

INDEPENDENT WINTER PROJECT 2023

Star Tracker

Motor drive for Bresser Equatorial EXOS-2 mount

Susnata Chattopadhyay*

22MS077

*Indian Institute of Science Education and Research, Kolkata,
Mohanpur, West Bengal, 741246, India.*

June 28, 2025

Abstract

This paper presents the overview of my project to motorise the Right Ascension axis (of the mount) where I have tried to document as many things as possible including the problems which I faced, and how they were resolved. Wherever possible I tried to acquire my materials from scrap (recycled motherboard, LED Monitor and WiFi Router) to minimise the cost. For the body I have used an old smartphone box. The overall expenses sum to Rs 5565. It took me about 3 weeks to complete this, after which I also had to try a few modifications. This paper is organised into 7 sections, where in the first section I list the materials required, in the second section I discuss briefly about the theory and the working principles behind this project. In the following sections I have provided the script, it's explanations and some additional notes. At the end I provide some references for the reader to check out.

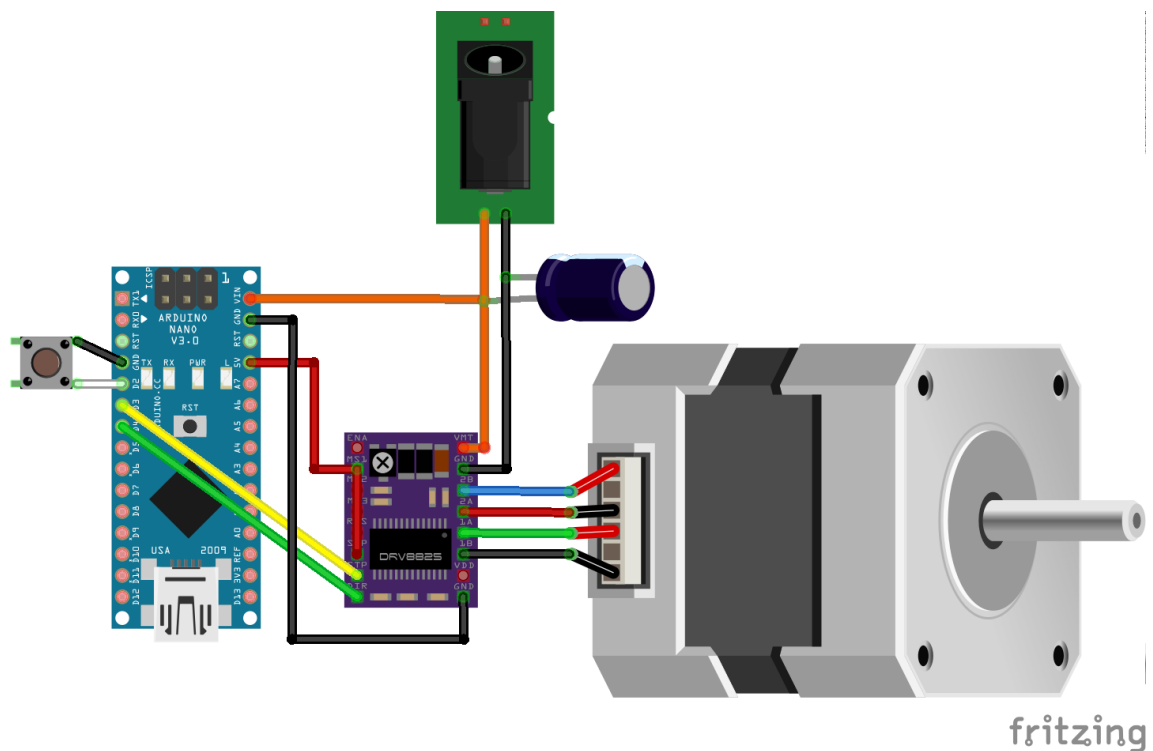
1 Materials Required

1. Arduino Nano
2. DRV8825 Motor Driver and heat sink
3. NEMA17 42HM48-1684 Stepper motor (0.9 °, 4.4 kg-cm) and it's Bend Bracket.
4. Multimeter
5. 100 μ F, 22 pF capacitor
6. Jumper wires/ 22 AWG *solid core* wires
7. IR Remote
8. CR2025 cell
9. TSOP Module/ IR receiver
10. GT2 Timing Belt, 158mm long, 6mm wide
11. GT2 40 Teeth Pulley, 6mm bore.

*Email: sc22ms077@iiserkol.ac.in

12. GT2 16 Teeth Pulley, 5mm bore.
13. 400 pin Breadboard
14. 12V 2A (Barrel Jack) Power Adapter
15. 2.5 in bolt.
16. M3 Screws

2 Circuit Diagram



3 Theory

The whole purpose of this project is to create a device (at low cost) which will mimic the celestial sphere's rotation (about Polaris), so that when viewed through the eyepiece the stars remain stationary. Thus it is now possible to explore deep sky objects with long exposure *imaging*.

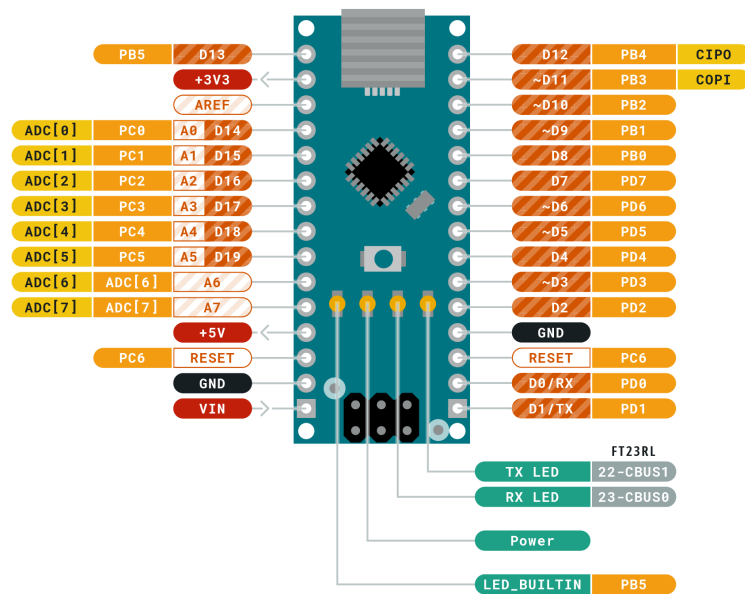
To rotate 360° , Earth takes $23\text{h } 56\text{m } 4\text{s} = 86164\text{s}$. This means to rotate by a degree it takes $86164/360 = 239.344\text{ s}$. The EXOS-2 worm gear has 1:144 ratio, i.e. it takes 144 RA knob rotations to move the scope by 360° or 24h. (*Remember RA is measured in hms format*).

Now our NEMA17 motor takes 32 microsteps to complete each step, by moving 0.9° . Which means it takes $360/0.9 = 400$ steps to complete one motor rotation. The pulley ratio is $40/16 = 2.5$. So one complete rotation of the motor will only rotate the RA knob (worm screw) by $1/2.5$ times, which actually needs 144 rotations to make a complete a 360° RA axis rotation. So, the total number of **microsteps** needed to move the scope by *one degree* is

$$\frac{32 \times (400 \times 2.5 \times 144)}{360^\circ} = 12800$$



ARDUINO NANO



Ground	Internal Pin	Digital Pin	Microcontroller's Port
Power	SWD Pin	Analog Pin	
LED	Other Pin	Default	

ARDUINO.CC

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1886, Mountain View, CA 94042, USA.

Figure 1: Pin Layout of Arduino Nano

Hence, how long to properly follow stars? It's exactly the 239.344 secs we calculated above. This corresponds to 239.344/12800 seconds per microsteps. This time in microseconds is our step delay.

4 Scripts

```
#include <IRremote.h>

// in micros, step duration to follow stars
// Calculated value 18698.75 microseconds
// -----65 teeth belt-----
// 19776 is bit slow
// 19276 is too fast
// 19676 is ideal
// -----64 teeth belt-----
// 19676 is too fast
// 19976 is too fast
// 20176 is too fast
// 20376 is bit slow
const unsigned long STEP_DELAY = 20276; // Decrease this to increase motor
// speed in the required direction (default, RA axis along star drift)

// pin connections
const int dirPin = 4;
const int stepPin = 3;
IRrecv Signal(2);
const int ledPin = LED_BUILTIN;

// rotation variable
int rotation = 1;
unsigned long int signal_data = 1;

// buffer time in micros: when we are at STEP_DELAY_MIN micros from having
// to change pulse status let's call (blocking) delayMicroseconds
// to allow a smooth pulse signal
// this must be less than 16383 and more than what logic
// does at each loop to honor STEP_DELAY
unsigned long STEP_DELAY_MIN = 100;

unsigned long lastTime; // micros, last time pulse has changed status
boolean stepPinStatus = false;

int velocity = 1; // speed is a reserved keyword ;)
//unsigned long i = 0; // for debug

void setup() {
  Signal.enableIRIn();
  Serial.begin(19200);
  Serial.print("RAtto: READY");
  // init Driver Motor pins as Outputs
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  // Set motor direction clockwise (HIGH)
  digitalWrite(dirPin, HIGH);
  // make sure no signal is sent
  digitalWrite(stepPin, LOW); // no signal
  // init Button pin as Input Pullup so no need resistor
  // pinMode(SignalPin, INPUT);
  // init led and turn on for 0.5 sec
  pinMode(LED_BUILTIN, OUTPUT);
```

```
digitalWrite(LED_BUILTIN, HIGH);
delay(500);
digitalWrite(LED_BUILTIN, LOW);
Serial.print("and RUNNING with pulse length:");
Serial.println(STEP_DELAY);
// Resolution is 4microsecs on 16Mhz board
lastTime = micros();
}

void runStep(long stepDelay) {
    unsigned long halfStepDelay = stepDelay/2; // duration of each pulse
    unsigned long dt = micros() - lastTime; // delta time elapsed since
    //previous stepPin status change
    // micros() overflow after approximately 70 minutes but "-" logic
    // above will work anyway to calculate dt

    if ( dt > halfStepDelay) { // no luck, too fast, we can't
        \\\honor halfStepDelay.
        Serial.print("Can't honor the speed, dt:");
        Serial.println(dt);
    } else if ( dt > halfStepDelay-STEP_DELAY_MIN) {
        // need to switch pulse in halfStepDelay-dt micros, let's do it:
        // remaining delay to keep current pin status to honor halfStepDelay
        delayMicroseconds(halfStepDelay-dt); // which is less than STEP_DELAY_MIN
        //change pin status
        stepPinStatus = !stepPinStatus;
        digitalWrite(stepPin, (stepPinStatus?HIGH:LOW));
        /* debug
        i++;
        if (i == 1000) {
            unsigned long pulseDuration2 = (micros()-lastTime);
            Serial.println("logB:");
            Serial.print("stepDelay: "); Serial.println(stepDelay);
            Serial.print("halfStepDelay: "); Serial.println(halfStepDelay);
            Serial.print("lastTime: "); Serial.println(lastTime);
            Serial.print("dt: "); Serial.println(dt);
            Serial.print("delayMicroseconds:"); Serial.println((halfStepDelay-dt));
            Serial.print("pulseDuration2: "); Serial.println(pulseDuration2);
            delay(5000);
            i= 0;
        }
        */
        lastTime = micros(); // last time we updated the pin status
    }
}

void loop() {
    runStep(STEP_DELAY/velocity);
    /*
    * logic executed at each loop must take less than (STEP_DELAY/2-STEP_DELAY_MIN)
    * and less than STEP_DELAY_MIN. This is just a an IF for now...
    */
    // cycle velocity on button press
    if ( Signal.decode() ) {
        digitalWrite(LED_BUILTIN, HIGH);
        Serial.print("New Velocity:");
        signal_data = Signal.decodedIRData.decodedRawData;

        if ((signal_data == 0x3A653A09)) { // + PRESSED
            digitalWrite(dirPin, HIGH);
            rotation = 1;
        } else if ((signal_data == 0xAFAB9550)) { // - PRESSED
```

```
        digitalWrite(dirPin, LOW);
        rotation = -1;
        Serial.print("Direction Reversed!");
    } else if ((signal_data == 0x92B87D57)) { // 1 PRESSED
        velocity = 1;
        Serial.print("Speed: Normal");
    } else if ((signal_data == 0xA01691C3)) { // 2 PRESSED
        velocity = 4;
        Serial.print("Speed: 4x");
    } else if ((signal_data == 0x729AAC37)) { // 3 PRESSED
        velocity = 8;
        Serial.print("Speed: 8x");
    } else if (signal_data == 0x1DB589D0) { // 4 PRESSED
        velocity = 16;
        Serial.print("Speed: 16x");
    }
    else if (signal_data == 0x2711BBF0) { // 5 PRESSED
        velocity = 32;
        Serial.print("Speed: 32x");
    }
    Serial.println(rotation*velocity);
    //delay(10); // wait to release button... this is blocking
    digitalWrite(LED_BUILTIN, LOW);
    Signal.resume();
    lastTime = micros(); // reset time
}

}
```

4.1 Explanation

Here I include some explanation of important places which might help you understand the code.

Inside the `setup()` function, `Serial.begin()` is used to establish the serial connection between the development board at a **baud rate** of 19200. We used the `pinMode()` function to configure the digital pins connected with STEP and DIR as output pins.

Baud Rate

It is the rate at which the number of signal elements or changes to the signal occurs per second when it passes through a transmission medium. The higher a baud rate is the faster the data is sent/received. While *Bit Rate* is the number of binary bits (1s or 0s) transmitted per second.

$$\text{Bit rate} = \text{Baud rate} \times \text{bits per signal or symbol}$$

4.1.1 Control spinning direction

To control the spinning direction of the stepper motor we set the DIR (direction) pin either HIGH or LOW. For this we use the function `digitalWrite()`. Depending on how you connected the stepper motor, setting the DIR pin high will let the motor turn CW or ACW.

4.1.2 Control Speed

The speed of the stepper motor is determined by the frequency of the pulses we send to the STEP pin. The higher the frequency, the faster the motor runs. You can control the frequency of the pulses by changing `delayMicroseconds()` in the code. The shorter the delay, the higher

the frequency, the faster the motor runs. I have included some links in the References section, those will help the reader learn more.

4.2 Remote Functions

You can also customise the remote to turn the motor turn on/off automatically. Try this as a challenge.

Button	Operation
+	Anti-Clockwise
-	Clockwise
1	Normal Speed
2	4×
3	8×
4	16×

5 Additional notes

- During the project I had to manually design large heatsinks for the stepper driver as I got to know with small heatsinks (which were provided by the manufacturer) heats up quickly and it shutdown thus restricting it's use to a few minutes. I built them by taking a few measurements of the dimensions of the driver and then cutting them from the corners of an old recycled motherboard heatsink. This was a very painstaking process!



Figure 2: These instruments were used (along with an electric angle cutter) to cut the pieces out of the aluminium heatsink

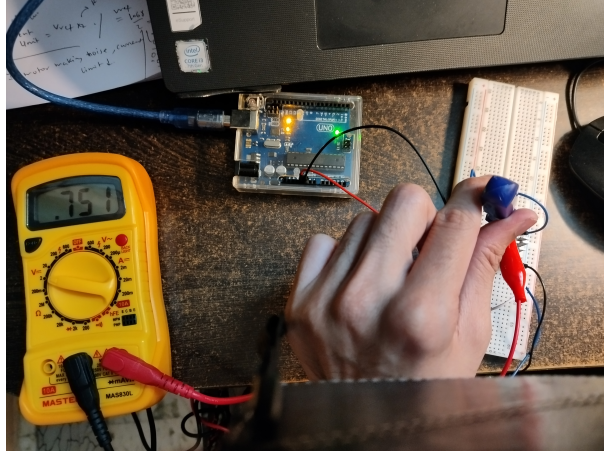


Figure 3: An example to show how you should change the potentiometer to adjust the voltage, using a mutimeter and screwdriver.

5.1 Set the current limit in Motor Driver

Before you start programming your Arduino and start using the driver there is one very important thing you need to do that a lot of people forget: **set the current limit!**

For this driver (DRV8825), we calculate the V_{ref} by

$$V_{ref} = I_{max} \times R_{sen} \times A_V$$

I_{max} is the maximum current required per phase of the motor. R_{sen} is the shunt resistance (can be R250 or R100). A_V is the gain of the driver (in my case it is 5 V/V). You can find it in the datasheet. For my NEMA17 motor it was 1.68 A. So the corresponding $V_{ref} = 1.68 \times 0.25 \times 5 = 2.1$ V. For safe side it is advised to keep the voltage a little below ($\simeq 2.08$ V) to protect the circuit components.

5.1.1 SMD Resistor Code

It is important to learn the nomenclature of the Surface Mounted Resistances (which you will see as black boxes on the driver).

1. **3-Digits:** $250 \equiv 25 \times 10^0 = 25\Omega$, $653 \equiv 65 \times 10^3 = 65k\Omega$
2. **4-Digits:** $7992 = 799 \times 10^2 = 79900\Omega = 79.9k\Omega$.
3. The letter "R" is used to indicate the position of a decimal point for resistance values lower than 10 Ω . Thus, R250 would be 0.25Ω and 0R01 would be 0.01Ω .

Now use a voltmeter with the red probe clipped to a screw driver (- 2.0) which shall be used to adjust the potentiometer (circular metallic thing) on the driver, with the black probe connected to the GND of the stepper driver. (You can watch some videos on Youtube).

5.2 Testing the robustness of the motor

Before actually finalizing the circuit of the star tracker, you should run a demo code to check how long the motor can continuously rotate by (preferably 1-2 hrs), without the stepper driver heating up too much and shutting down by itself. You must install proper big heatsinks on both above and below the driver. Extend the pins (using pin extender) to fix it to the breadboard.

5.2.1 Motor neither vibrating nor spinning?

I ran into this problem when I was working on it for the first time. There might be number of reasons. First recheck your wirings (trivial) for any loose connections. Your Arduino might not be powered (**Vin**, **Gnd**). Low current power adapter might have been used (not enough to provide sufficient torque to spin the motor). Changing the polarity of wires.

Check the continuity of the circuit Look for the *WiFi* symbol on your multi-meter, rotate the knob to that mode. Now using the red and black probes alternately connect to 2 pairs of wires Red-Blue and Black-Green, if a *beep* is sounded then circuit is fine! (otherwise short circuited). You can also determine if the circuit is broken by measuring their *resistances*. For the 2 pairs of wires their resistances roughly match with the rated Resistance $\simeq 1.8\Omega$.

Faulty Breadboard Ensure the the voltage across the power grids on either side of the board remained constant as you moved down the board. This turned out to be *one* of my case. Clip two probes of multimeter to two jumper wires and keep plugging and checking like this.

Stepper Driver corrupted Somehow with the help of my friend, I found out my stepper driver got damaged and it didn't show any voltage on multimeter. The motor worked fine when I replaced it with new driver.

5.3 Code Not Uploading

The following error might show if Arduino IDE is unable to upload code:

```
avrdude: stk500_recv(): programmer is not responding
```

This happened with me, while I was operating the motor. All of a sudden there was a *gor-gor* sound, and the motor stopped spinning and the Arduino became unresponsive. The following Forum post ([link](#)) was helpful in solving this issue.

5.4 Caution

Make sure to replace the jumper wires with 22 AWG solid core wires, as they have higher current rating. Otherwise excessive heating will cause the silicon insulation of your wires to melt.

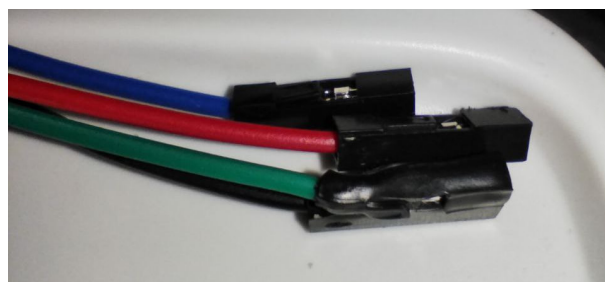


Figure 4: After running the motor for some time, these connecting wires heated up so much that the female pins started to melt.

[BONUS] Upgrading the Bracket

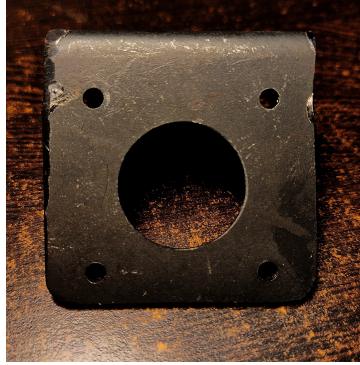


Figure 5: Former metallic bracket for the stepper motor

The initial metallic bracket (Fig 5) that was installed did not have the facility to slide the NEMA Stepper motor sideways in order to tighten the timing belt. In order to overcome this issue, I decided to 3D print a modified (and sturdy!) bracket. To create this, I worked on the initial design by *spandit* (at [Thingiverse](#)).

I used [FreeCAD](#) a free 3D modeling software to design this. The final design based on my Bresser EXOS-2 mount measurements can be found [here](#)¹. However, I strongly recommend the attempters to create this model themselves – it is a fun exercise.

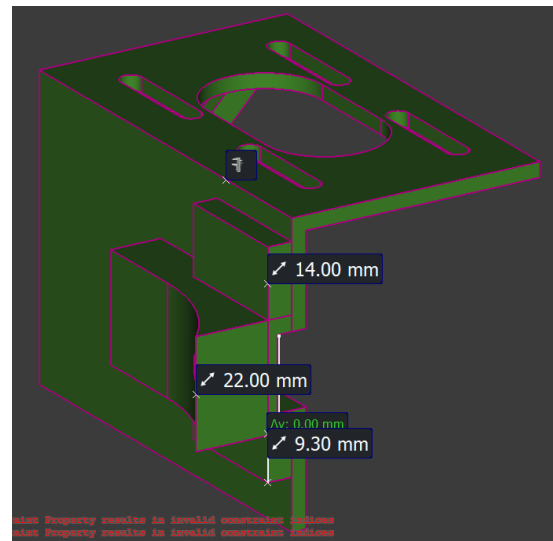
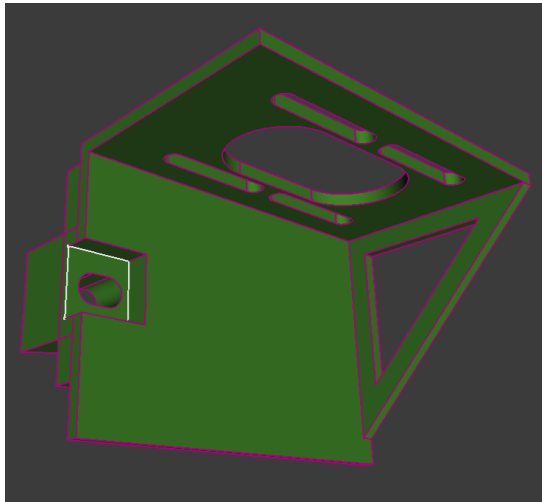


Figure 6: Some of the design snapshots

¹Start with removing all the fillets. My design was a bit tight for the mount, so I had to sand the edges. You can obtain a more snug fit by decreasing the padding of the *bottom fin* and the length of the part labelled as 14 mm

The Final piece!

Update: With the fitting of upgraded bracket, the project that started as a dream in Nov 2023, finally came to a cherished conclusion after 1.5 years of improvements. And I believe this document shall serve as a detailed guide for any amateur (like me) who is eager to explore the universe right from their backyard/rooftop!

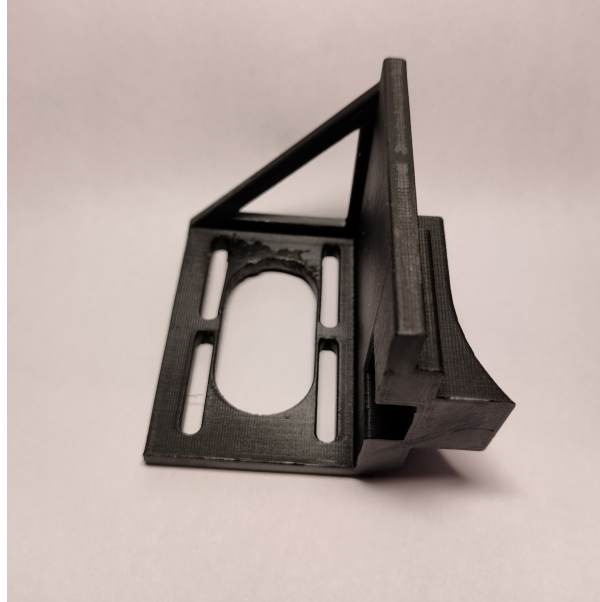


Figure 7: Printed with heat tolerant ASA material (with 60% infill) the upgraded bracket mount is ready !

Future Improvements One can upgrade to TMC2226 because it has a larger heatsink as well as better heat dissipative capacity - better for high current rated motors.

6 Acknowledgement

I am grateful to his friends Comrade Shantonu Dutta and Diptanuj Sarkar for their generous contributions and guiding me with their invaluable experiences. This project could never have been achieved without Shantonu's help all along. I am also thankful to my family for funding this project(:)) and motivating me all throughout this span.

7 References

1. Motivation for this project [<https://www.cloudynights.com/topic/731261-yet-another-diy-ra-drive-for-eq5-exos2-eq3-etc-etc/>]
2. Please read these websites carefully to set up your motor driver [<https://www.makerguides.com/drv8825-stepper-motor-driver-arduino-tutorial/>],[<https://microcontrollerslab.com/drv8825-stepper-motor-driver-module-arduino-tutorial/>]
3. Datasheet for DRV8825 Stepper motor driver: [<https://www.ti.com/lit/ds/symlink/drv8825.pdf>]
4. IR Remote working: [<https://www.makerguides.com/ir-receiver-remote-arduino-tutorial/>]

8 Results



Figure 8: M42 or Orion Nebula from my home. This image is a stack of 138 Lights (8s ISO 500), 36 Darks and 47 Bias Frames. Images are stacked and processed using [Siril](#)- a free software.

9 Gallery

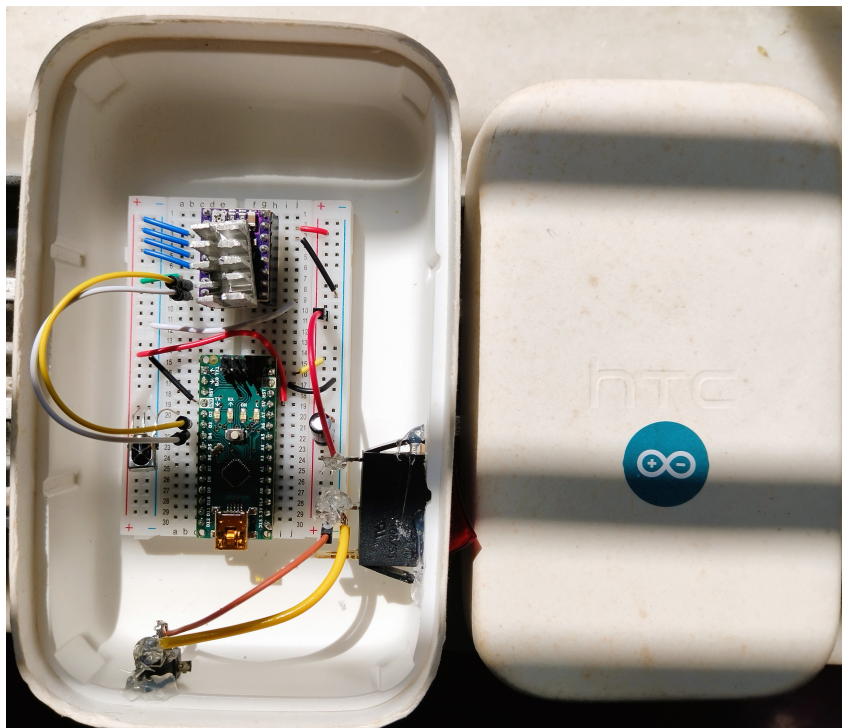
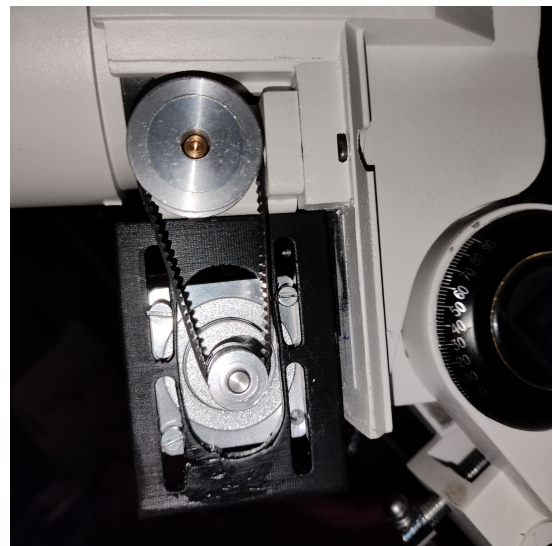


Figure 9: After completion - StarTracker!



(a) After mounting the stepper motor and attaching the belts (Old Bracket)



(b) With the new bracket and belt.

Figure 10: Mounting the motor



Figure 11: And we are ready to stare !