A Star Tracking Drive for Dobson Mounts, Implemented with Arduino Microcontrollers.

Dr Mike A. Casbon

Abstract

A Dobson mounted Newtonian reflector telescope is a compact, portable and affordable system for the amateur astronomer, and its combination of ease of construction, good resolution and excellent light gathering ensure it remains popular. However, the rotation of the Earth means that only images with very short exposure times can be captured (fig. 1). If a Dobson telescope mount is fitted with stepper motors then its motion can be precisely controlled, in order to follow a given star around the sky it only remains to calculate the correct speeds at which to drive the Altitude and Azimuth motors. If we know the latitude of the telescope, and the position of the target at a given time then the new position after a set time interval can be calculated, and hence the average speed required of each drive to arrive in the correct place at the new time can be determined.

Figure 1. A 30 second exposure with a fixed camera.

Introduction

The purpose of this project was to produce a practical system, simple to use in the field, for the purpose of astro-imaging, The electronic hardware is simple to assemble, and the software is easy to instal using the free Arduino development suite. The fact that our system will move in a series of straight lines, whereas the stars move in smooth curves, is tolerated, The time interval chosen for the prototype was 1 minute, which gives our “thrupenny bit” 1440 sides, and should this prove too coarse then the microcontrollers used could certainly handle updating every six seconds. Likewise, there is no allowance for motion other than sidereal, as the short exposures required for Lunar and most planetary imaging will not show any ill effects from this simplification. Also, there is no compensation for field rotation, that will have to be handled in image stacking software.

The Mathematics

The calculations involved are simply trigonometry, but can appear daunting when viewed as a whole, the problem must be broken down into steps, each solvable in a simple manner. If the altitude and azimuth figures can be converted to a Right Ascension/Declination type coordinate

system, referenced to the axis of the Earth rather than vertical to the observer, then it is simple to add a rotation appropriate to our fixed time interval to the RA figure, the Dec remaining unchanged of course. We can then work backwards from this to a new altitude and azimuth, and so find the difference we have to compensate for.
For the purposes of this exercise I have assumed the celestial objects are all fastened to a crystal sphere, of radius 1, this makes the maths a lot easier, and has a classical appeal.

1) convert the Alt/Az figures to rectangular x.y,z coordinates, with $+x_a$ being east, $-x_a$ being west, $+y_a$ north, $-y_a$ south, $+z_a$ vertically upwards, and $-z_a$ vertically downwards.(fig. 2)

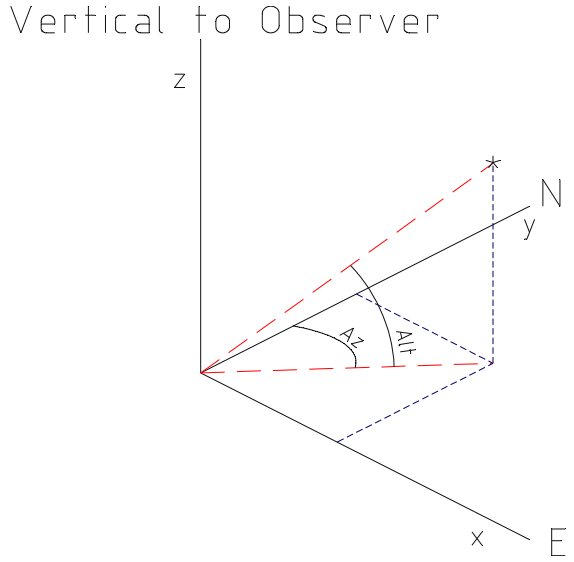

Figure 2 Converting to x,y,z format

2) Looking along the x axis, regarding the y_a and z_a as being on a plane, convert y_a and z_a to 2D Mag/Angle form (fig. 3).

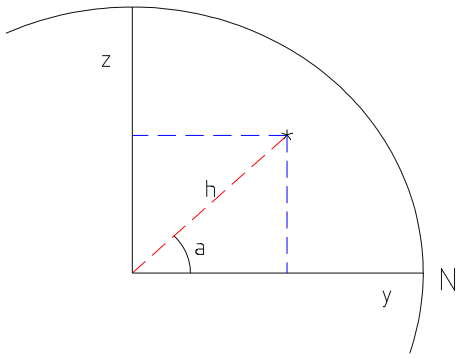

Figure 3 convert the z and y into Mag/Angle format

3) Add the difference between 90 degrees and the latitude of the observer to the angle, so the reference is now rotated to the celestial pole rather than horizontal to the observer (fig. 4).

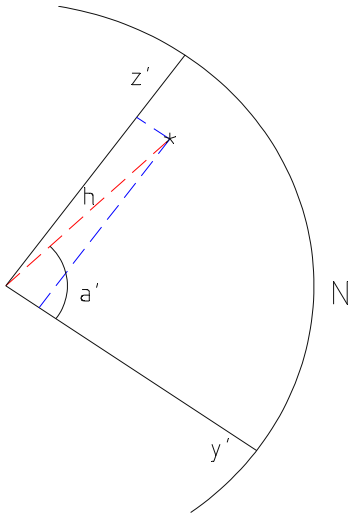

Figure 4 Rotate the angular reference from vertical to aligned with the celestial pole

4) Convert the new Mag/Angle figures back to rectangular, note that x_a has not been changed by these operations, as in effect we rotated our reference around the x axis.

5) The new pole referenced z_p is the tangent of our declination, and will remain unchanged by the time shift.

6) Looking down the z_p axis, from the celestial pole, regard x_p and y_p as being on a plane, and convert to Mag/Angle, with due east being 90º (arbitrary)

7) Add the appropriate angle increment for our time period to the angle (0.25º for our 1 minute), and convert back to 2D rectangular, x_p' and y_p', z_p is unchanged (fig. 5).

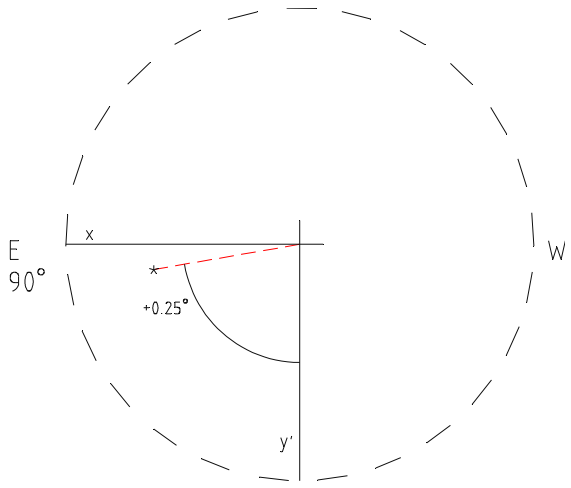

Figure 5 Convert to Mag/Angle format and add the time shift rotation

8) Looking along the east-west x axis once more, change y_p' and z_p to 2D Mag/Angle

9) Subtract the difference between 90º and the observer's latitude, to restore the reference to horizontal for the observer.

10) Convert back to rectangular y_a' and z_a', (x_a', unaffected by this operation, is equal to x_p')

11) Convert back to Alt'/Az' form, and find the differences, dAlt and dAz.

12) Scale the required angular velocities according to the pulses/degree factor of the telescope.

The Software and Hardware

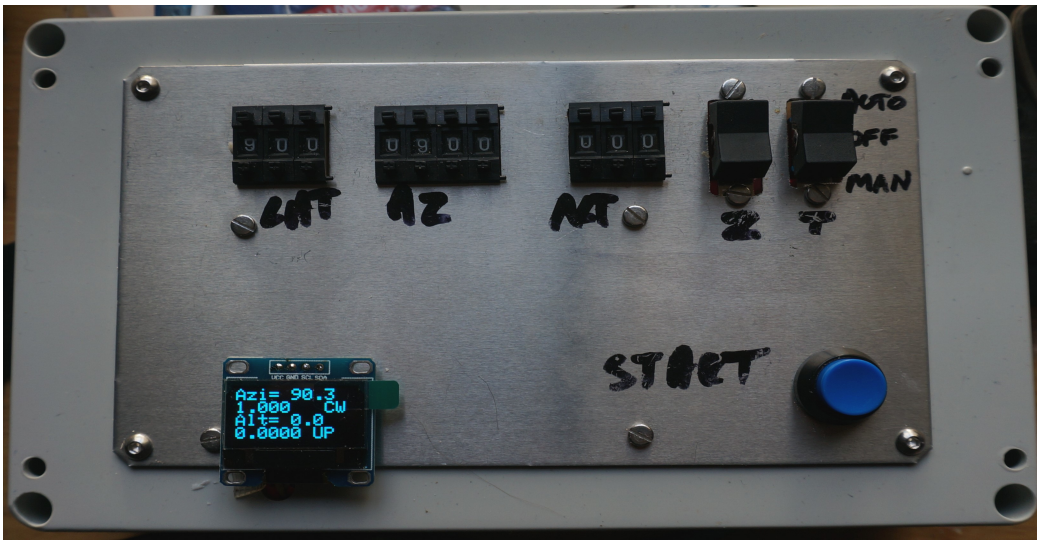

Figure 6 The prototype autotracker, at the North Pole

Those who have dealt with this type of problem before will be aware that while finding, for instance, the sine of an angle gives a unique answer, the reverse process does not hold, we get two possible answers (ignoring multiple rotations), and while mathematicians can happily deal with divide by zero events, a microcontroller will simply hang up. The first issue has been dealt with in the code by If/Else branches, where different calculations are applied depending which quadrant the point is in, the second, usually occurring in atan(a/b) statements, has a calculation executed if b>0, or b<0, but gets a flat “angle = 90” if b=0. The result seems robust in all tested scenarios – but expect the unexpected!

In the prototype the calculations were performed on a master Arduino Mega 2560 and then the speed information was transferred to a pair of Arduino Unos, via two 10 bit parallel buses, giving 1024 different speeds for each axis. This was satisfactory for the Alt drive, but the lazy assumption that the maximum speed for each axis would be sidereal, and that all results would be a blend of these two proved to be not only lazy but wildly inaccurate in the case of the azimuth. In fact a Dobson mount tracking through the vertical has to instantaneously swing through 180°. Since we have stipulated a time period of 1 minute, then this reduces to a more manageable 720x sidereal, and some drives may indeed achieve it, however, it does carry the risk of damage to equipment and people, especially if cables are attached, so limiting the maximum speed should be considered. To cope with the larger range of speeds in software terms, and to simplify construction, the information is now transferred as 16bit numbers via the two wire I2C bus, so we have up to 65,536 speeds available on each axis. To improve the spread of possible speeds across the required range a dB compression is applied for the transmission. The tracking information is robust, and if sighting is lost for any reason then as soon as it is manually restored the speeds fed to the motors will be correct again.

Calibration and use in the field.

Only three inputs are required, the latitude of the telescope, together with the altitude and the azimuth of the object to be tracked. The latter can be taken from a calibrated mount, or read in advance from a program such as Stellarium, in which case no alignment of the mount is required

(other than up being up). The data is entered via thumbwheel switches, and a synchronising button pushed at the appropriate time. If the Alt/Az data is taken from a reference this can be done well in advance of observing commencing, run the simulator time a few minutes ahead, enter the predicted Alt/Az figures, wait for real time to catch up with the simulator and press the button. As long as the unit remains powered then the speed outputs will be correct.

Adjustment for the gear ratio of a particular mount is performed by altering one number in the code for that Uno. Setting the latitude to the North pole results in an azimuth drive of 1x sidereal, I then put a laser pointer on the mount and timed the moving spot on a wall 20 feet away. A quick hop to the equator and looking due east results in an altitude drive of 1x sidereal, I have a digital inclinometer on my scope, but the moving laser spot method would work just as well.

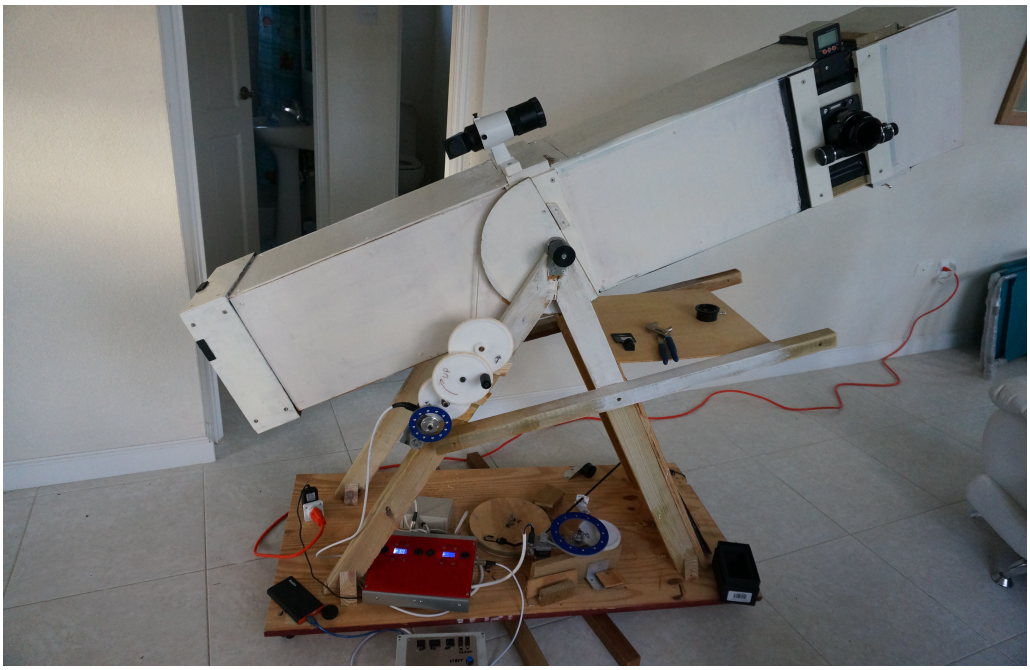

Figure 7 The prototype attached to a 10” Newton/Dobson

Timing is controlled by the crystal oscillator on the master Mega, for ultimate precision the 60 second interval can be checked by means of the serial monitor feature of the programming suite, select “time stamp” and observe the time interval of the updates coming through over an hour or two, there is a single number to adjust in the Mega code to trim this if required.

Getting the correct polarity of the Up/Down and CW/ACW signals is just 50:50 depending on the gear train and how the motor windings are connected. Should the direction of travel be incorrect then either the wires of the stepper motor can be changed, or the software altered to suit, whichever is the easiest.

Notes

Three Arduinos are used, to simplify the coding and to ensure continuous pulse output during the recalculation. For the master a Mega 2560 was used, as this has enough I/O pins for the decade switches, and more programming space, The cheaper Uno version was used for the oscillators, clones can be purchased for around £7.

Motor control was performed by ready made controllers, which are cost effective and provide all the features needed such as current limiting and selectable microstepping, and only require a pulse

train logic signal from a Uno, and a direction signal from the master Mega. Experienced Arduino users might consider using motor control shields connected to each Uno, but in cost terms there is little to recommend that complication.

Use of the I2C bus means only two wires are needed to connect the controllers and the little OLED panel, which provides a readout of the speeds and new position. The only complexity is the connection of the thumbwheel switches, being of the Binary Coded Decimal type these have one common wire and 4 data lines each. I opted for tenths of a degree, but I'm not convinced this resolution is essential, and for a fixed observatory the latitude could be hard coded instead – so the number of switches could be reduced to five. Such switches are very expensive from the mainstream distributors, but I purchased a set of ten via Amazon for a reasonable sum (make sure they are the BCD type, not the ten wire type).

The OLED panel is optional, but very useful, this was obtained from Bitsbox. You could run blind, or monitor what's going on via the USB programming port on each controller, but this is a nuisance in the field, as it would require a PC to be connected up.

I added a couple of centre OFF rocker switches to the Uno pulse outputs, allowing the motor drivers to be connected to the original manually adjusted oscillators, set to a speed faster than the automatic feed, this allows fine centring of the image by advancing or retarding each axis as required.

Stiction – while a certain amount of this is tolerable or even desirable in a manual mount, it will make an automatic mount move in a series of jerks rather than a smooth continuous motion, I used ball bearing races on my trunion pins to good effect, and one of the better grade lazy Susan turntable mounts for the azimuth.

I am happy to provide the code and support to anyone who is interested, as a thank you to the community for all the excellent freeware that I use. I don't claim my coding efforts are very elegant, but they do work and I have commented throughout should anybody wish to go beyond simply uploading and using the programs.

Future development.

It might be possible to calculate field rotation, by running the calculations in parallel for a notional point close and directly below the target, and then comparing the relative bearing each minute. If I ever find this is essential, I might try, if somebody else wants a go, feel free to modify the code.

As it stands, this will only work in the northern hemisphere, if anybody would like to adapt it for south of the equator, again, feel free to modify and share.