the peace and quiet stargazing, it's incredibly annoying.

This telescope, despite what the instruction manual says, does not have tracking either. Since the Earth rotates, stars in the sky move so the telescope must continually update its position to keep the object in the eyepiece. Without tracking, the telescope doesn't move, and the object that we're trying to track simply disappears from the eyepiece after a few seconds.

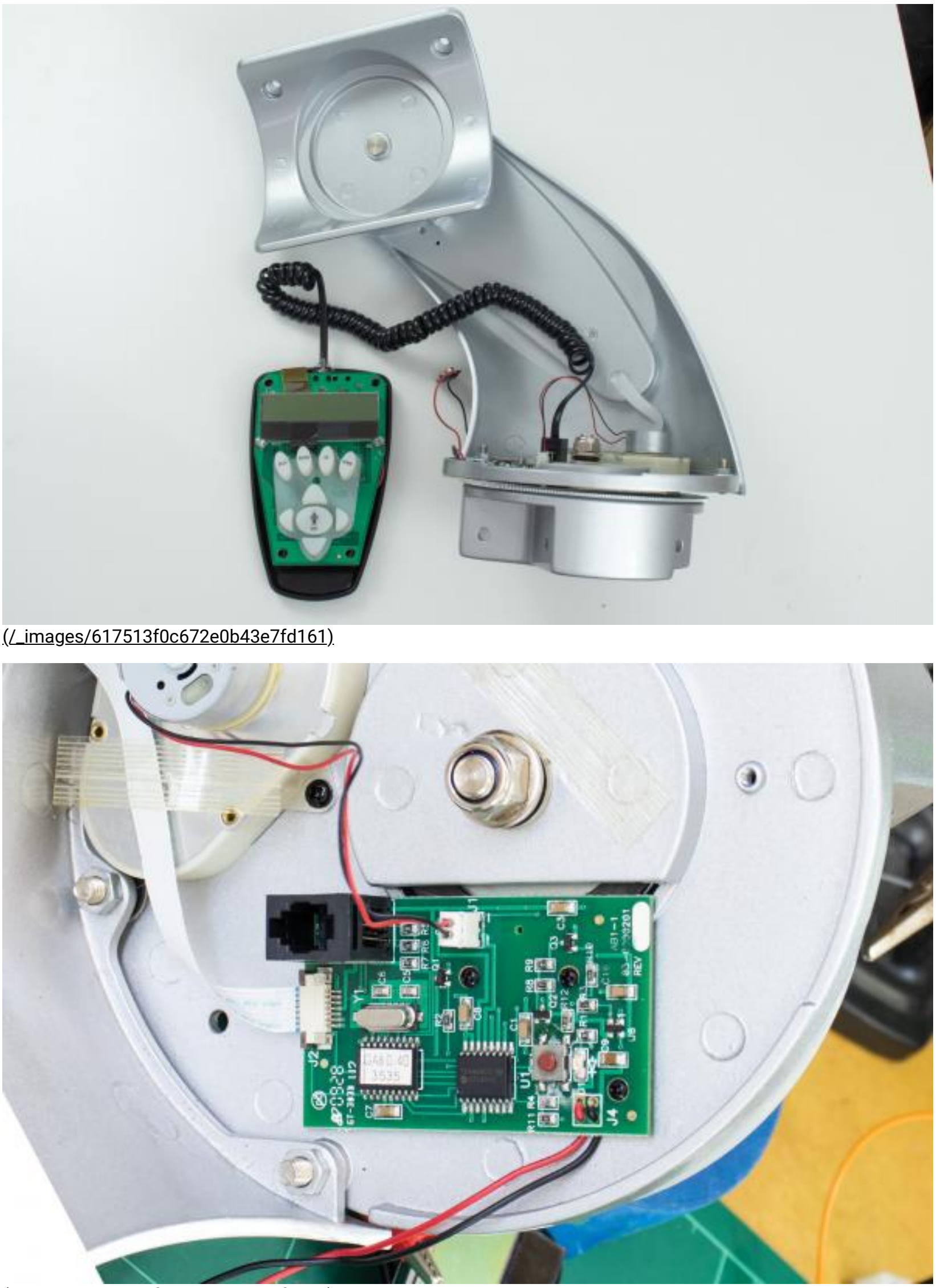
Our objective was to fix the issues we described above and implement our very own Arduino-based telescope controller. Rather than only focusing on the workings of this exact telescope model, we want to provide a broader overview of how to reverse engineer products like this and show you how you can implement your own Arduino-based functionality.

For our Arduino-powered telescope, the 'money shot' feature we wanted to implement was the ability to control it with a computer through a planetarium app. A planetarium is basically just Google Maps for stars and planets, which includes plenty of information and visuals about everything you can see (and can't see) above you. This means that instead of navigating through individual object entries on a small 16x2 LCD screen, we can see a replica of the sky above on a laptop! If the system works correctly, all it would take is a click on any star, planet or nebula and our telescope will move to it automatically and track it. Expensive GoTo telescopes have this feature built-in, but getting it working on our Bushnell reflector telescope is an exciting prospect!



THE TEARDOWN

First, we need to disassemble all of the bits and pieces of the telescope and figure out how the electronics communicate. The hand controller connects to the base of the telescope through an RJ9 port, and after we took it out and disassembled the plastic base, we found a fairly basic PCB with a microcontroller, a four-channel MOSFET motor driver and a set of hall effect sensors for reading the current position of the telescope.



(/_images/617513f0c672e0b43e7fd161)

(/_images/617513f0c672e0b43e7fd163)

It may seem weird that position reading of the telescope is necessary – after all, if we drive the motors at a known speed, shouldn't we know how far it has moved? Unfortunately, the motors included in this telescope are not very high-tech; they're simply DC motors that are stepped down massively through a gearbox to the drive gears. This means that we can't accurately predict where the telescope will end up after we cut power to the motor. The battery voltage, the stiction and lubrication of the motor/gearbox, and any resistance from moving the heavy telescope would quickly throw off any estimated readings. As a side note, expensive telescope mounts often use stepper motors in the same way as 3D printers, where the position is known by stepping the motor at a known angle.

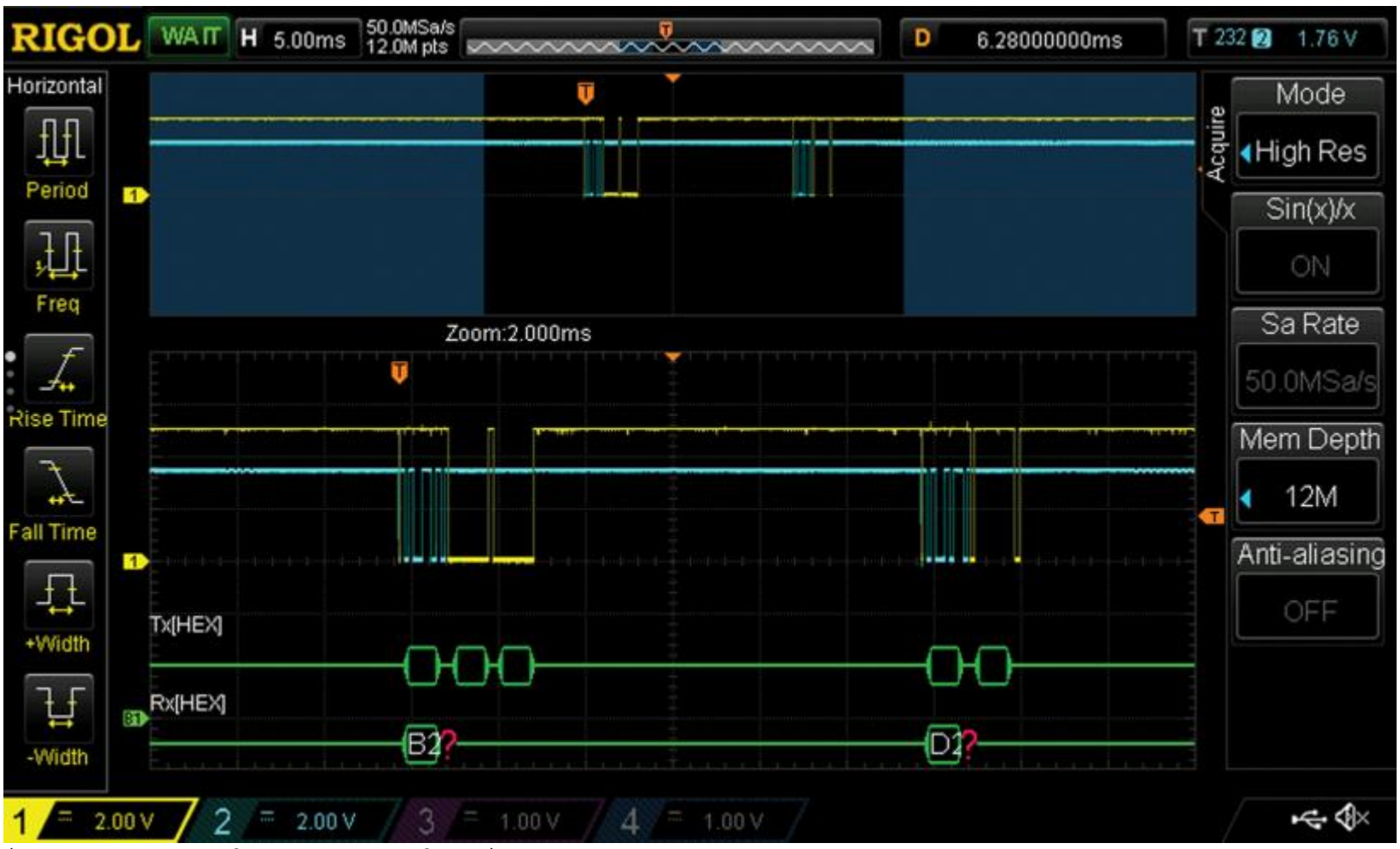
There is a toroidal magnet on each of the Altitude (up/down) and Azimuth (left/right) axes. While we haven't tested it, our educated guess would be that these magnets have rapidly alternating fields. The two hall effect sensors on the nearby PCB can detect the movement direction of the magnets to a precise amount as the poles flip.

DECODING

The next call of order was to decode the communication protocols between the hand controller and the telescope base. We stuck an oscilloscope probe on both the transmission and receive lines of the RJ9 connection and plugged in the hand controller. The DS1054Z scope we're using is able to live-decode RS232 connections, which is super handy when figuring out what does what in the data.

It's worth noting that you don't absolutely need an oscilloscope to reverse-engineer electronics gadgets. In our case, a simple USB-to-Serial converter would've done wonders for pulling the data on the serial lines. For many cases of digital electronics, an Arduino or similar devices can be used for pulling the data you need. But, if you do have access to a scope, it speeds up the process considerably.

The first thing we were able to find was on the receive line – the data from the base going to the hand controller. Every half-second or so, the base sends six bytes of data. The first is a byte with all of its bits set (i.e. 11111111), followed by two bytes that appear to increase or decrease depending on the altitude. The next three bytes were the same, but for the Azimuth.



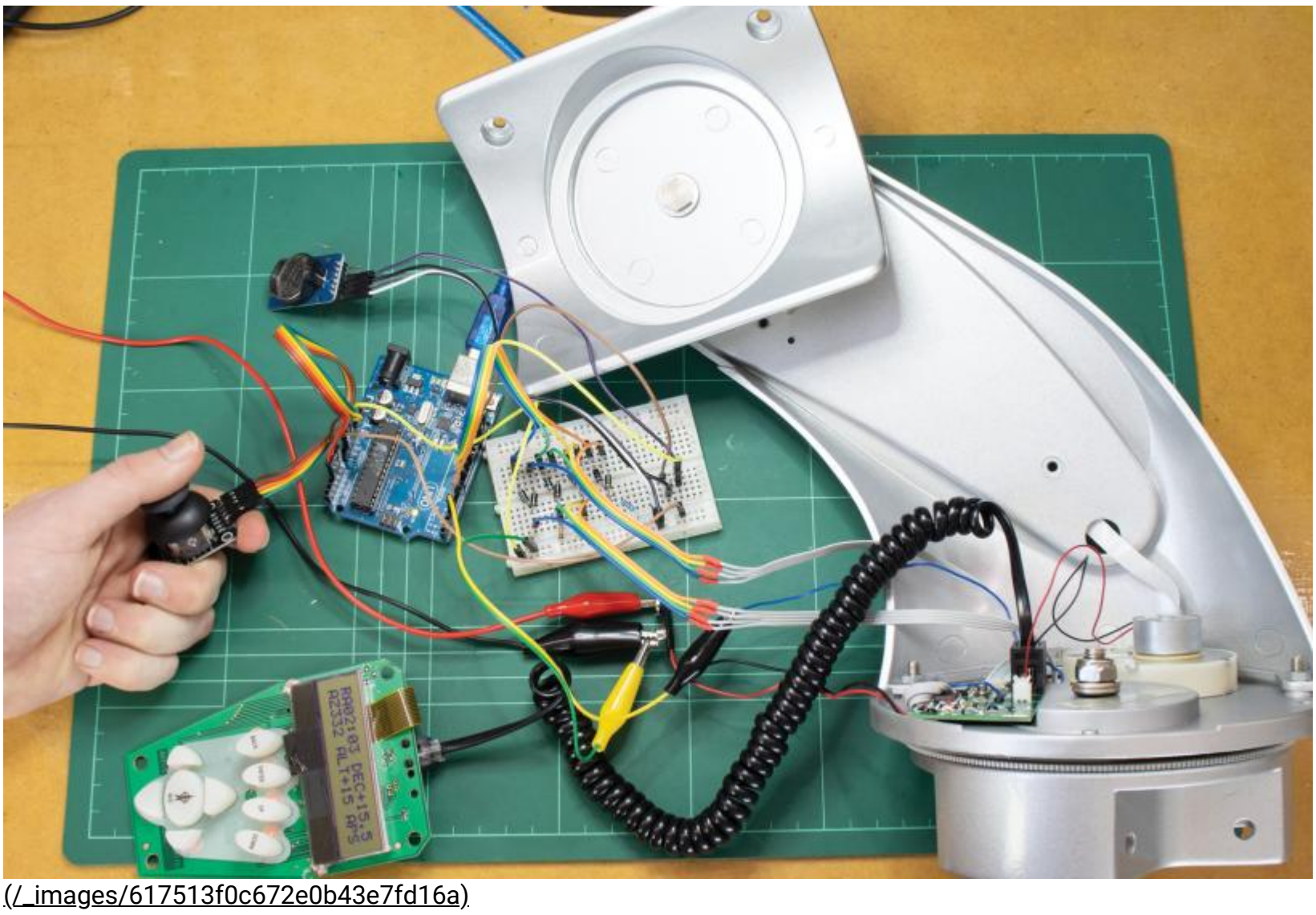
(/_images/617513f0c672e0b43e7fd167)

It became obvious fairly quickly that the position feedback of the telescope was encoded in two 16-bit numbers. This surprised us quite a bit to learn that the telescope is this precise! That's 2^16 = 65,536 individual values across each of the Azimuth and Altitude directions. However, we had to do some tuning to get the right values out of the mount so the full range may actually be smaller.

Annoyingly, this data is only sent whenever the hand controller is plugged in, suggesting the base is not just blindly sending out data whenever the telescope is powered as we were hoping. This means that whatever our final solution is, it will require us to either imitate the commands sent by the hand controller (not fun) or just leave it plugged in while our Arduino snoops on the data.

Next up, we need to decode the data coming from the hand controller. This proved to be more frustrating than expected because, when a direction button is pressed, the controller sends data so fast it's difficult to discern the components of each data frame. Nonetheless, we didn't want to imitate sending button presses to the main controller anyway, because it wouldn't fix the "all-or-nothing" movement problem.

There wasn't any documentation on the protocol either, and it didn't appear to resemble the commands sent by more popular telescope brands like Sky-Watcher or Meade. So, the next best thing is to physically tinker with the driving electronics itself!


(/_images/617513f0c672e0b43e7fd16a)

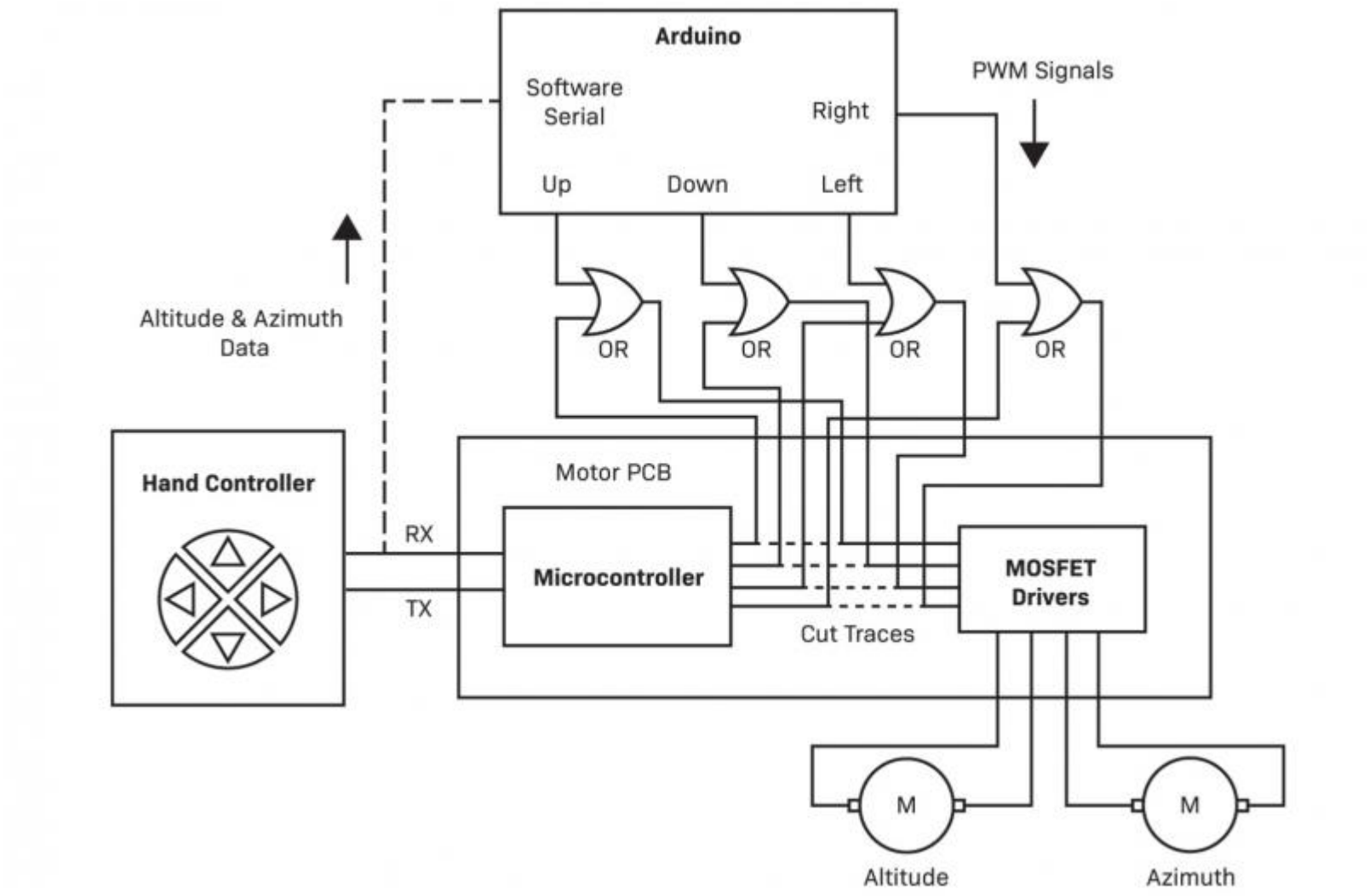
FUNDAMENTAL BUILD

PARTS REQUIRED:

Jaycar Altronics Pakronics

1 x Arduino Uno or Compatible	XC4410 (https://www.jaycar.com.au/p/XC4410)
1 x Joystick Module	XC4422 (https://www.jaycar.com.au/p/XC4422)
8 x 1N4004 Diodes or equivalent	ZR1004 (https://www.jaycar.com.au/p/ZR1004)
4 x 10kΩ Resistors*	RR2798 (https://www.jaycar.com.au/p/RR2798)

* Quantity required, may only be sold in packs. Breadboard and prototyping hardware also required.



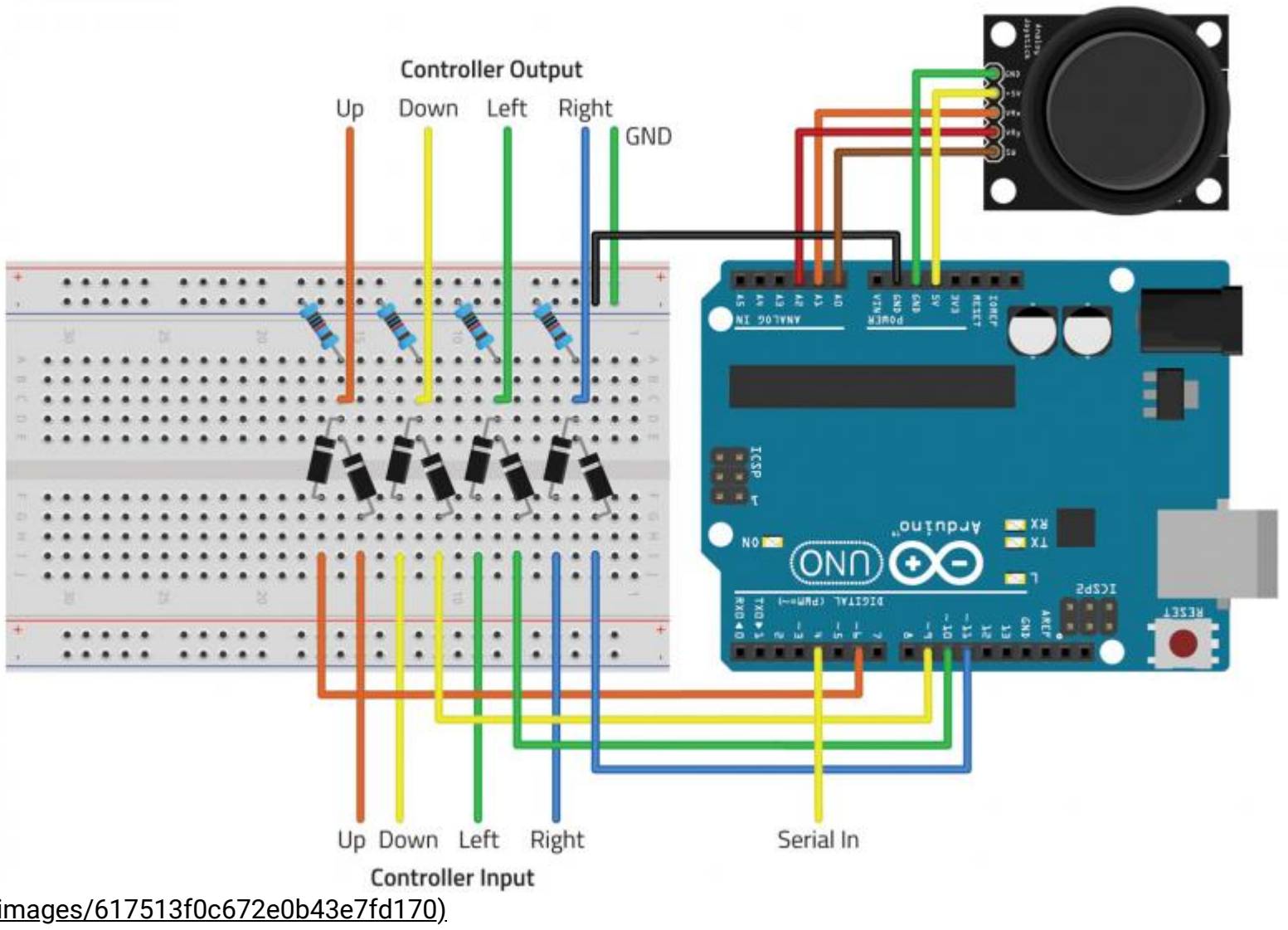
(/_images/617513f0c672e0b43e7fd16e)

Our fundamental build will be a messy but functional version of the final telescope. The first thing we focused on doing was reverse-engineering the motor control system.

After doing some poking around with a multimeter in continuity mode, we whipped up a schematic for the workings of the motor driver system. Fiddling with the microcontroller is not something super interesting since it's difficult to know what code is running on it, so we must look for another place to inject motor signals into the system.

As we mentioned earlier, the telescope has a four channel MOSFET driver with only two motors. Turns out, this system is configured as a H-Bridge system that provides a voltage to different terminals on the motors based on which pins have a digital signal on them. PWM is used here to vary the speed of the motors, which is generated directly by the microcontroller.

Note: We did not end up using the RTC (real time clock) module that you see in the photo above as we were able to request the time from the computer.

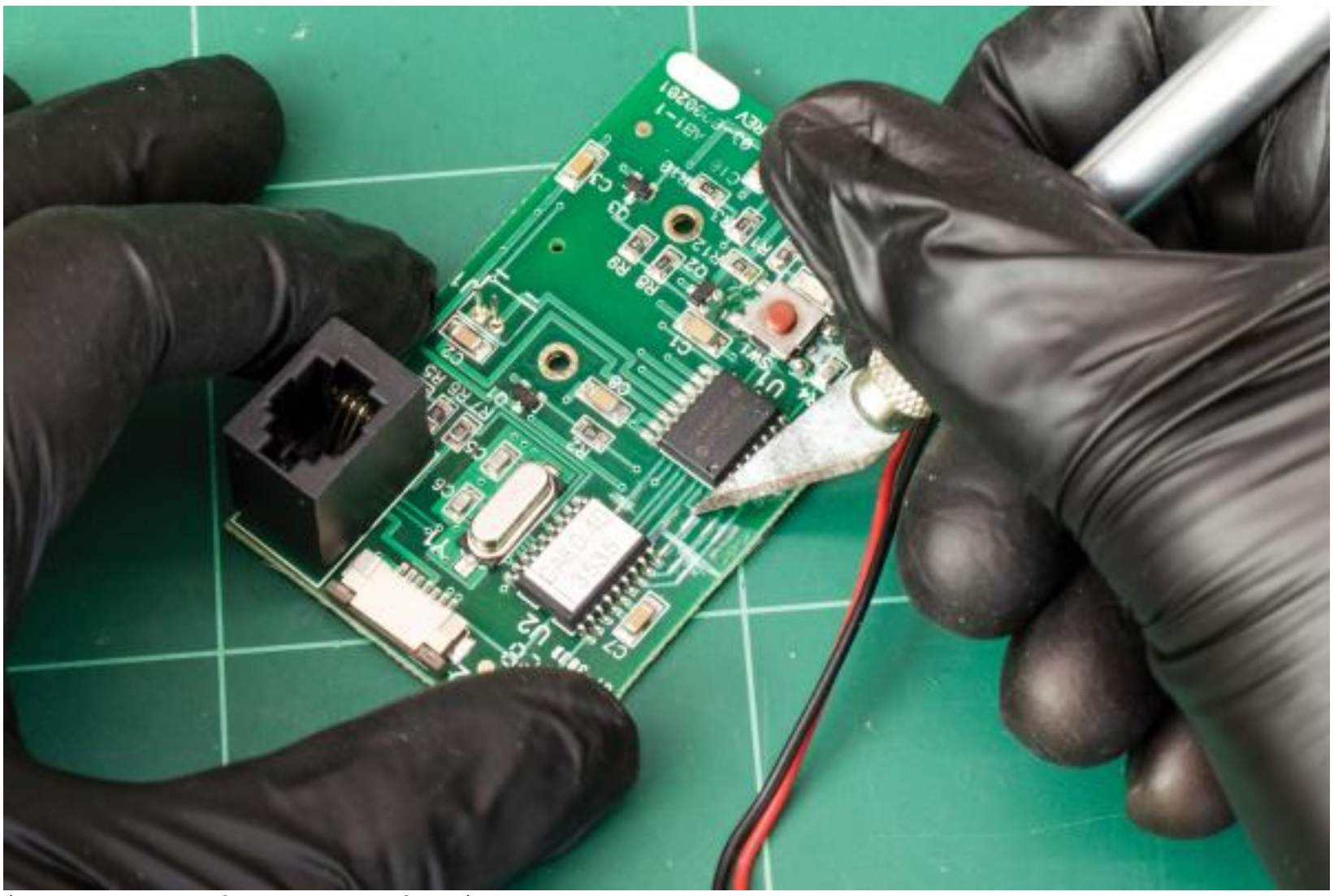


(/_images/617513f0c672e0b43e7fd170)

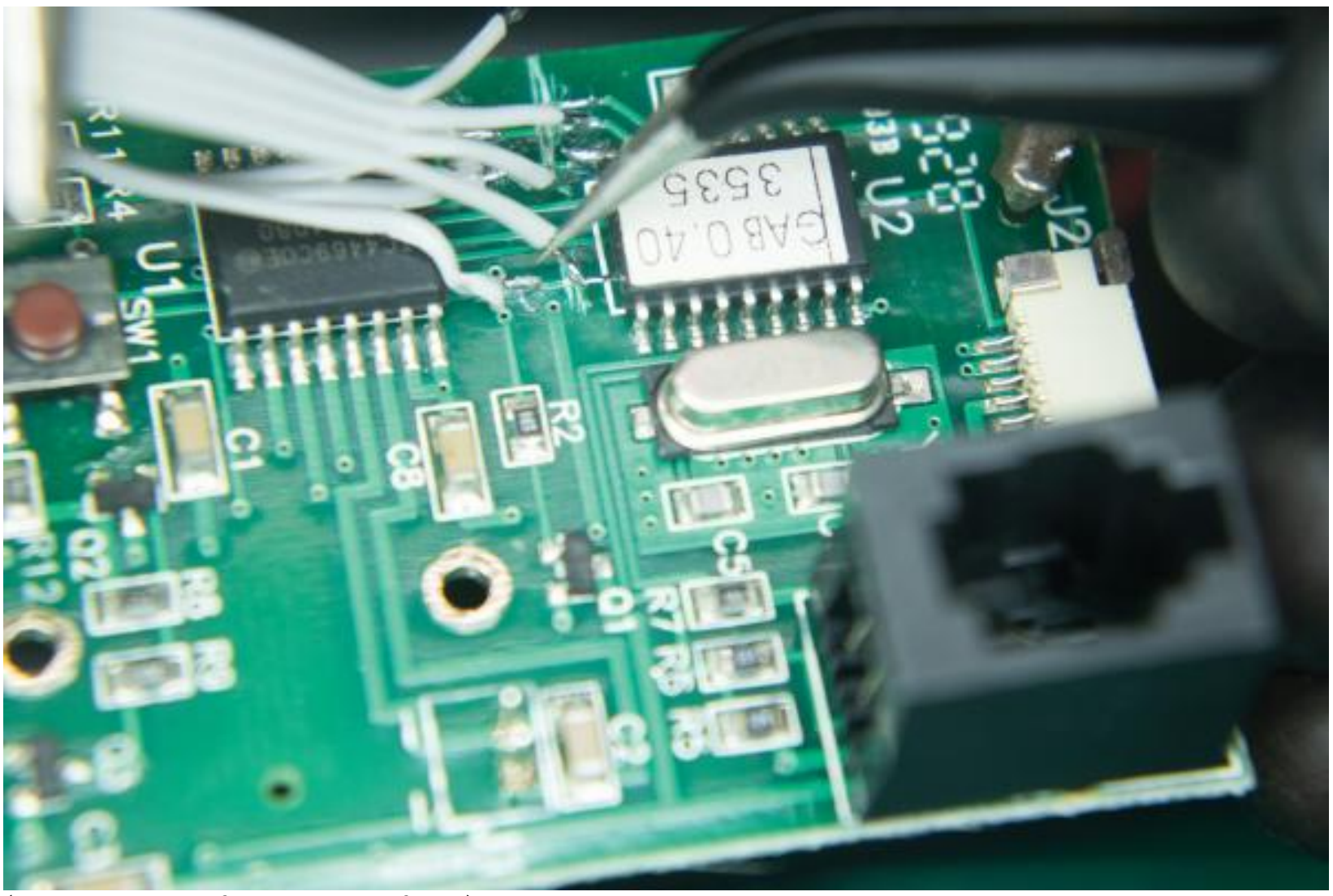
The solution we first came up with to add our own PWM signals was to just solder wires to the MOSFET driver pins, however, this could be damaging to the inbuilt microcontroller. If the microcontroller is outputting a LOW signal to the MOSFET driver, when we inject a HIGH signal, it will short to ground through the microcontroller pin, potentially damaging it. This also occurs in the opposite configuration when the inbuilt microcontroller is sending a HIGH signal and the Arduino is sending a LOW signal.

To fix this issue, we opted to add OR gates inline with each driver pin. That way, if either the Arduino or microcontroller is sending a high signal the OR gate turns on, and the diodes prevent any opposing signal states from shorting to ground. The resistor pulls the signal to ground when no voltage is applied to the OR gate.

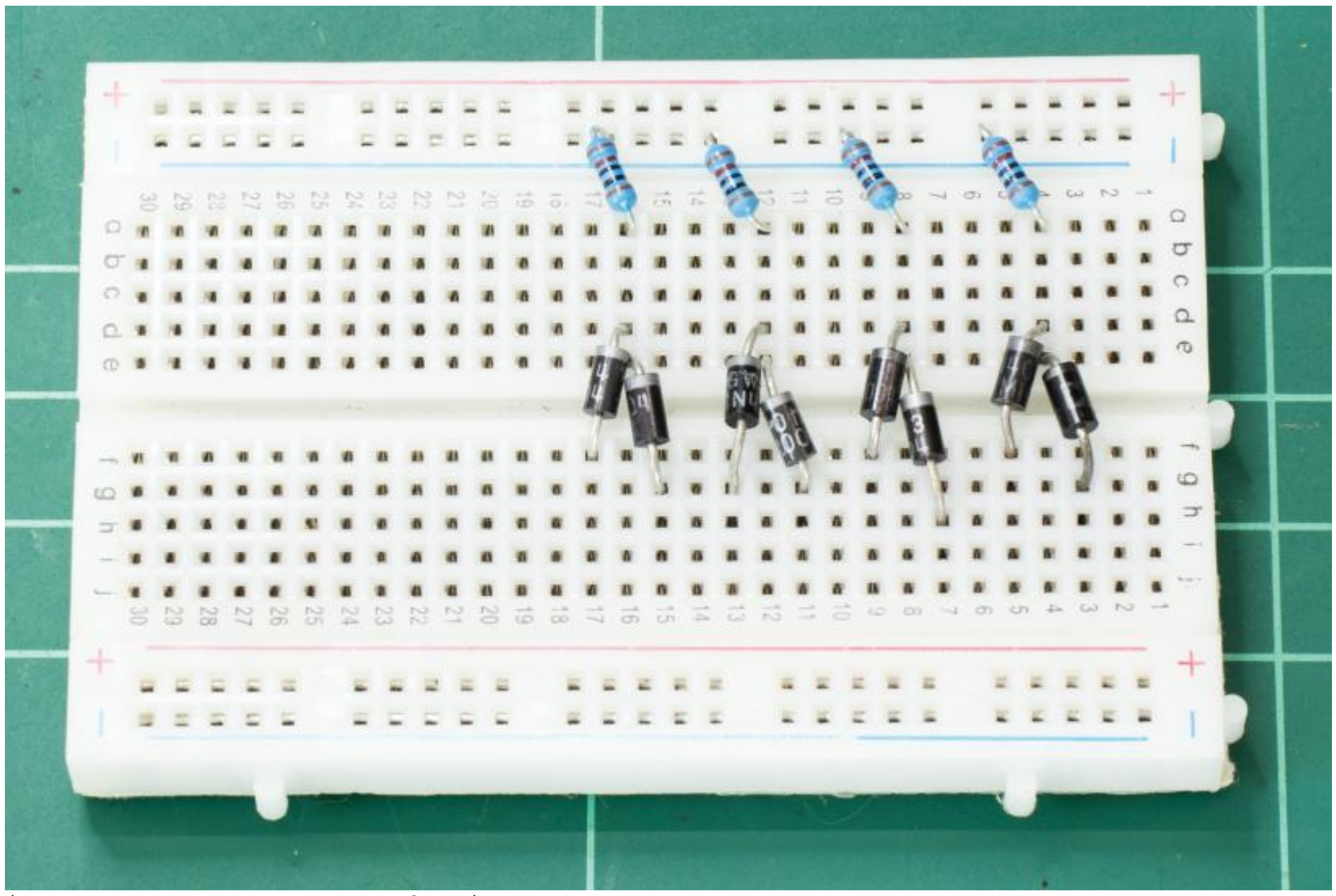
Practically speaking, this requires cutting traces between the microcontroller and MOSFET driver, and breaking out the input and output side to a separate breadboard. Some quite delicate destruction was required with a craft knife to remove small bits of copper and some areas of the solder mask so each of the eight points can be soldered to.



(/_images/617513f0c672e0b43e7fd173)

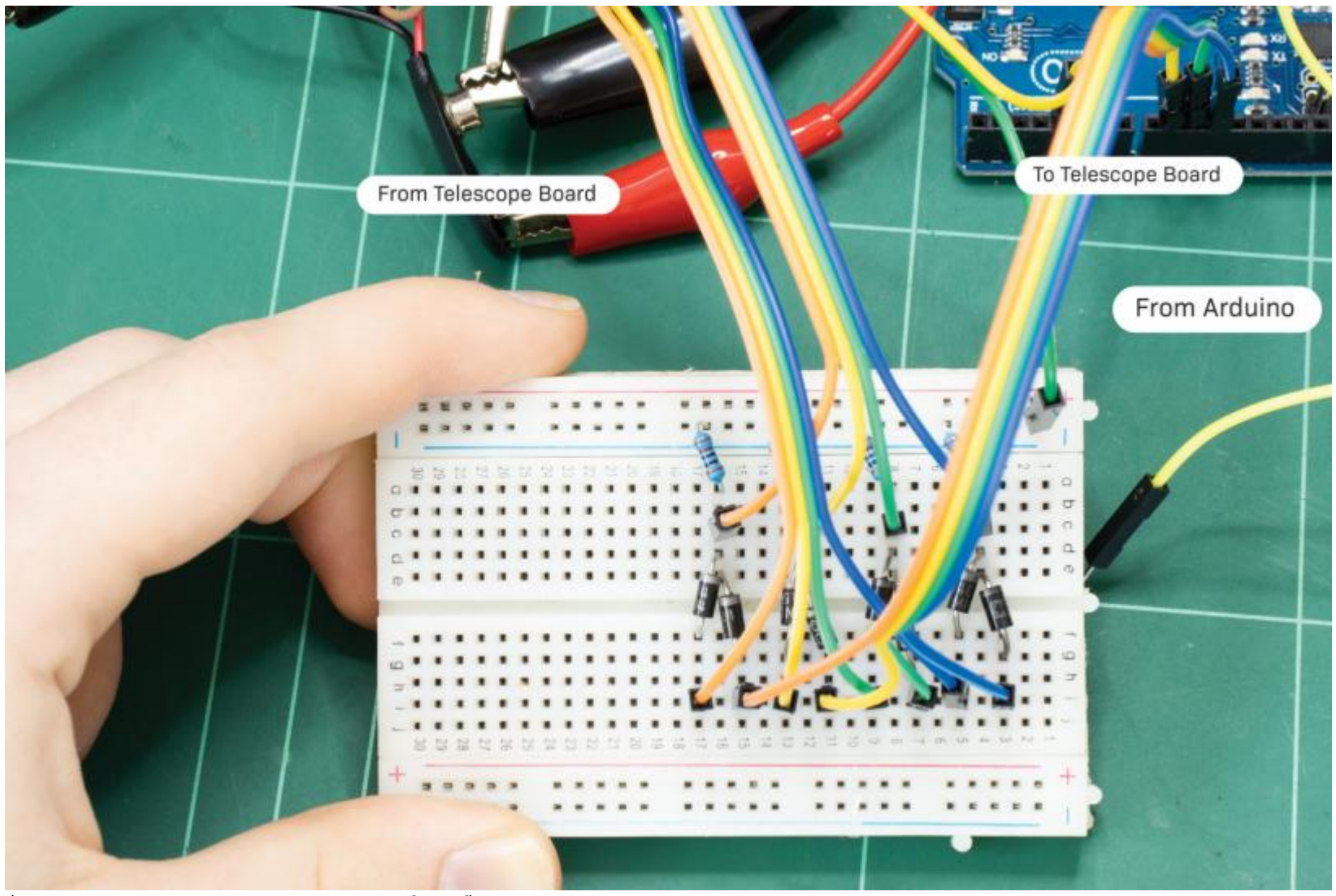


(/_images/617513f0c672e0b43e7fd177)



(/_images/6175140bc672e0de467fd10c)

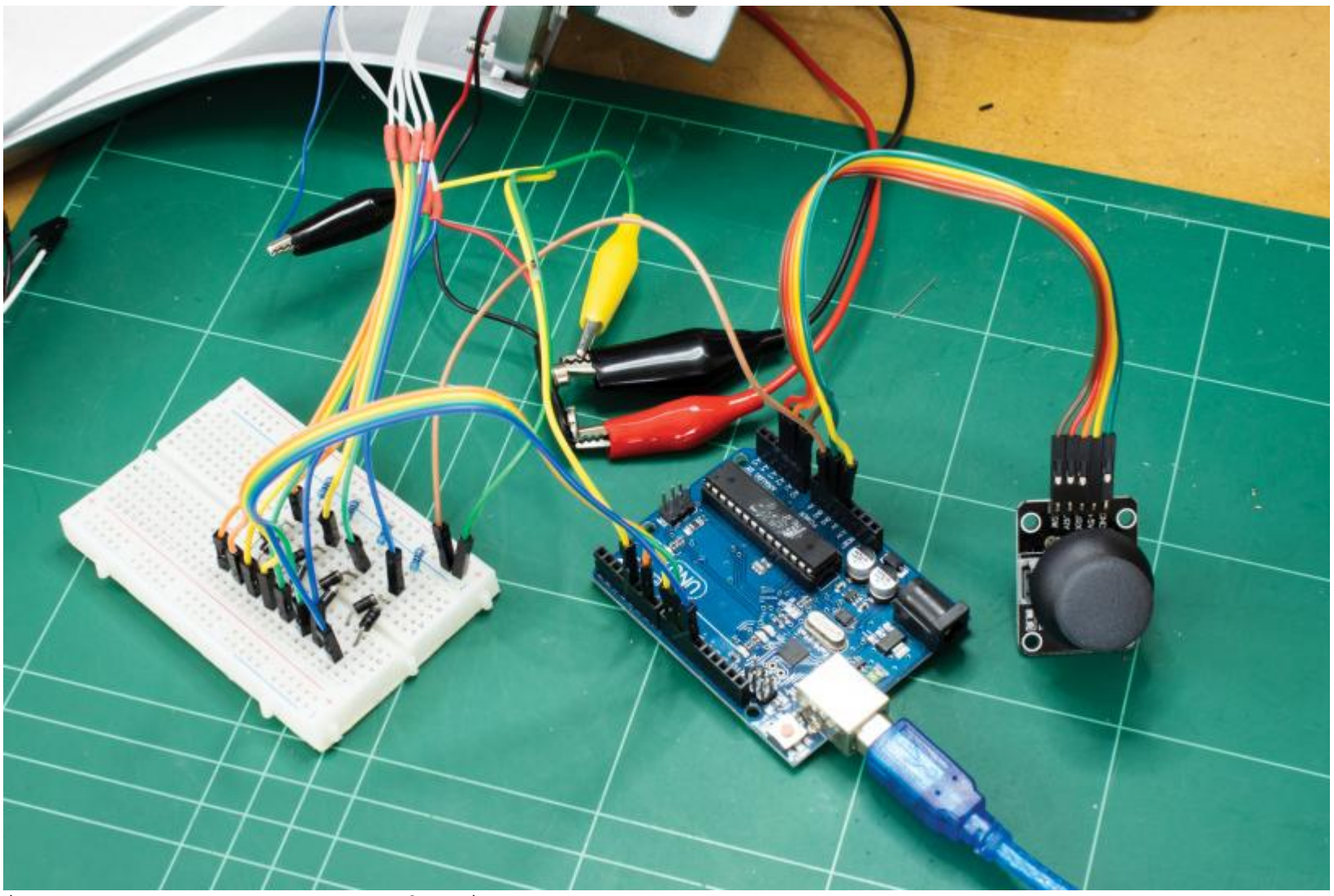
The diodes must be facing towards the resistor, and their anodes joined together on the same breadboard row. Once connected together to form the output, a 10kΩ pulldown resistor is added to ground. This is so that when there is a LOW signal on each data line, the output is pulled to ground instead of floating.



(/_images/6175140bc672e0de467fd10f)

One input of each gate is connected to the direction pins from the telescope microcontroller, and the output on each gate is connected to the trace where the microcontroller used to connect to. The other input is connected to a PWM Digital pin on the Arduino.

For the fundamental build, we're using a simple PlayStation style joystick control. These are handy little control devices that are just two-axis potentiometers with an inbuilt pushbutton. They're great where precise and intuitive analogue control is needed. GND, 5V and two Arduino analogue pins are all that is needed to connect the joystick up, however another digital input can be used if the pushbutton is desired.



(/_images/6175140bc672e0de467fd113)

Whenever adding delicate wires to a circuit board, it's especially worth thinking about strain relief. With tiny solder pads like our modification requires, a small tug is all it takes to rip wires off the board. It could also rip entire traces off the PCB, which would be very annoying. To add a bit of insurance to our soldering work, we wrapped the wires underneath the board, being careful not to go near any hall effect sensors or where the wires could be pinched and shorted.

To complete the wiring for the fundamental build, we simply added an alligator clip to the Serial wire line on the control board, connected the other end to Digital Pin 4 with a male header, and secured the board in place with screws.

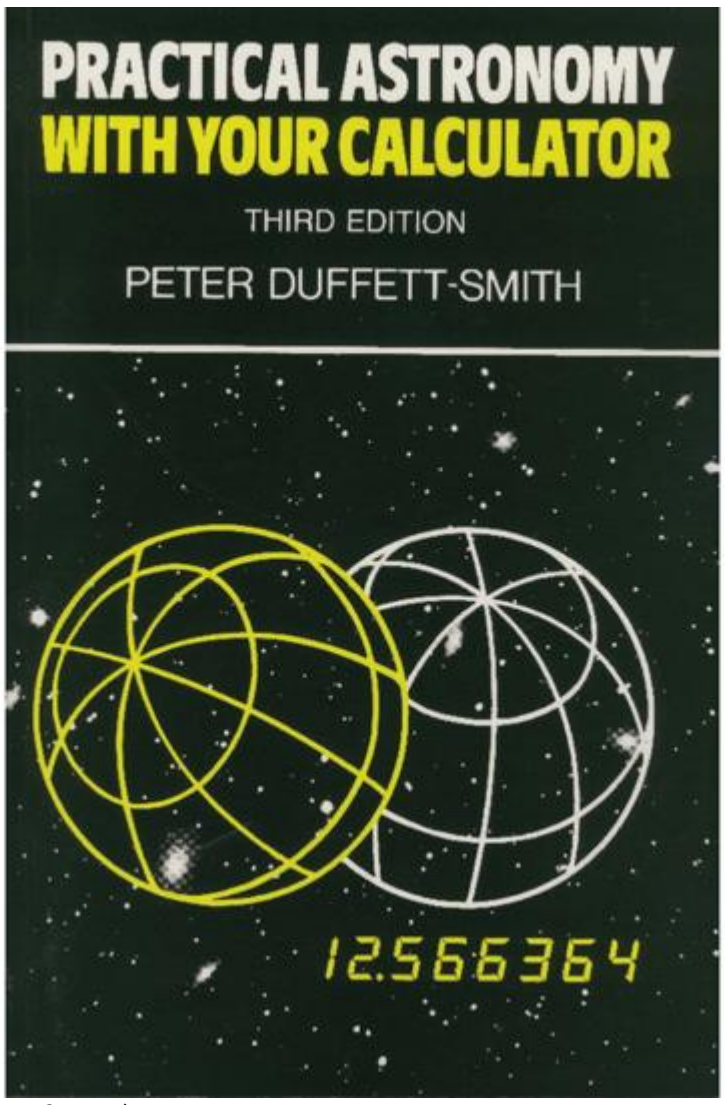
MATHS, MATHS, MATHS

Back in Issue 44, we built a Star Tracker that can move your camera precisely across the sky in accordance with the rotation of the earth, to make long exposures of the stars easy. The hinge system we built was orientated along the rotation of the Earth, so all you need to do is rotate the one motor at a specific rate. This telescope is substantially more complicated to program because it uses a different coordinate system – Azimuth and Altitude. Azimuth is defined as the horizontal angle between North and the direction your telescope is facing, while Altitude is the degrees above the horizon the telescope is facing. In other words, Left, Right, Up and Down – it's essentially a telescope with a turret mount.

Okay, that doesn't seem that complex. Why all the fuss? The main issue is that the position of astronomical objects in the sky change based on time and geographical location. Therefore, we need a coordinate system that remains fixed, regardless of when or where you're observing the stars. Without going into the exact definitions, the standard astronomy coordinate system is "Equatorial Coordinates", which are relative to the equator of the Earth. There is Right Ascension (in hours) and Declination (in degrees). You can think of Right Ascension as how far to look along the equator, while Declination is how far to look towards or away from the poles. Any astronomy book or planetarium program uses these coordinates, so we need to convert them into a set of horizontal coordinates that we can target on the telescope.

The bottom line is that we need to convert the equatorial coordinates – that don't change for the most part – into horizontal coordinates, which do. The best analogy we can think of that's similar to this problem is to imagine designing a mount for a gigantic spotlight that aims at an airplane moving across the night sky, lighting it up. Even if the airplane has a simple, straight path across the sky, there will still be a substantial amount of mathematics to calculate the angle at which the mount needs to face in both the Altitude and Azimuth directions at a given time.

We weren't going to derive the maths for this, so we managed to track down the book "Practical Astronomy with Your Calculator Or Spreadsheet" by Peter Duffet-Smith for some more info. The first thing we need to do is convert local time (the time on your watch or computer) to sidereal time. A sidereal day is how long an object in the sky takes to returns to that same position, which is 23 hours and 56 minutes later. Because the Earth has its own orbit around the Sun, one solar (regular) day is actually when the earth rotates approximately 361 degrees relative to the stars!



(/_images/6175140bc672e0de467fd117)

Despite the original edition being over three decades old, it's a goldmine of mathematics for a project like this.

Once we have sidereal time, we can convert it to an 'Hour-Angle', which is the time when the object we want to point at passes the meridian over our observation location. You can picture the meridian as a line stretching from North to South in the sky.

$$H = \mathrm{LST} - \alpha$$

(/_images/6175140bc672e0de467fd11a)

Where H is the hour angle, α is the target right ascension (hours), and LST is local sidereal time.

We promise, this is the last step! Finally, we can convert the Hour Angle and the declination from the target to our Azimuth and Altitude Target:

$$\sin a = \sin\delta \sin\phi + \cos\delta \cos\phi \cos H$$

$$\cos A = \frac{\sin\delta - \sin\phi \sin a}{\cos\phi \cos a}$$

(/_images/6175140bc672e0de467fd11c)

Where δ is Declination, Φ is our geographic latitude (e.g. -33.8 for Sydney), and H is the hour angle

From this result, we can calculate the inverse functions of each to get 'a' (altitude) and 'A' (azimuth). There are some extra trigonometry tricks we also need to implement to avoid ambiguous solutions, but these (and the rest of this maths implementation) can be found in the project code as they are out of the scope of this explanation.

SLEWING

Slewing is the process of moving the telescope to a set of provided coordinates. Sounds simple, but this ended up being one of the most complex parts of this project, even disregarding the maths! Slewing a telescope is an example of a 'closed-loop control system', where we make an action, observe the reaction of the system, and then assess whether we've made any progress towards the target coordinates.

A simple approach would be to run each motor at a set speed depending on which direction we need to move. If the coordinates are above and to the right, we run the UP and RIGHT MOSFET drivers until we've reached them. The problem with this is that if we run the motors at full speed to make slewing faster (since the telescope takes 30 seconds to a minute to turn 180 degrees), when we eventually do reach the target, we will overshoot and have to go the other way. If we run the motors too slow, we will either stall the motors from a lack of current or dinosaurs will re-evolve before we reach the target.

Speaking of which, what is 'reaching the target'? Just like how you can't go exactly 60.000km/h in your car, we can't exactly reach our target either. There will always be some error with a continuous system. We need to add tolerances to our control system so it can basically say 'Okay, that's good enough'. A tolerance too wide will mean slewing isn't accurate, and a tolerance too small will cause the telescope to oscillate between either side of the target.

CONTROL CODE

We're not going to sugarcoat it, there is a significant amount of code and maths responsible for keeping our telescope in control. It's not hard to accidentally let the telescope run into itself or extend past limits, so safeguards need to be put in place to ensure that doesn't happen. This code is used for both the fundamental and main builds, with some minor changes to accommodate the LED ring and the Wii Nunchuk controller.

The code shown here (taken from the loop function) is responsible for most of the positional control. You can see there are a couple of filters we apply to the joystick to make it easier to use and to prevent accidental movements. The slewing and tracking code essentially allow us to cancel a movement with the joystick, which is useful if the telescope decides to misbehave and tries to run into itself. Finally, we need to convert coordinates received from a computer (if one is present) to Horizontal coordinates, and the current coordinates of the scope back to Equatorial coordinates to send it back.

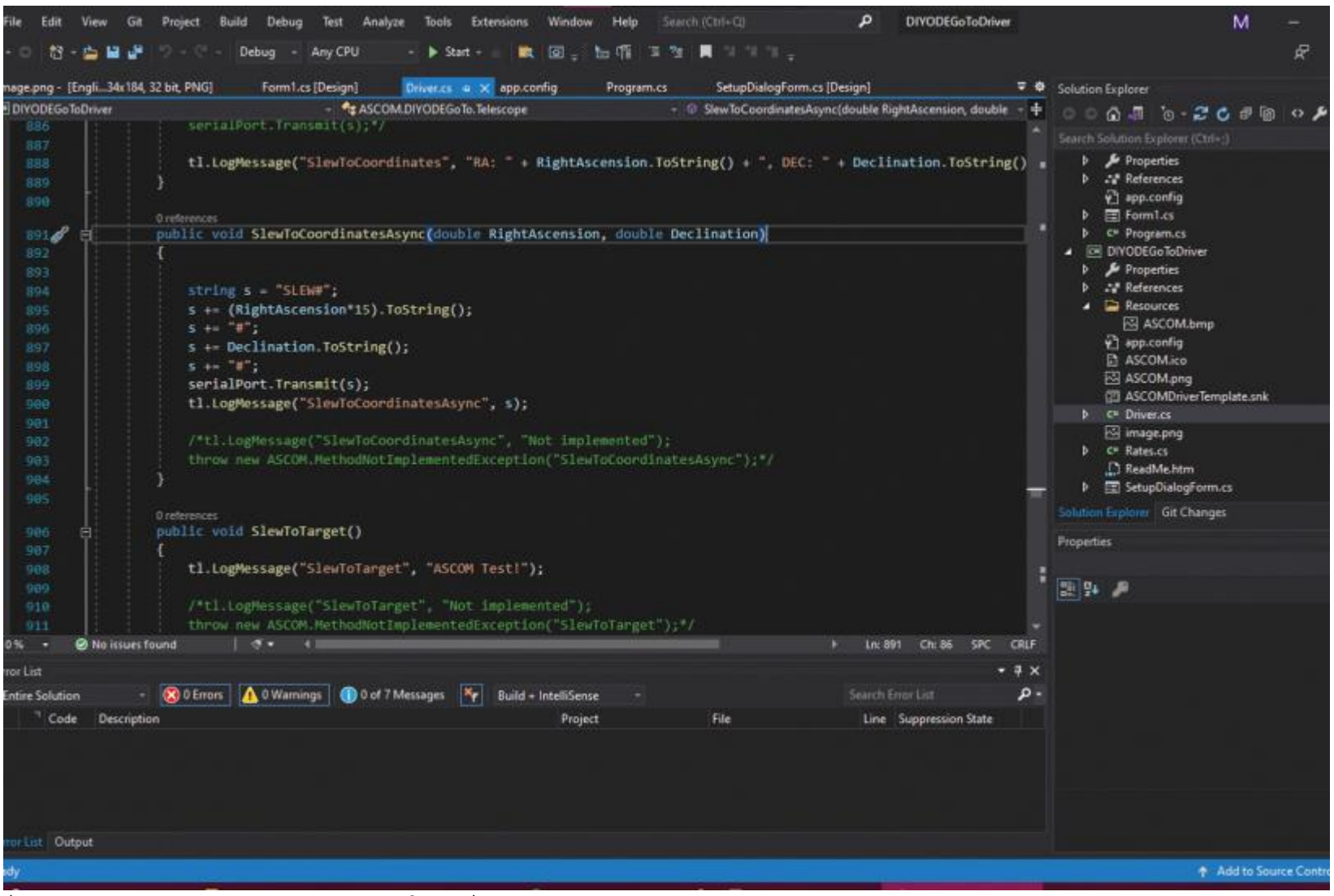
```
float jx = analogRead(JOYSTICK_X);
float x_speed = mapfloat(jx, 0, 1023, -1.0, 1.0);
x_speed = abs(x_speed) > 0.1 ? x_speed : 0;
float jy = analogRead(JOYSTICK_Y);
float y_speed = mapfloat(jy, 0, 1023, -1.0, 1.0);
y_speed  = abs(y_speed) > 0.1 ? y_speed : 0;
if(slewing && (x_speed != 0 || y_speed != 0)) {
  abortSlew();
}
//If we move the joystick, cancel slewing and
// move in joystick direction.
if(!slewing) {
  setAzRate(x_speed);
  setAltRate(y_speed);
}
if(slewing) {
  handleSlew();
}
convertEqToHor(&tar_alt, &tar_az, tar_ra, tar_dec);
convertHorToEq(&cur_ra, &cur_dec, cur_alt +
offset_alt, cur_az + offset_az);
if(!slewing && tracking) {
  handleTracking();
}
```

ASCOM

ASCOM is a very popular astronomy control system that allows hardware to interface with software in a universal way. Because our Arduino doesn't know how to talk to any external software, we need to program it with some Serial commands so it can communicate information to a custom ASCOM driver.

(/_images/6175140bc672e0de467fd120)

The ASCOM driver essentially acts as an interface layer between planetarium or astronomy software and your telescope. ASCOM also provides functionality to make your own telescope focuser, dome control (for observatories) and filter wheels for cameras. Microsoft Visual Studio is required for this, and once we installed the ASCOM developer pack, we started an empty Telescope Driver ASCOM project. The base code provides a large selection of custom functionality that can be implemented for use with telescopes. Everything from basic position reporting to automatic slewing can be done through the ASCOM driver!



(/_images/6175140bc672e0de467fd123)

Writing an ASCOM driver for such a cheap telescope like this is probably overkill. Saying that, the ability to control a frustrating system with software capable of accurate pointing precision is a great advantage.

When a blank telescope driver is created, it includes a template Driver.cs file that has many unimplemented methods. It's up to you to configure how this code is implemented, including opening, controlling and closing a serial connection between the Arduino and your driver. The most basic are the "Get" methods for Right Ascension and Declination. These fields are accessed whenever Stellarium or a ASCOM compatible software requests the current telescope position. So, we added a very simple Serial connection that just sends "RA#" or "DEC#" to the Arduino when the current position is desired. The hash acts as delimiter, meaning that during processing, we can stop reading the data when we see it.

```
public double Declination
  {
    get
      {
        serialPort.Transmit("DEC#");
        string s;
        s = serialPort.ReceiveTerminated("#");
        s = s.Replace("#", "");
        tl.LogMessage("Serial", s);
        double result = double.Parse(s);
        return result;
      }
  }
```

We also need to implement the AsyncCoordinates and the Slew methods, which are called whenever the user clicks on a "Sync" or "Slew" button respectively. In both cases, we need to transmit a set of coordinates to the Arduino. These are transmitted in this format:

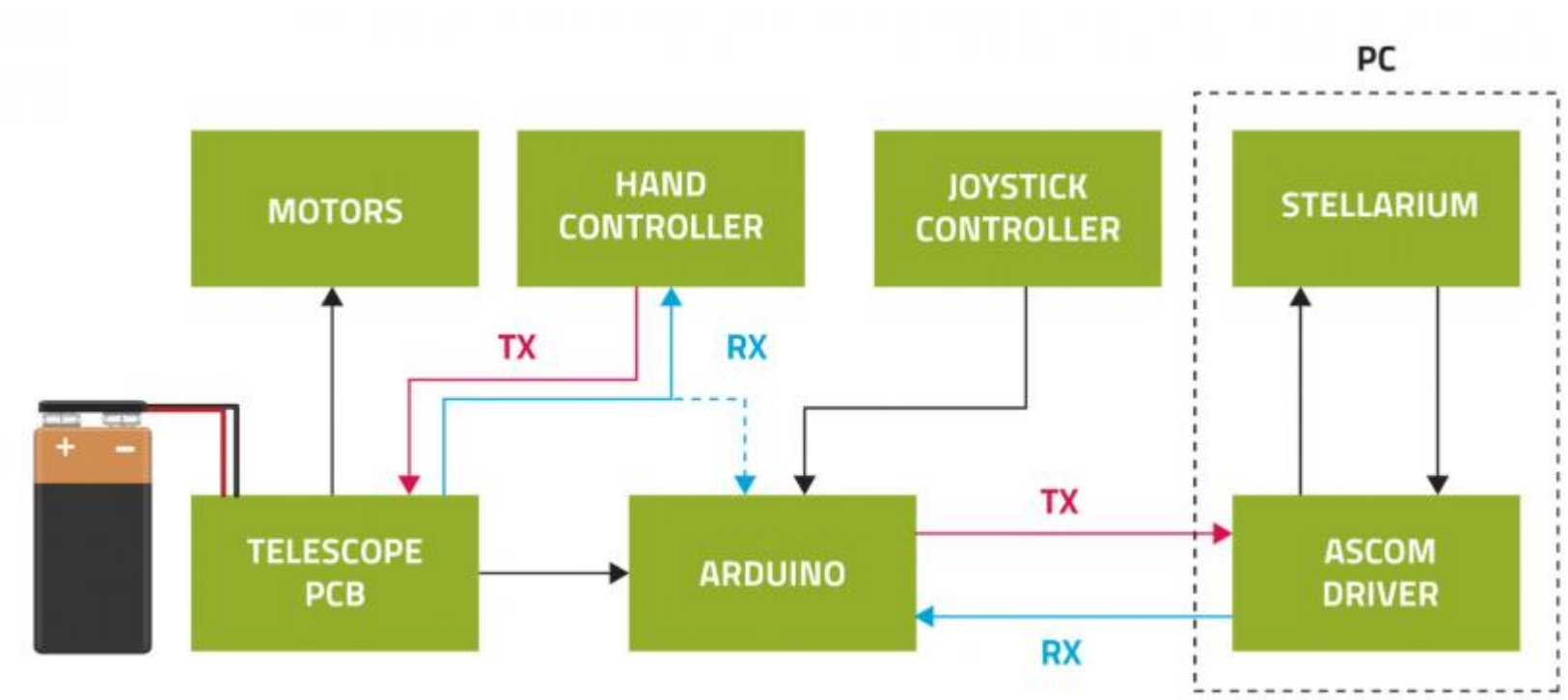
SLEW#13.33#21.88#

SYNC#110.98#54.86#

Of course, these exact formats of communication methods don't have to be followed. You could use '$' for a delimiter, you could use "TARGET" instead of "SLEW". Because we're implementing software on both the driver and Arduino side, we can use whatever communication method we want. This goes for any project where communication can be controlled from both ends.

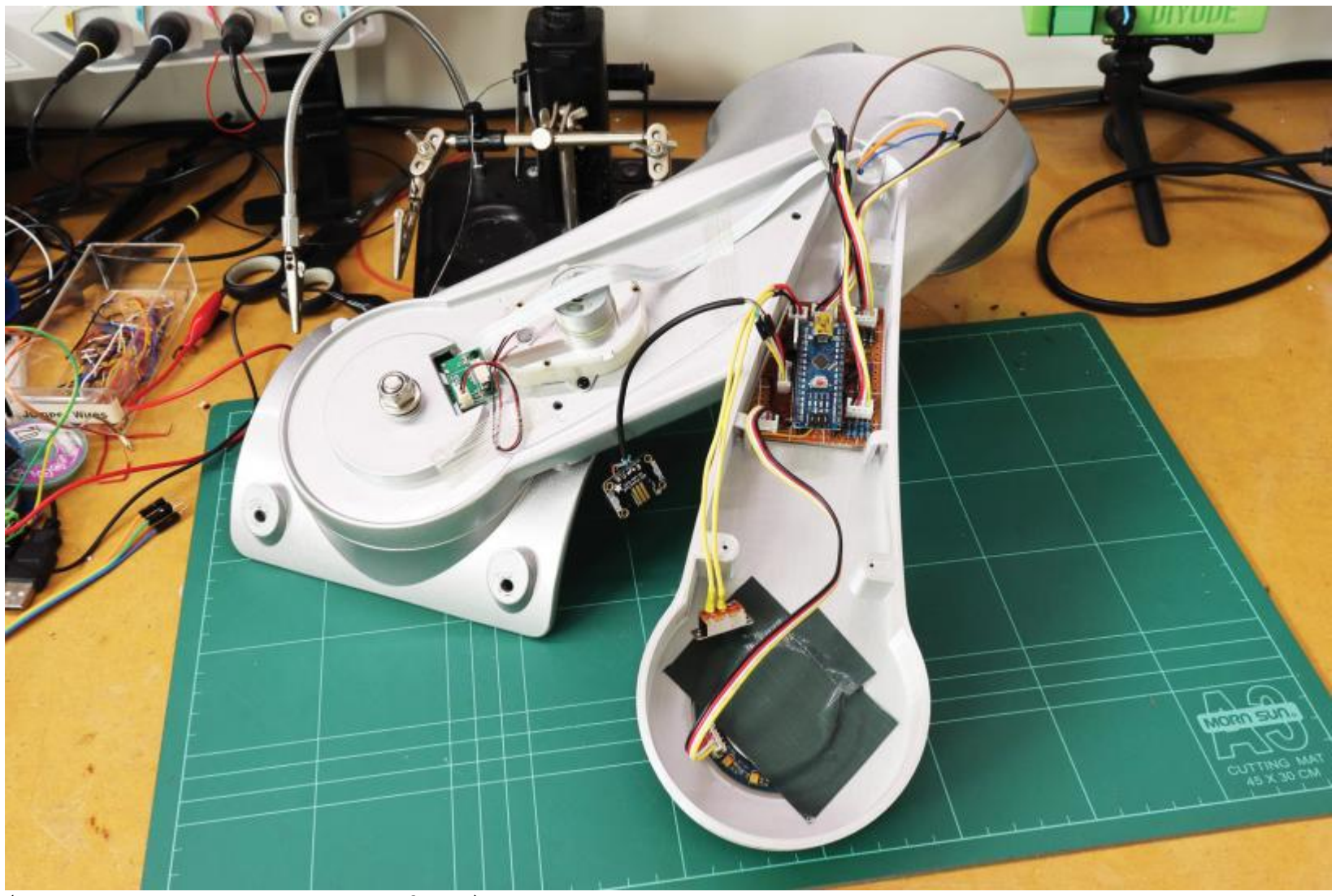
(/_images/6175140bc672e0de467fd11e)

Each ASCOM template driver also includes a Window form file that can be modified to make your own configuration UI. We added a DIYODE logo to our driver and rearranged some UI to make it easier to use.



(/_images/6175143ac672e0533e7fd132)

To finish the driver, simply compile and build it (With Ctrl+Shift+B). Note that Visual Studio must be ran as an Administrator for this to work. If ASCOM is installed on your system, the driver should now be accessible to any astronomy software suites! (Make sure to select the appropriate COM port that the Arduino is connected to.)



(/_images/6175143ac672e0533e7fd134)

MAIN BUILD:

PARTS REQUIRED:

Jaycar | Altronics | Pakronics

1 x Arduino Nano or Compatible	XC4414 (https://www.jaycar.com.au/p/XC4414)
8 x 1N4004 Diodes or equivalent	ZR1004 (https://www.jaycar.com.au/p/ZR1004)
4 x 10kΩ Resistors*	RR0596 (https://www.jaycar.com.au/p/RR0596)
5 x Female Grove Connectors	-
1 x 5V 7805 Regulator	ZV1505 (https://www.jaycar.com.au/p/ZV1505)
1 x 40-pin Female Header Strip	HM3230 (https://www.jaycar.com.au/p/HM3230)
1 x Mini RGB NeoPixel Ring	-
1 x Wii Nunchuk (or compatible)	-
1 x Wii Nunchuk Breakout Adapter	-
1 x Mini USB Cable	WC7792 (https://www.jaycar.com.au/p/WC7792)
1 x JST XH Connector Pair*	PT4457 (https://www.jaycar.com.au/p/PT4457)
5 x Male-to-Male Grove Wires#	XC9192 (https://www.jaycar.com.au/p/XC9192)
Heatshrink	WH5668 (https://www.jaycar.com.au/p/WH5668)
Solid Core Wires	-

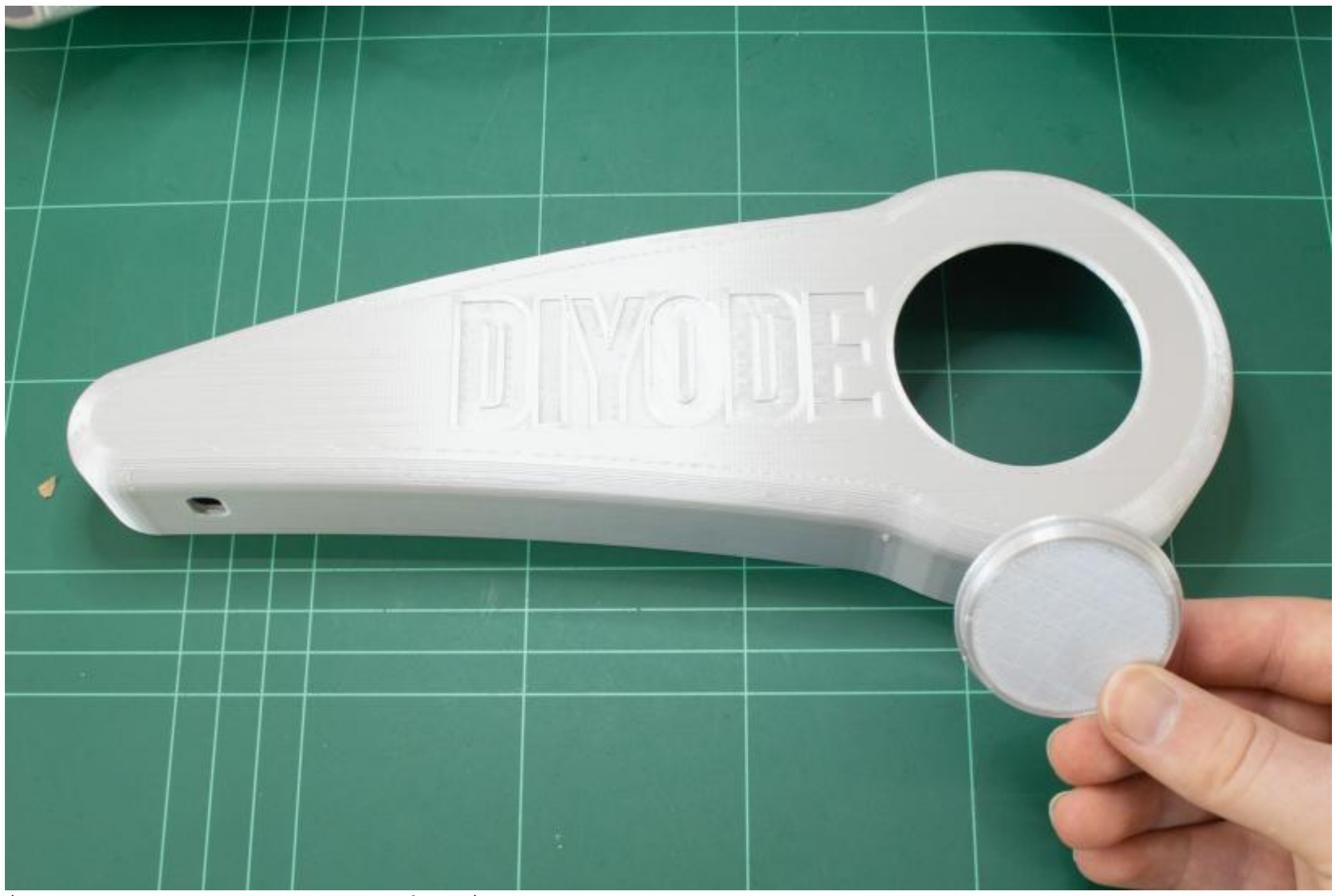
* Quantity required, may only be sold in packs. # We chopped these in half to make connection to other solder points in the circuit easy.

After tinkering about with the telescope for a while and getting a good code system working, we're now ready to build the system into a neat package. Wires hanging out everywhere is handy for prototyping, but less handy when it becomes a night-time tripping hazard and an absolute eyesore.

3D PRINTING

We're 3D printing some parts for our telescope to clean everything up and provide a nice finish to the parts.

Since we need to keep the hand controller connected, we can't design the electronics to fit in its place. Therefore, we're mounting the Arduino circuitry onto the side of the telescope, and 3D printing a new bracket to cover all of the electronics.



(/_images/6175143ac672e0533e7fd138)

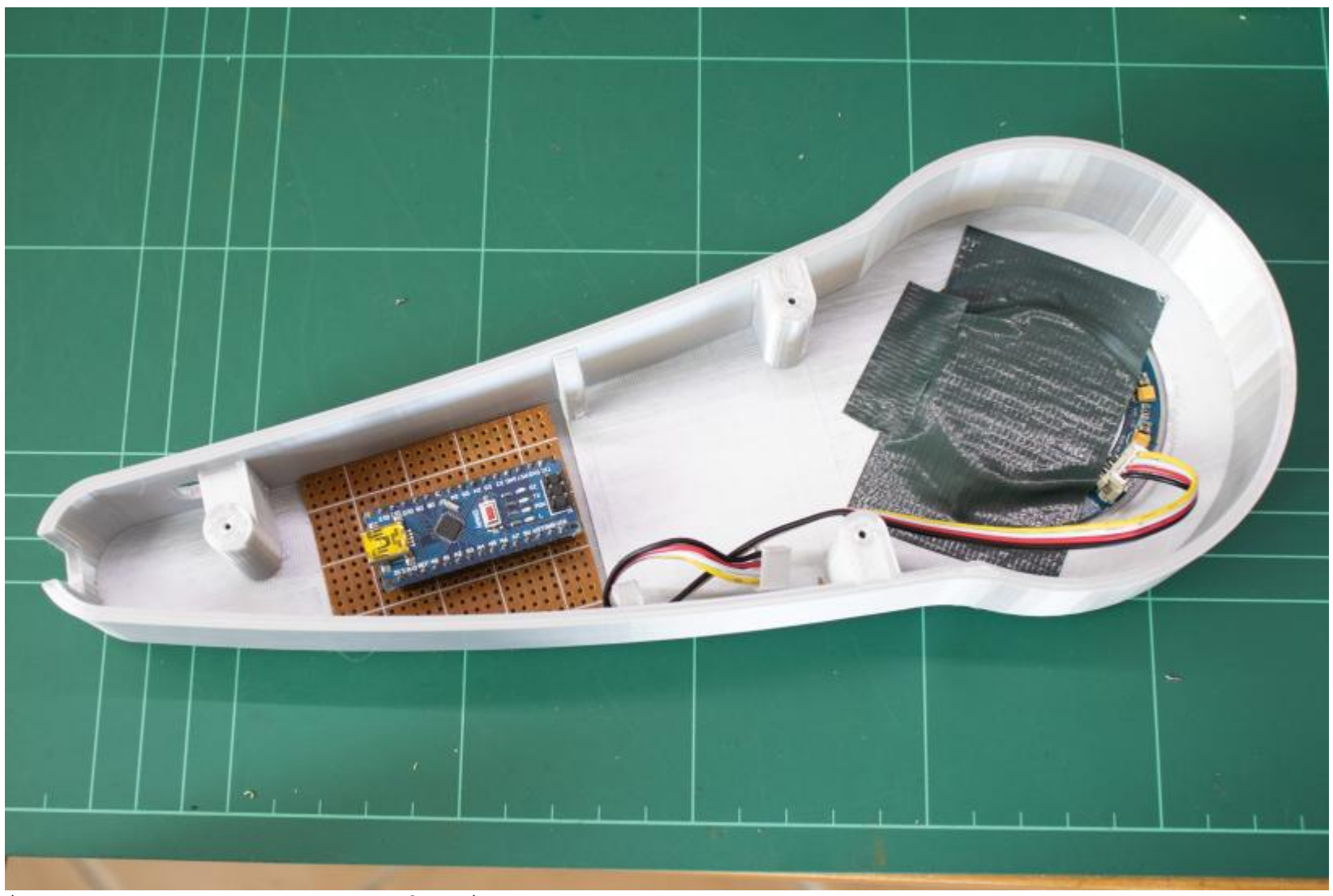
The bracket also has a centre disc that covers up the hole made by the RGB LED ring. Obviously, these prints will only suit this series of Bushnell telescopes, but we've provided the STLs for those who want to check it out.

ARDUINO CONTROL BOARD

Our main build will use an Arduino compatible Nano to control the slewing, movement functionality and the Nintendo Nunchuk inputs. We're also adding a NeoPixel LED ring to the telescope both for some added aesthetic effects and to provide a visual indicator of the current mode the telescope is in.

We snapped off a trapezium-shaped section of perfboard, which needs to be cut to size to fit our 3D printed chassis. Most external connections from our board will use the handy 4-pin Grove connectors. They are better than standard pin headers as they are non-reversible (so incorrect connection is not possible), are more durable and don't suffer from friction-fit problems. In past projects such as the Nano Vending Machine, we often found that regular pin headers can be unreliable, leading to disconnection problems once the whole machine is assembled.

We placed our NeoPixel ring into the 3D printed enclosure and secured it with duct tape, ensuring that the boards fit well.



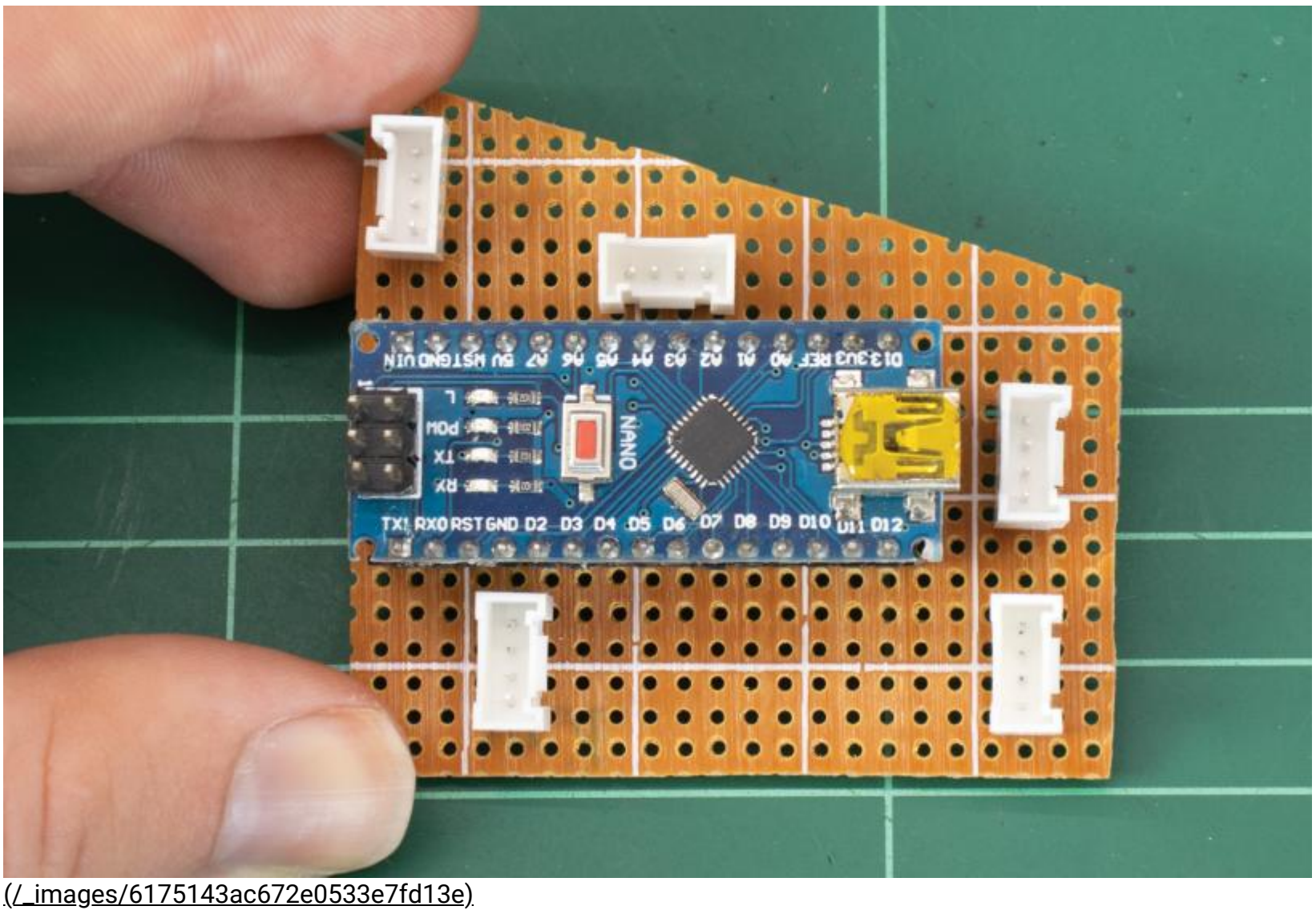
(/_images/6175143ac672e0533e7fd13b)

We have five grove connectors on our board:

1. 9V Power, Serial Data In and Wake Up signal for Hand Controller
2. PWM Motor Control In
3. PWM Motor Control Out

4. NeoPixel Ring
5. Wii Nunchuk

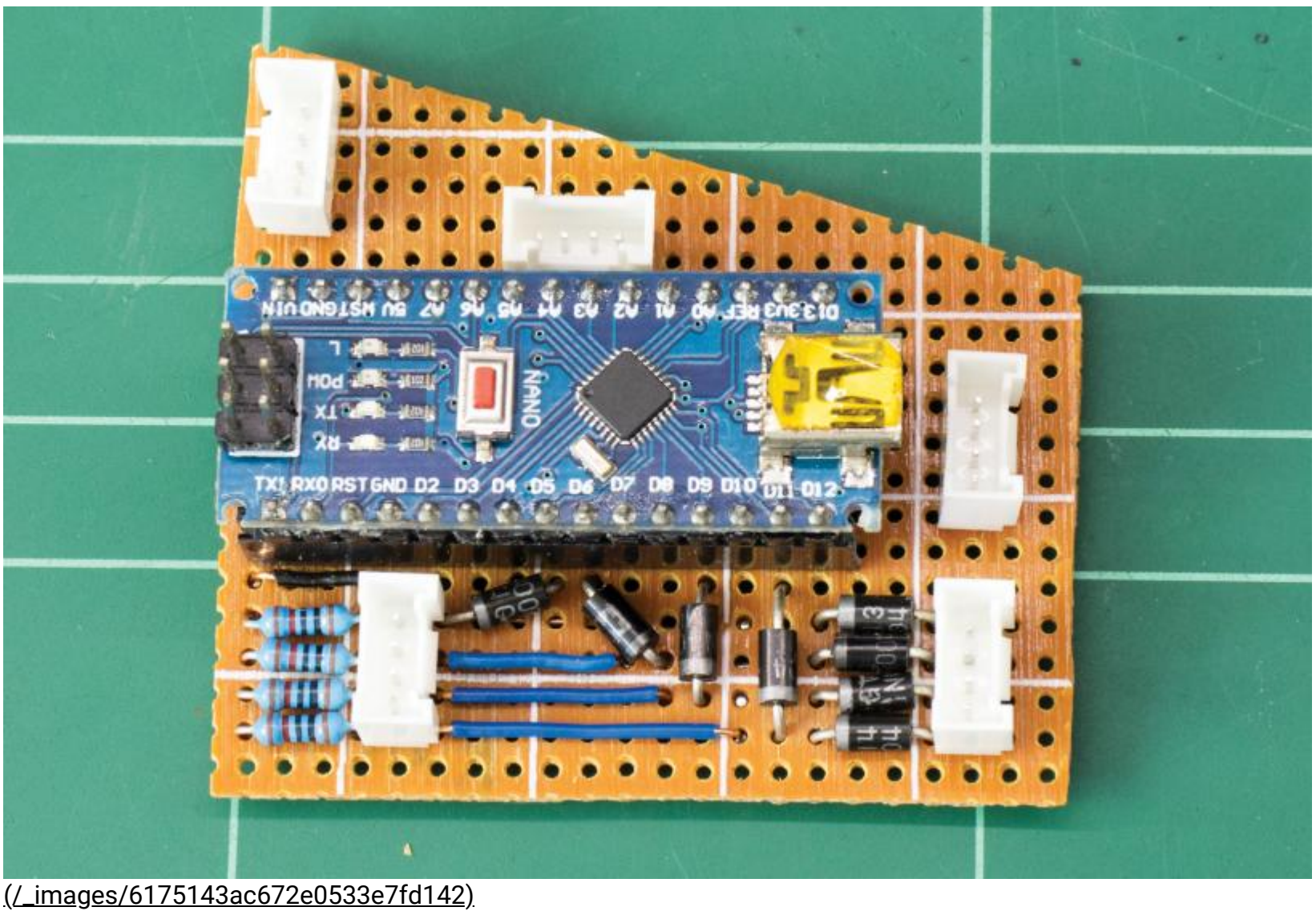
These headers were added in addition to the pin headers for the Arduino Nano. We will also have a 2-pin JST XH connector that goes to a switch, to turn on or off the 7805 regulator added later on.



(/_images/6175143ac672e0533e7fd13e)

Next up, we soldered in the four OR gates on the control board to combine the original motor PWM signals with those from the Arduino. As we saw in the Fundamental Build, each is built with two diodes from each input and a pulldown resistor. We built our control board pretty compact, so watch out and make sure no diodes are connected in reverse or to the wrong gates.

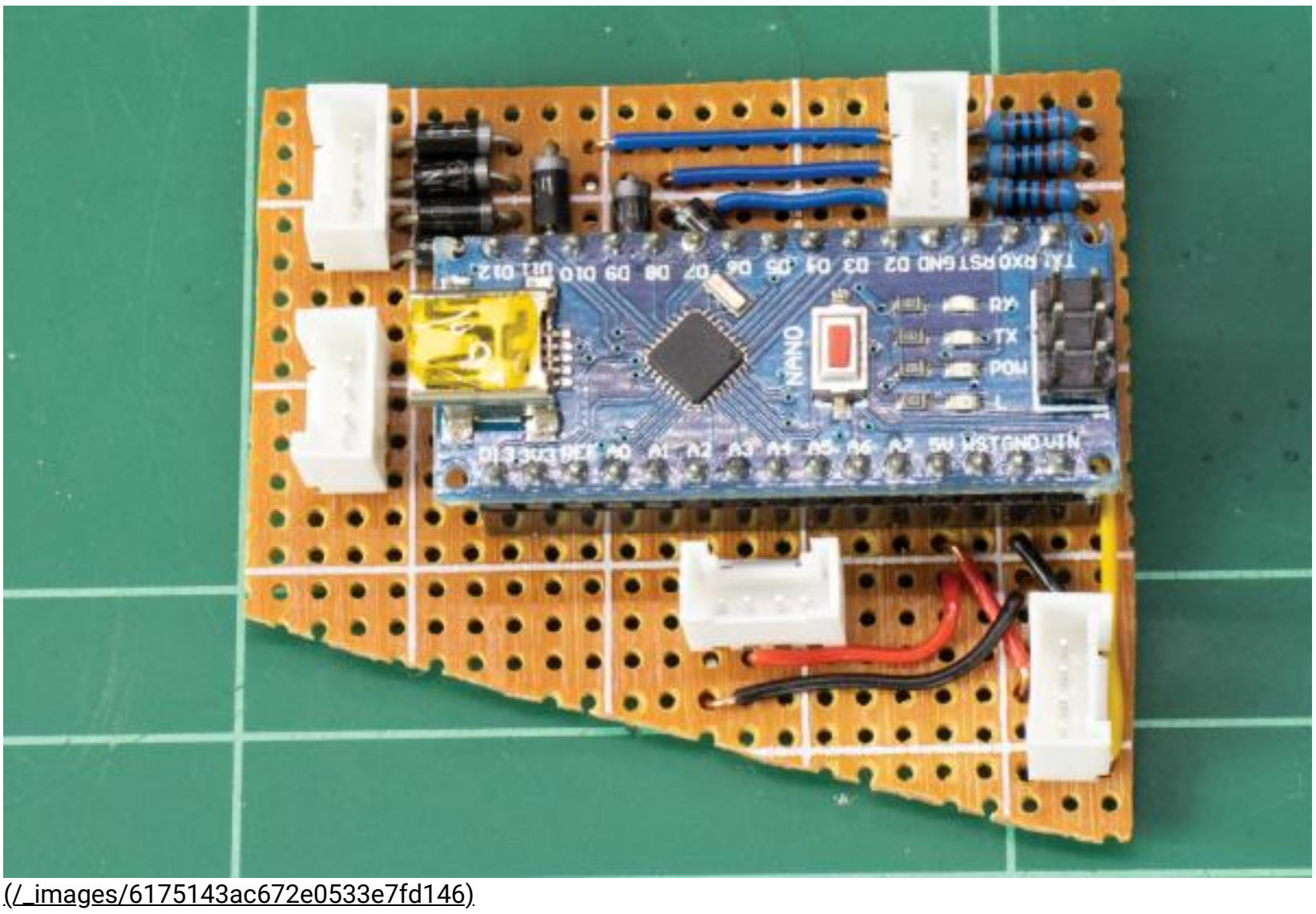
Each of the four input lines go into their own OR gate, with the other input of each coming from one PWM output of the Arduino. The signal coming out of the OR gates are fed back into a Grove connector to be used as the output.



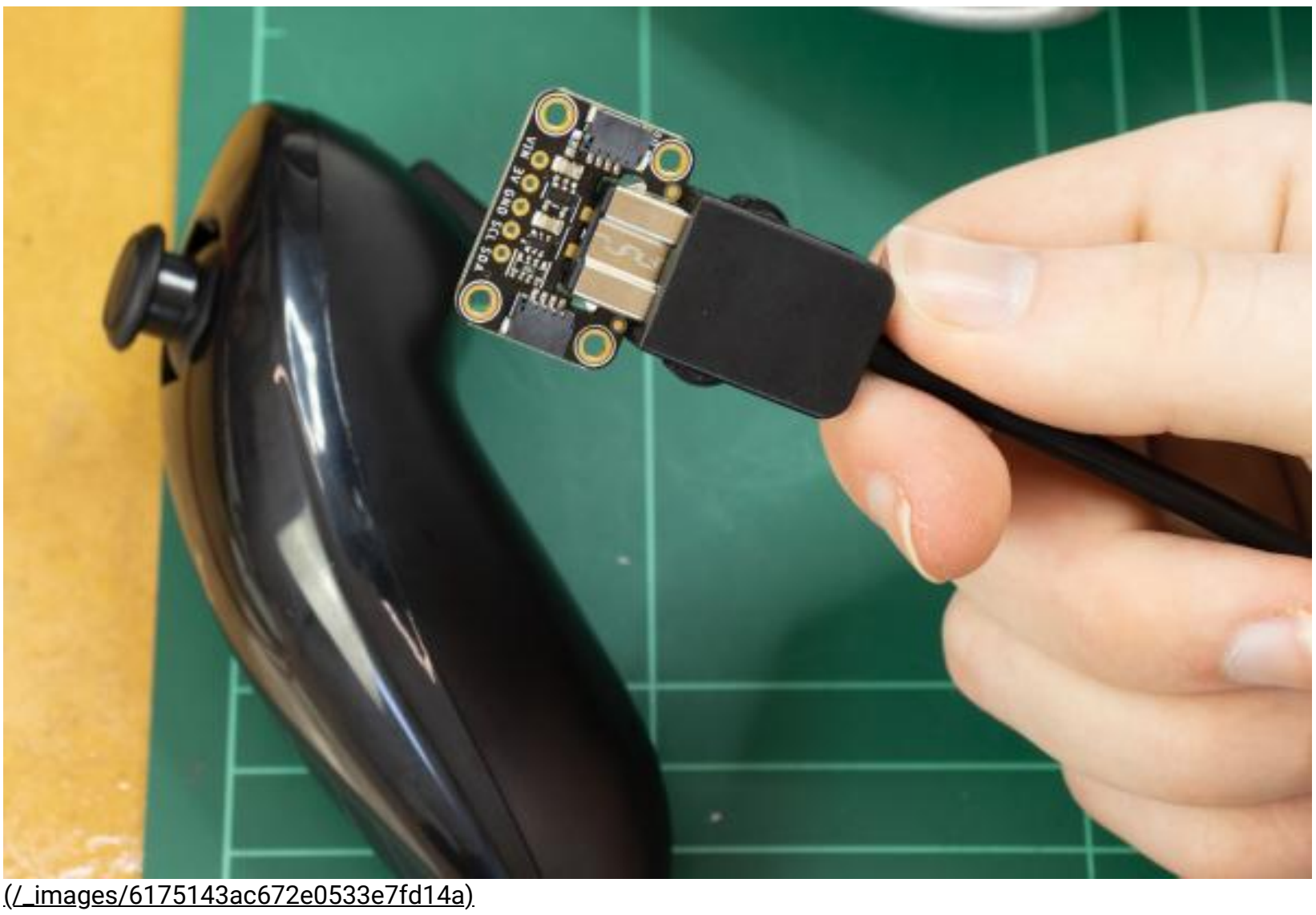
(/_images/6175143ac672e0533e7fd142)

We then sorted out the NeoPixel ring and Wii Nunchuk controller. Yes, you read that right! We're going to be using a Wii Nunchuk for controlling the final build of our telescope. While it may seem weird to be using a controller from a game console released just over 15 years ago, it turns out Nintendo added some pretty cool features to the humble Nunchuk. Not only does it have a two-axis joystick and two buttons, but it also includes an accelerometer and gyroscope for access to movement data. Considering it can be used with one hand, it's a great candidate for our telescope controller! It's also super easy to access over an I2C bus with an Arduino and grab its current data. The applications for this handy little unit stretch far beyond telescopes, we can imagine tons of uses for the Wii Nunchuk for DIY electronics projects.

The Joystick will control the X-Y movement of the telescope in the Azimuth and Altitude directions, while the other two buttons will be used for tracking and slewing. The Z (lower) button will be used for slewing to the target object, set either by holding the button for 2 seconds or through the planetarium software. The C (upper) button will be used for enabling or disabling automatic object tracking, since in experimentation we sometimes found the automatic tracking distracting when observing.



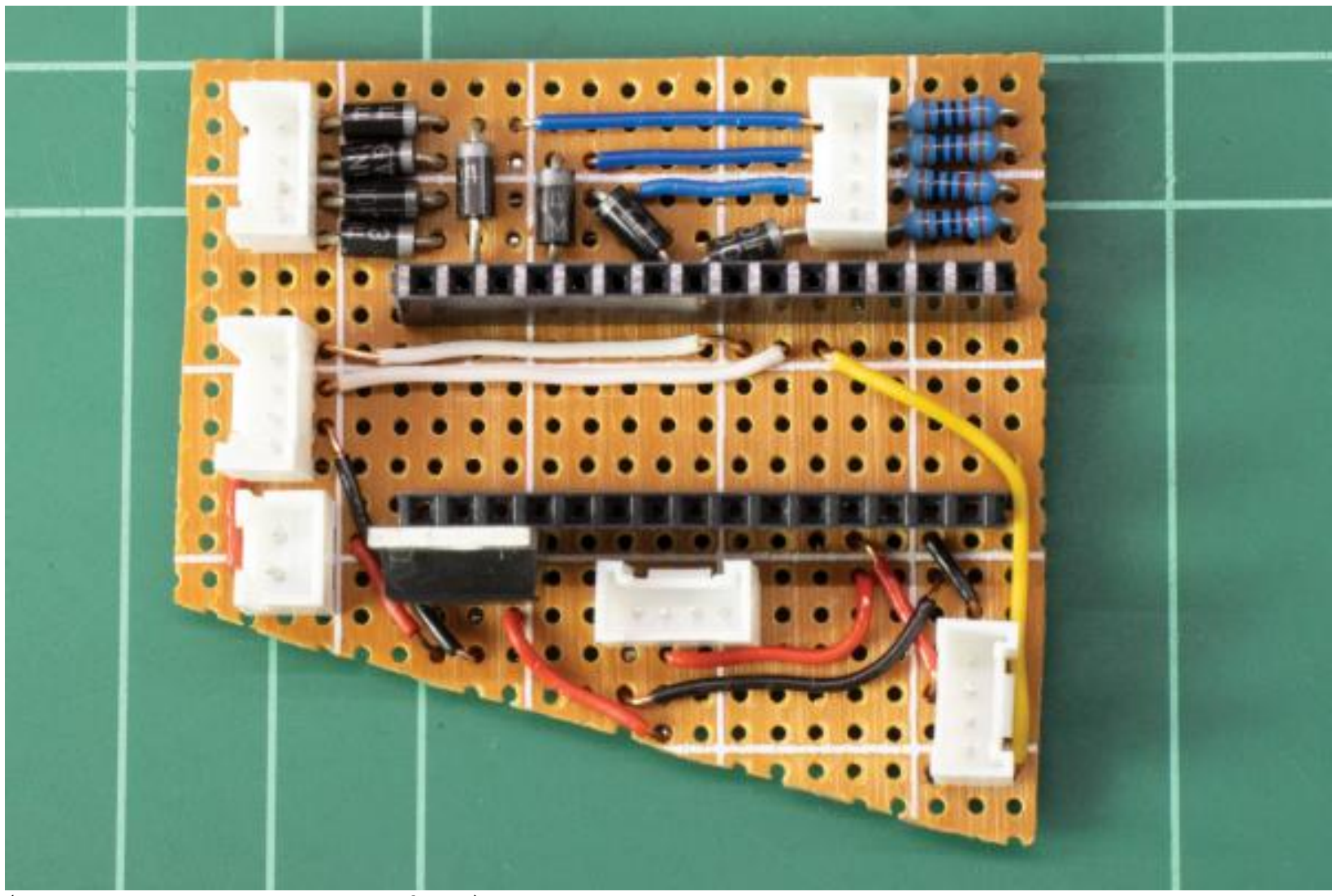
(/_images/6175143ac672e0533e7fd146)



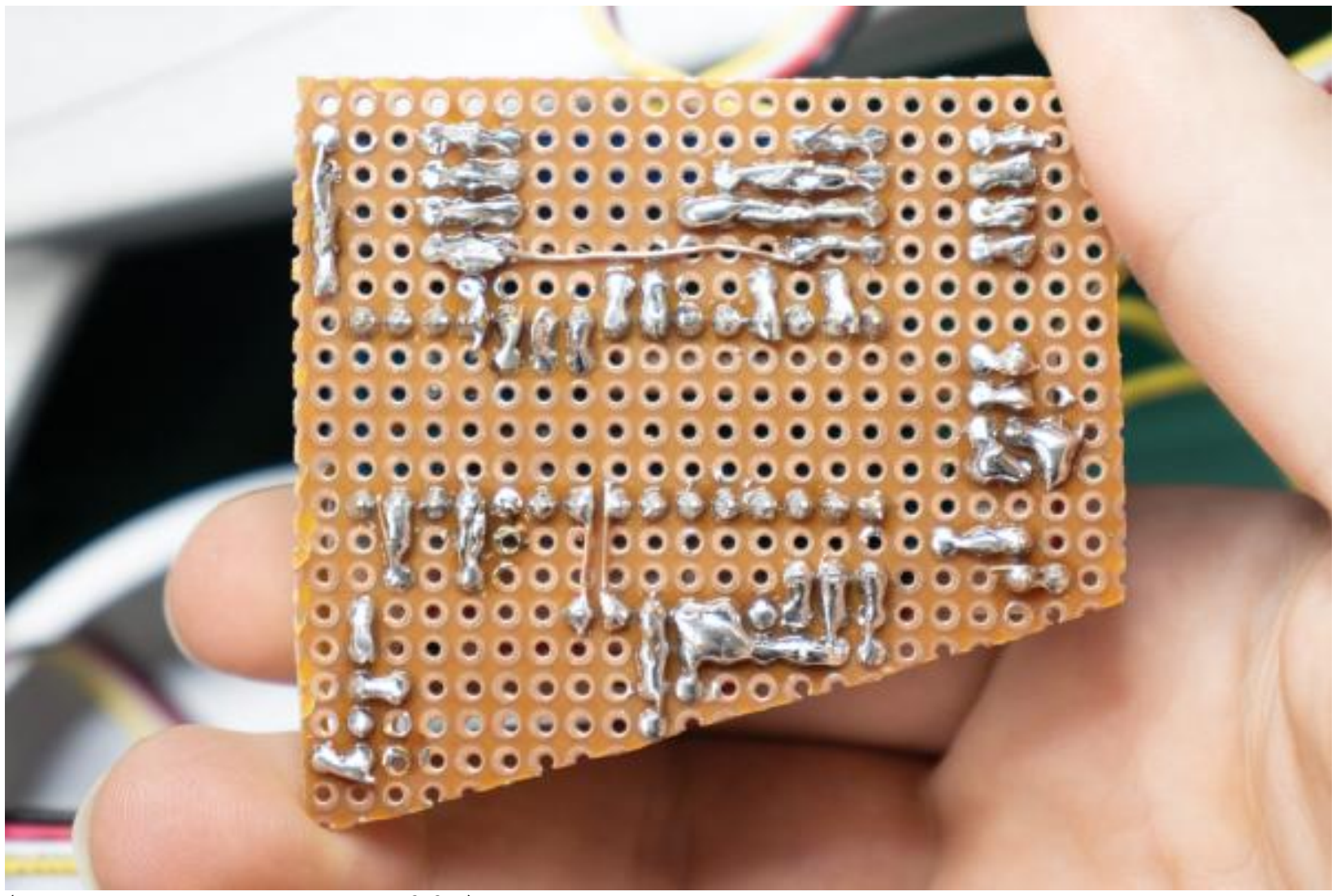
(/_images/6175143ac672e0533e7fd14a)

We also need to add some extra wires for controlling and reading from the main board. One of the white wires in the following photo is for receiving serial data from the main board, while the other one is for periodically waking up the hand controller. The controller goes to sleep after ten minutes, so after getting frustrated by having to press a button on the keypad every few minutes during the Fundamental build, we added a wire to simulate pressing the ‘Up’ button on the hand controller.

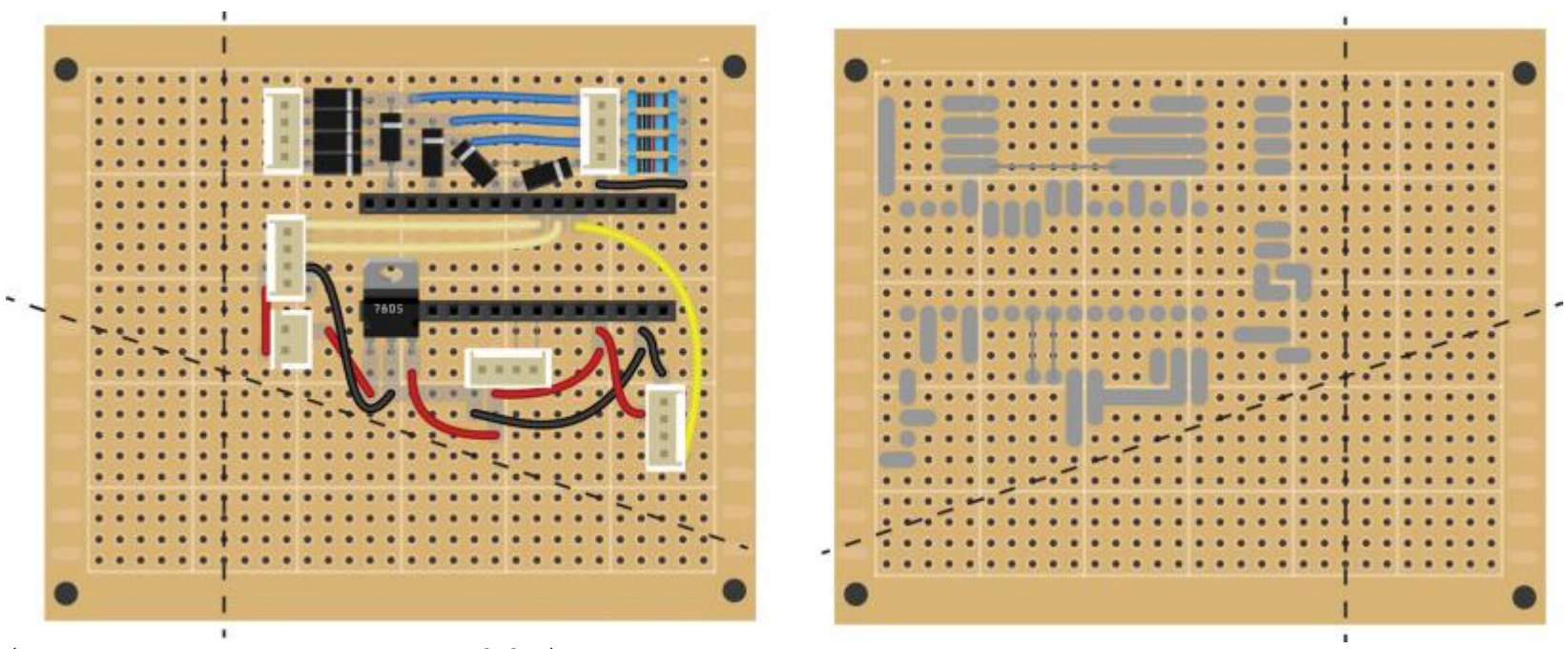
The underside of the board is quite compact – if you do wish to reference from it take careful note of where the traces go.



(/_images/6175143ac672e0533e7fd14d)

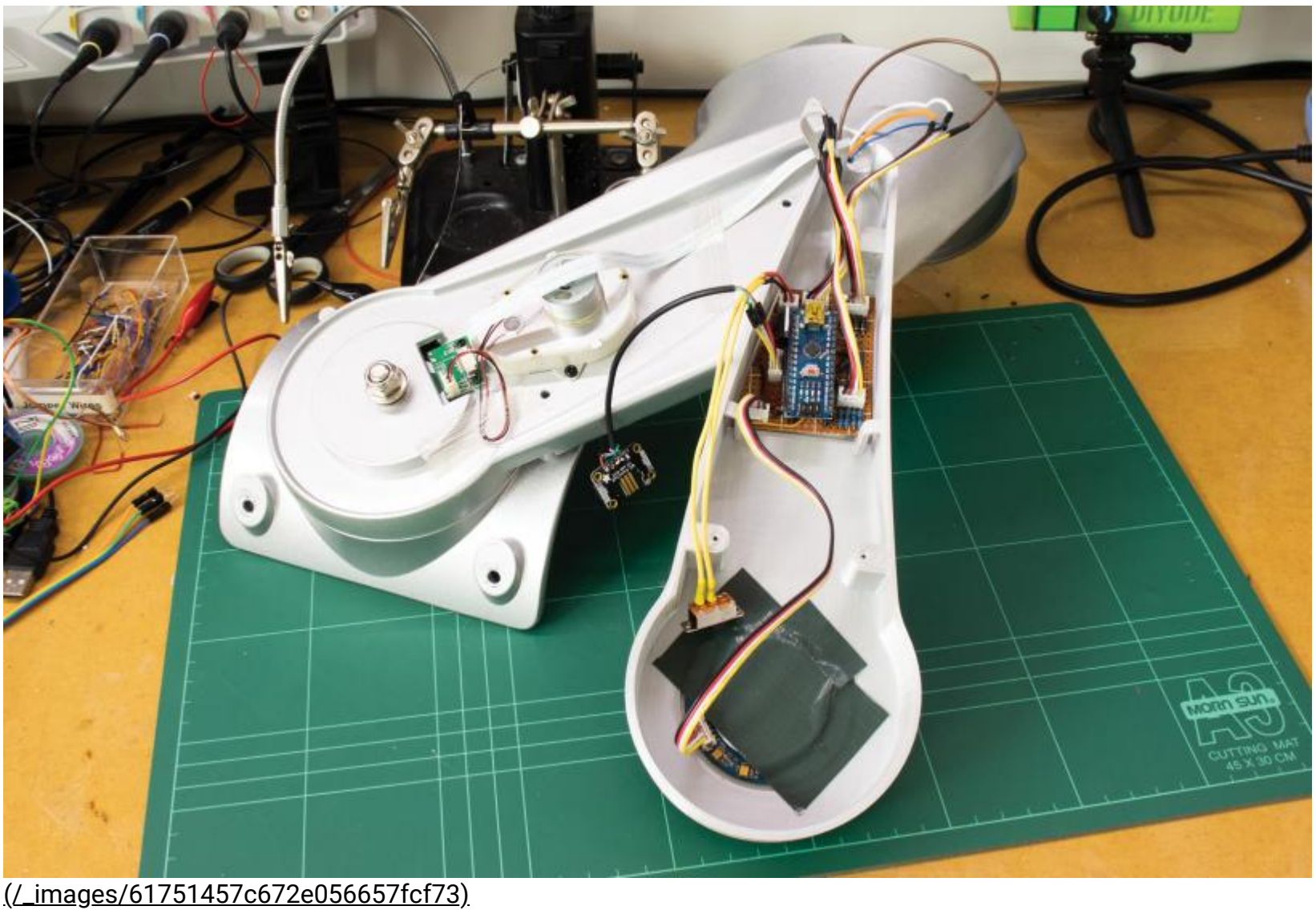


(/_images/61751457c672e056657fcf6c)



(/_images/61751457c672e056657fcf70)

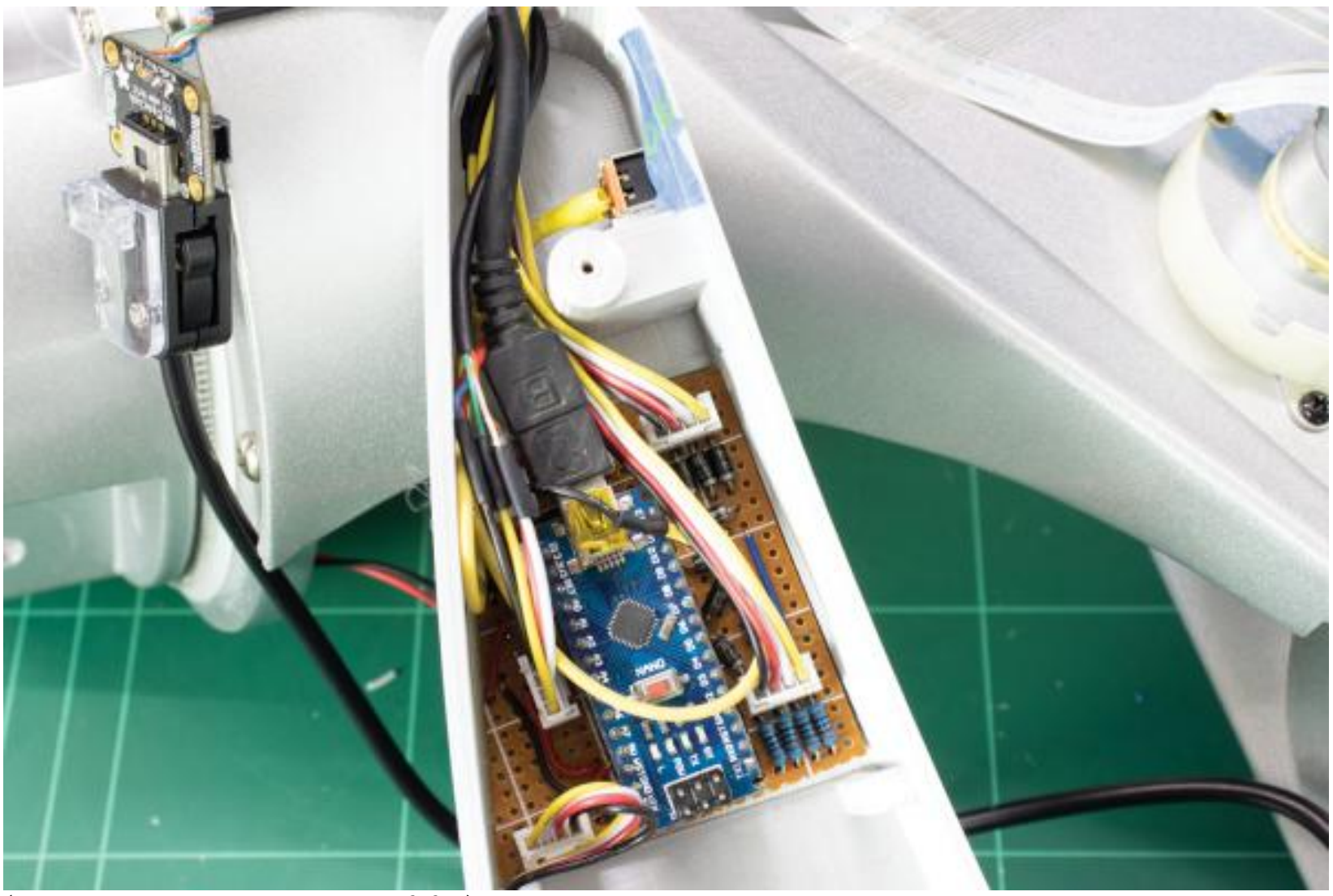
After inserting the board into the 3D-printed enclosure, we can finally get to connecting everything up. Because of the limited space inside the arm of the telescope, we need to be mindful of how wires are routed. This requires some careful wire management, or some elbow grease depending on your strategy. We have purposely routed all the electronics away from the DC motor to reduce wire snags and problems with getting the enclosure shut!



(/_images/61751457c672e056657fcf73)

To start reassembly, we used some cable ties to compact all of the wires down and inserted a mini-USB cable for programming and computer telescope control. A long cable is recommended. We then flipped the whole unit over and screwed in the 3D printed enclosure with self-tapping screws, holding it in place.

We're pretty happy with how the project looks. There aren't any ugly wires hanging out and we felt it's loyal to the original factory look whilst spicing up the design somewhat.



(/_images/61751457c672e056657fcf77)

(/_images/61751457c672e056657fcf7b)

All that's left now is to reassemble the tripod and re-mount the telescope!

STELLARIUM

Stellarium is our Planetarium of choice for this telescope, and it makes finding stars, nebulae and even satellites incredibly easy. It's also smooth, has nice graphics and has a simple control interface which is great for showing off the telescope's capabilities. The 32-bit version (required due to the 32-bit ASCOM driver) of Stellarium can be downloaded from https://stellarium-web.org/ and installed locally.

Once installed, just go to the Plugins section and enable the Telescope Controller. After an application restart, it's simply a matter of clicking on the telescopes button at the bottom of the screen and adding a new telescope. Just select the 'ASCOM' telescope type and open the driver window to configure the Serial connection – on whatever COM port the Arduino is on.

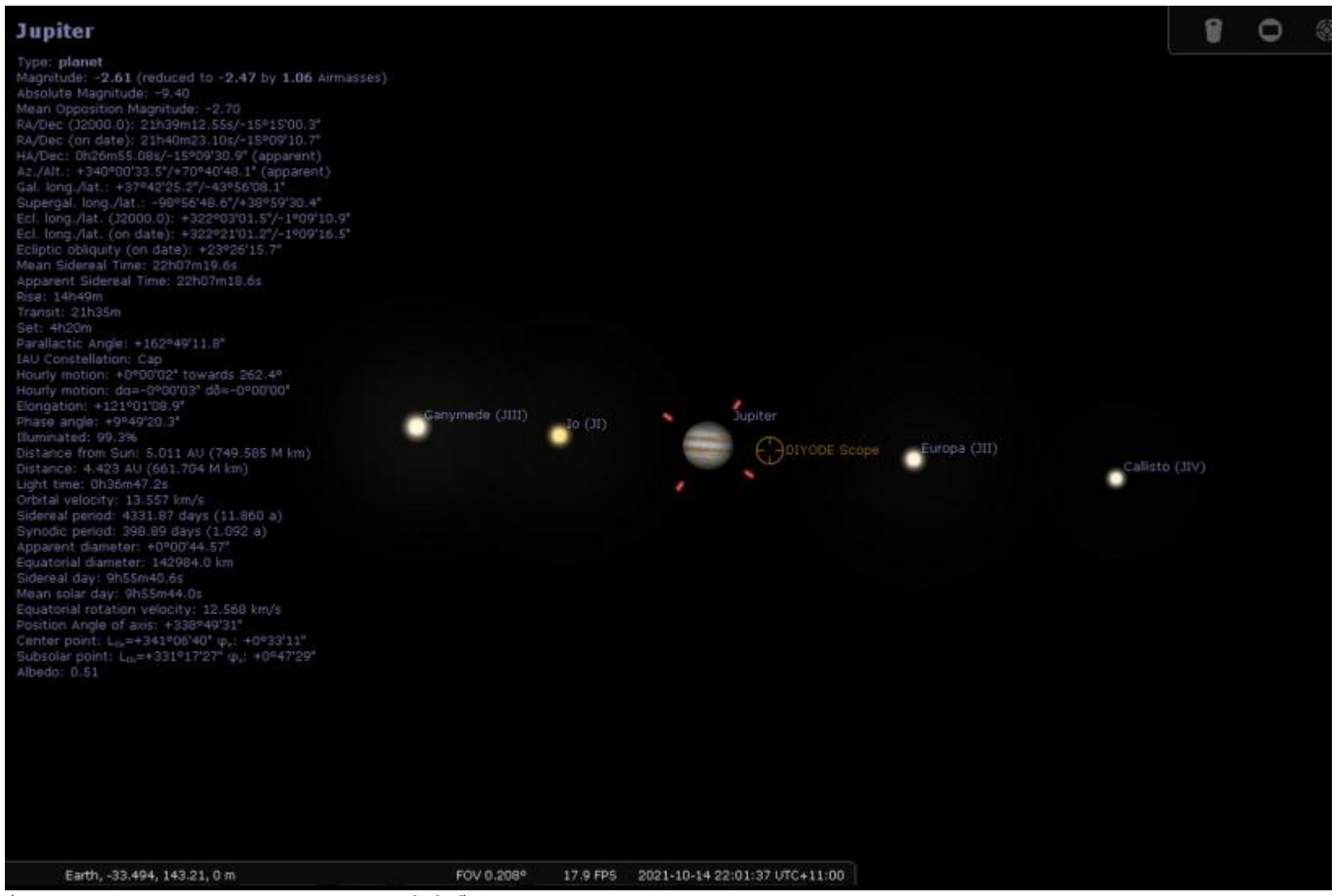
When Stellarium connects the Arduino, it usually hangs for a few seconds while it waits for the Serial connection to initialise. After that, the telescope reticule appears on the screen, wherever the Arduino thinks the telescope is pointed at! Moving the telescope will move the reticule accordingly.

Before we get any reasonable readings from the system, we need to implement a basic calibration sequence that can transform the current position of the mount to an actual coordinate in the sky. When the mount is first powered on, the Arduino thinks the mount is pointed at Altitude 0 and Azimuth 0, i.e. directly towards the horizon facing North.

Stellarium includes a "Sync" command that essentially tells the telescope the coordinates of a selected object. The telescope (in our case, the Arduino) can then use these coordinates to offset the coordinates received by the mount. In other words, we centre an object in the eyepiece – so we know the telescope is pointed at that object - and then use Stellarium to sync that object in the software.

TESTING

Once reassembled, we flicked the power switch and let the telescope auto-home. Once homed, the Wii Nunchuk has full control of the telescope. It's surprising just how accurate the joystick is, especially considering that it can be fully operated in one hand. It's easy to move the telescope at different rates between slow and fast, which is leaps and bounds ahead of the original movement system.



(/_images/61751457c672e056657fcf7f)

The C and Z buttons on the back of the Nunchuk are also super handy for slewing and tracking without fussing about with software. Often, we didn't have the patience to get Stellarium working for a short observing session and resorted to just using the Nunchuk to eyeball the slewing. It worked just fine! While we didn't implement any usage for the Nunchuk's inbuilt accelerometer and gyroscope, there could be a number of 'quick gestures' to do certain telescope actions. Tilting or panning could be used for minute adjustments.

(/_images/61751457c672e056657fcf81)

One issue that cropped up quite quickly was backlash. This is an issue that affects even the most expensive telescope mounts, but cheaper ones especially. Essentially, when changing directions in an axis, there is a short period of time where the drive gears do not mesh teeth. The result is a "deadzone" when an axis is moved in the opposite direction, sometimes causing jerkiness or uneven tracking. We implemented a very basic backlash compensation algorithm in the Arduino code but this turned out to be more hassle than it was worth, so we did away with it.

Overall, this project turned out pretty well considering the performance we ended up with was comparable to mounts 2-3x the cost. It's not suitable for astrophotography due to the inaccuracy of the motors and the backlash, but it's awesome for visual observing! The Nunchuk allows anybody to move the telescope precisely where they want it, without messing about with over or undershooting a target. Stellarium also worked well. Once calibrated, it helped us move the telescope to fainter objects in the sky that can't be seen with the naked eye.

WHERE TO FROM HERE?

This project differs somewhat from our usual process of 'Here is a cool project, here's how to make it!' because the telescope we have access to is not a super common unit. This project focuses rather on the broad process of adding your own functionality to existing gadgets you have lying around the house. We hope you've gained something out of seeing how we approached adding Arduino support to a 15-year-old motorised telescope.

One big issue we had with this project from the get-go was trying to understand the cryptic software protocols the telescope was using. We could have spent hours and hours trying to decode each and every byte of the communication between the hand controller and the telescope, but ultimately scope needs to be considered with a project like this. Why bother trying to inject your own commands into a system with no online documentation or standard when you could just inject it in an entirely different way? That was our thoughts when it came to the motor control system, which we instead used hardware to create custom motor signals instead of software. We encourage you to think outside the box when repairing or upgrading your own gadgets. Sometimes, the most correct or obvious solution isn't necessarily the most practical or time efficient.

It's great getting Arduino access into your own gadgets, because not only does it teach you every aspect of how a product works, but more importantly, it becomes a platform to make virtually unlimited tweaks and upgrades in the future. For example, one of the upgrades we're planning to make to this Bushnell telescope is to add WiFi connectivity so planetarium mobile apps can control the telescope wirelessly.

By the way, it's okay to not know how something works (that's the point of doing this whole project!), but definitely don't go tinkering with high-voltage, expensive or dangerous appliances and tools unless you've got a firm grasp on their working principles and the appropriate qualifications.

Anyhow, we hoped you learned something new about reverse engineering electronics, astronomy or perhaps even a touch of maths. Be sure to show us your tinkering projects on social media, we'd love to see what is possible!

(/_images/61751457c672e056657fcf84)

SHARE:

(https://twitter.com/intent/tweet?text=Seeing%20Stars%20-%20DIYODE%20Magazine&url=https%3A%2F%2Fdiyodemag.com%2Fprojects%2Fhacking-motorised-goto-telescope-arduino-uno-nano&via=diyodemag)

COMMENTS:

Please log in (/login?return=projects%2Fhacking-motorised-goto-telescope-arduino-uno-nano) to comment or to subscribe to this thread.