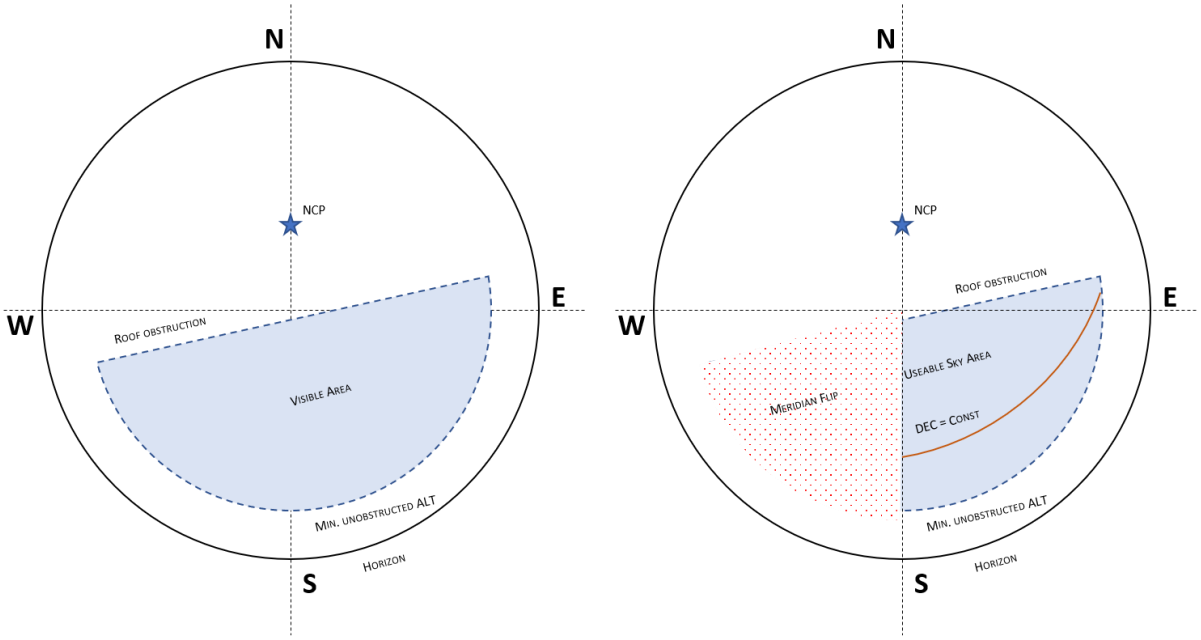
Fast Polar Alignment without Polaris

Abstract:

While there are solid methods and good tools available for polar alignment, I was not content for my setup.
Having limited time to mount, align, photograph and demount my gear is driving me to reduce all parts that are not 'photography'.
Setting up on my terrace, located to the southern side of the house, is a further limitation, as Polaris cannot be seen. As the meridian flip may introduce additional uncertainties, the available sky is further limited as shown in the two following schematic pictures.



Key focus areas for me are:

- Ability to align the mount without Polaris in sight.
- Automated process to speed up alignment process.

Words of caution:

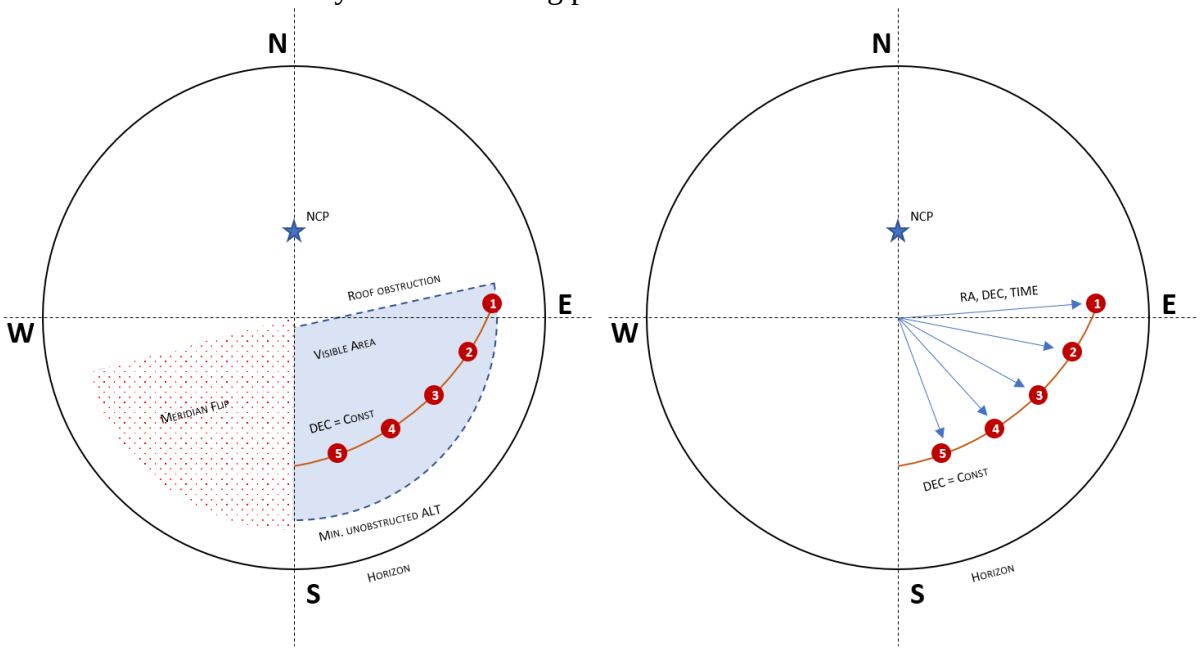
- This is a proof of concept and has not been soundly implemented.
- Some values are hardcoded and need to be adapted for your setup.
- The slewing assumes your mount to be free of obstruction. If it is not damage may occur.

Requirements:

- Equatorial mount with Azimuth and Altitude adjustments.
- GoTo Mount driven with EQmod. (other ASCOM options may work, but are untested). Basically, a Skywatcher mount or a mount that has been fitted with Skywatcher electronics.
- Camera that can interface through SharpCap. (e.g. ASI ZWO, …)
- SharpCap registered version for use of scripting. (12 EUR / year)
- PlateSolve2 (free)
- RStudio installation, R installation with the following libraries, 'pracma' 'lubridate'. (free)

Method:

The scripting function in SharpCap is used to slew the mount along a RA line with constant DEC. At regular intervals pictures are taken and the time of the picture taking is recorded to later establish the assoc. HA and LST.
These pictures are then sent through plate solving to determine the RA and DEC positions of the centers of the pictures.
Finally, the plate solving results are collected in a CSV file.
This is shown schematically in the following pictures.



At this point there is a handoff to RStudio that processes the previous CSV file.

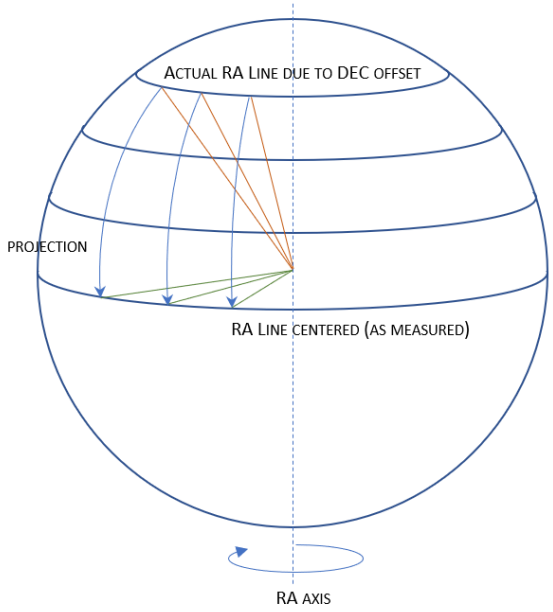
Correction for refraction:

First the polar coordinates are transformed to Cartesian. Then these are corrected for refraction based on their altitude, ambient temperature and pressure. The refraction corrected Cartesian coordinates are transferred back to polar for the next step.
Formula is from “Practical Astronomy with your Calculator or Spreadsheet” by Peter Duffett-Smith and Jonathan Zwart.

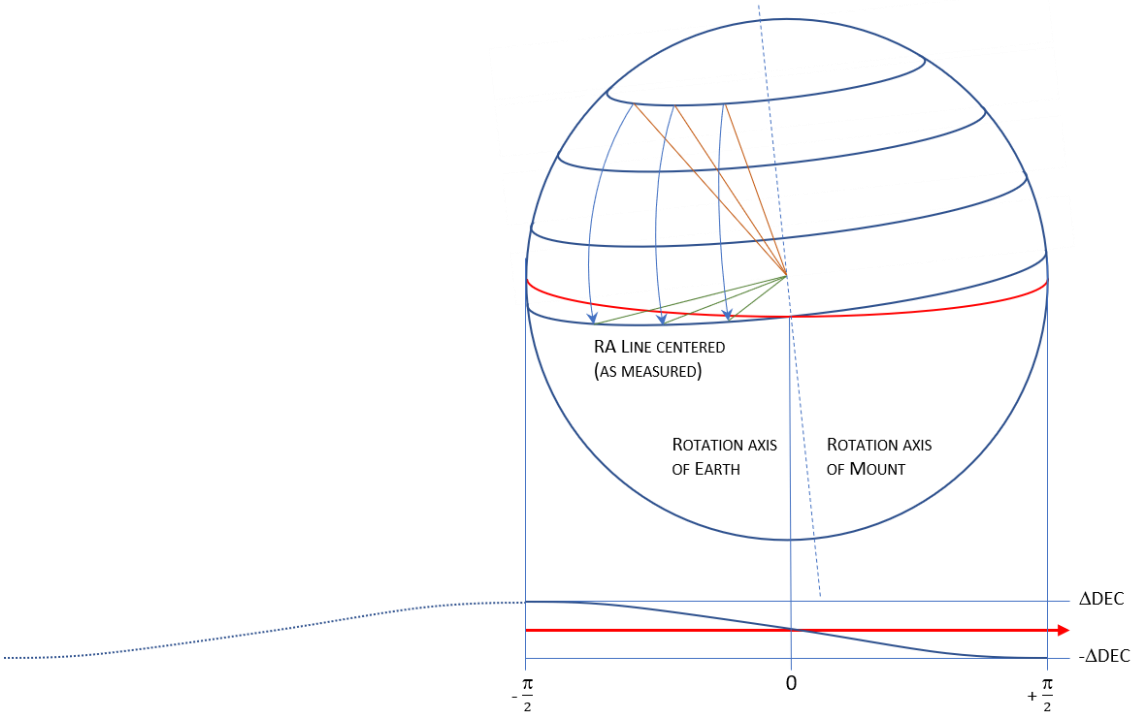
$$R(deg) = \frac{P \cdot (0.1594 + 0.0196a + 0.00002a^2)}{(273 + T) \cdot (1 + 0.505a + 0.0845a^2)}$$

Projection onto rotational plane:

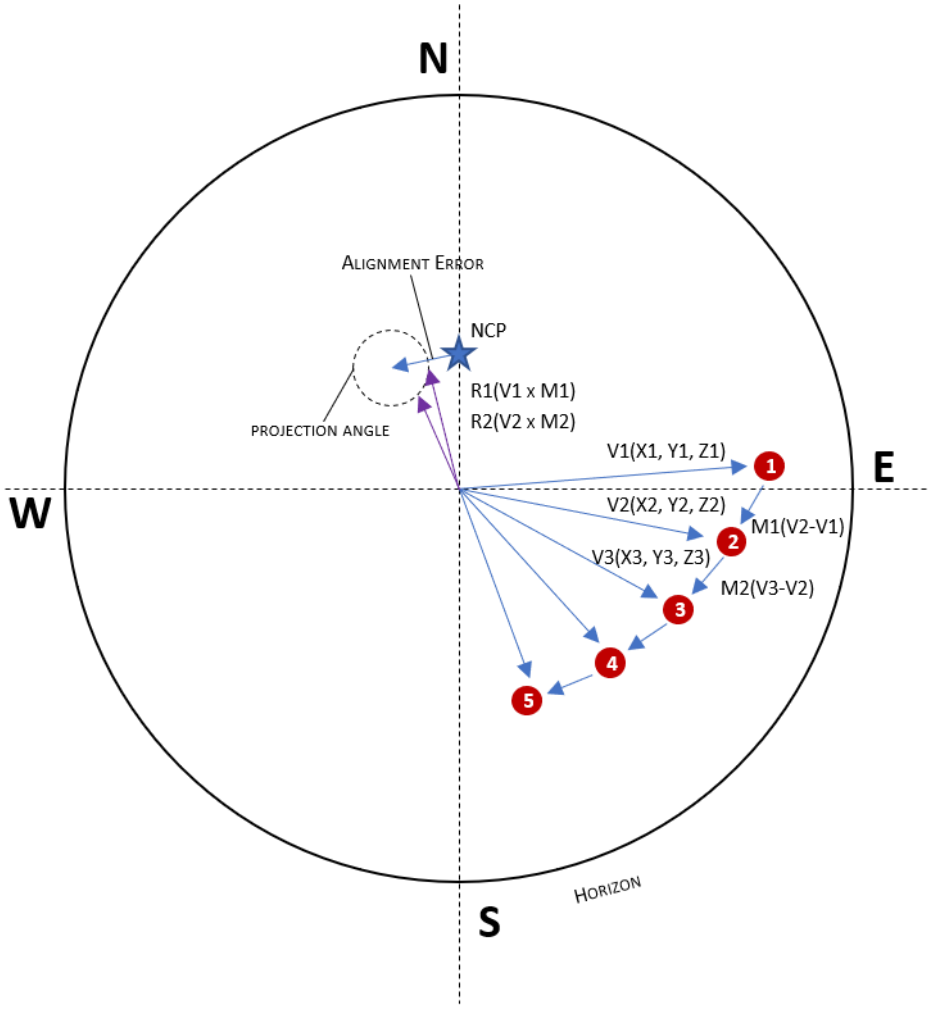
The angular distance to the rotational plane needs to be established. The meaning is that the rotational plane of the measurement has to be projected so that it is centered in the coordinate system. This allows the use of vector cross product math in the next step.
As we are rotating along the RA axis, the correction behaves like a constant DEC offset. Thank you polar coordinate system.



To visualize what is happening, see the following picture. The now projected measurement points describe a plane centered in the polar coordinate system. When we add the plane of Earths actual rotation, we see that these are at an angle to one another.
If we plot the difference between the two… it looks like a sinus wave.



To calculate the angular distance, we use the ‘Optim’ optimizer function in R. The underlying concept is, that each individual rotation vector for each increment collapses into one rotation vector, if the projection is applied.
Again, we transfer the polar coordinates, RA, DEC, LST and HA into vectors in Cartesian space.
Then the pointing vectors to each center of picture are generated.
By subtracting the vectors from one another we create the movement vectors.
Calculating the cross product of a pointing vector with its movement vector we get the vector along the rotational axis.
As we vary (optimize) the projection angle, the resulting vectors for the rotational vectors collapse into one. This is done with the root sum square (RSS) of each individual deviation.
The function in R returns the angle only.



Rotation vector and mount correction angles:

The target rotational vector (earths axis) is given from the Latitude information.
With the projection correction established, actual rotation vector is determined.
Next, we compare the actual rotational vector with the target vector.
From this we can calculate the Azimuth and Altitude corrections.

Repeatability /Accuracy:

Without modifying the mount or telescope setup, three measurement runs were performed:
The resulting correction angles from the individual runs are within 2 arcmin, which I consider to be quite good.

	DEC MEAN (DEG)	ALT COR (DEG)	AZ COR (DEG)
1	5.5	-1.408	-1.086
2	20.5	-1.412	-1.072
3	35.5	-1.412	-1.061
Deviations		0.004	0.025

Example:

The following is actual recorded data as an example.
Coordinate system:
x → east y → north z → up

Input data from plate solve:

	SIDEREAL	RA	DEC	ANGLE
1	19.36739	0.8864344	0.09171645	85.61
2	19.87199	0.7555315	0.09549766	85.62
3	20.37645	0.6246649	0.09922166	85.65
4	20.88144	0.4936618	0.10286705	85.67
5	21.38588	0.3627995	0.10634911	85.78
6	21.89122	0.2317001	0.10963149	85.83
7	22.39615	0.1007129	0.11267702	85.97
8	22.90118	6.2528796	0.11541381	86.10
9	23.40643	6.1218077	0.11780488	86.26

Refraction correction:

	ALT_DEG	refraction	SIDE_COR	RA_COR	DEC_COR
1	17.57899	0.04688032	19.36945	0.8858952	0.09233387
2	22.61615	0.03639342	19.87355	0.7551242	0.09598659
3	27.48945	0.02991615	20.37768	0.6243438	0.09963467
4	32.13120	0.02559132	20.88243	0.4934037	0.10323257
5	36.44048	0.02257574	21.38667	0.3625916	0.10668450
6	40.32089	0.02042040	21.89185	0.2315354	0.10994806
7	43.63405	0.01888864	22.39662	0.1005881	0.11298248
8	46.23927	0.01784106	22.90151	6.2527937	0.11571328
9	47.99413	0.01720069	23.40661	6.1217610	0.11810149

Resulting projection angle correction in deg:
-5.495

Resulting Pointing vectors (with projection corr):

	x0	y0	z0
1	0.9364108	-0.2634865	0.2317534
2	0.8821007	-0.3504703	0.3147521
3	0.8124328	-0.4313619	0.3922753
4	0.7285139	-0.5048211	0.4630585
5	0.6319918	-0.5694182	0.5256894
6	0.5242641	-0.6241933	0.5792494
7	0.4074844	-0.6680139	0.6226667
8	0.2835809	-0.7001899	0.6552221

Resulting Movement vectors (with projection corr.):

	x1	y1	z1
1	-0.05431010	-0.08698381	0.08299869
2	-0.06966795	-0.08089161	0.07752318
3	-0.08391886	-0.07345919	0.07078319
4	-0.09652215	-0.06459709	0.06263094
5	-0.10772769	-0.05477517	0.05355999
6	-0.11677966	-0.04382056	0.04341737
7	-0.12390355	-0.03217603	0.03255532
8	-0.12889403	-0.01996258	0.02113017

Resulting Rotation vectors without projection correction:

	x2	y2	z2
1	0.074374119	-0.7125356	-0.6976830
2	0.068392857	-0.7205827	-0.6899877
3	0.060934228	-0.7280375	-0.6828239
4	0.052312927	-0.7345983	-0.6764826
5	0.042518556	-0.7402946	-0.6709367
6	0.031598499	-0.7450642	-0.6662438
7	0.020076437	-0.7486810	-0.6626264
8	0.007922892	-0.7511695	-0.6600618

Resulting Rotation vectors **with** projection correction:

	x2	y2	z2
1	-0.01299191	-0.6860234	-0.7274635
2	-0.01297984	-0.6860070	-0.7274791
3	-0.01302637	-0.6860538	-0.7274342
4	-0.01295082	-0.6859960	-0.7274900
5	-0.01291350	-0.6859742	-0.7275113
6	-0.01303509	-0.6860276	-0.7274587
7	-0.01299293	-0.6860143	-0.7274721
8	-0.01300951	-0.6860177	-0.7274685

Resulting actual mount vector:

x	y	z
0.01300627	0.68599733	0.72748780

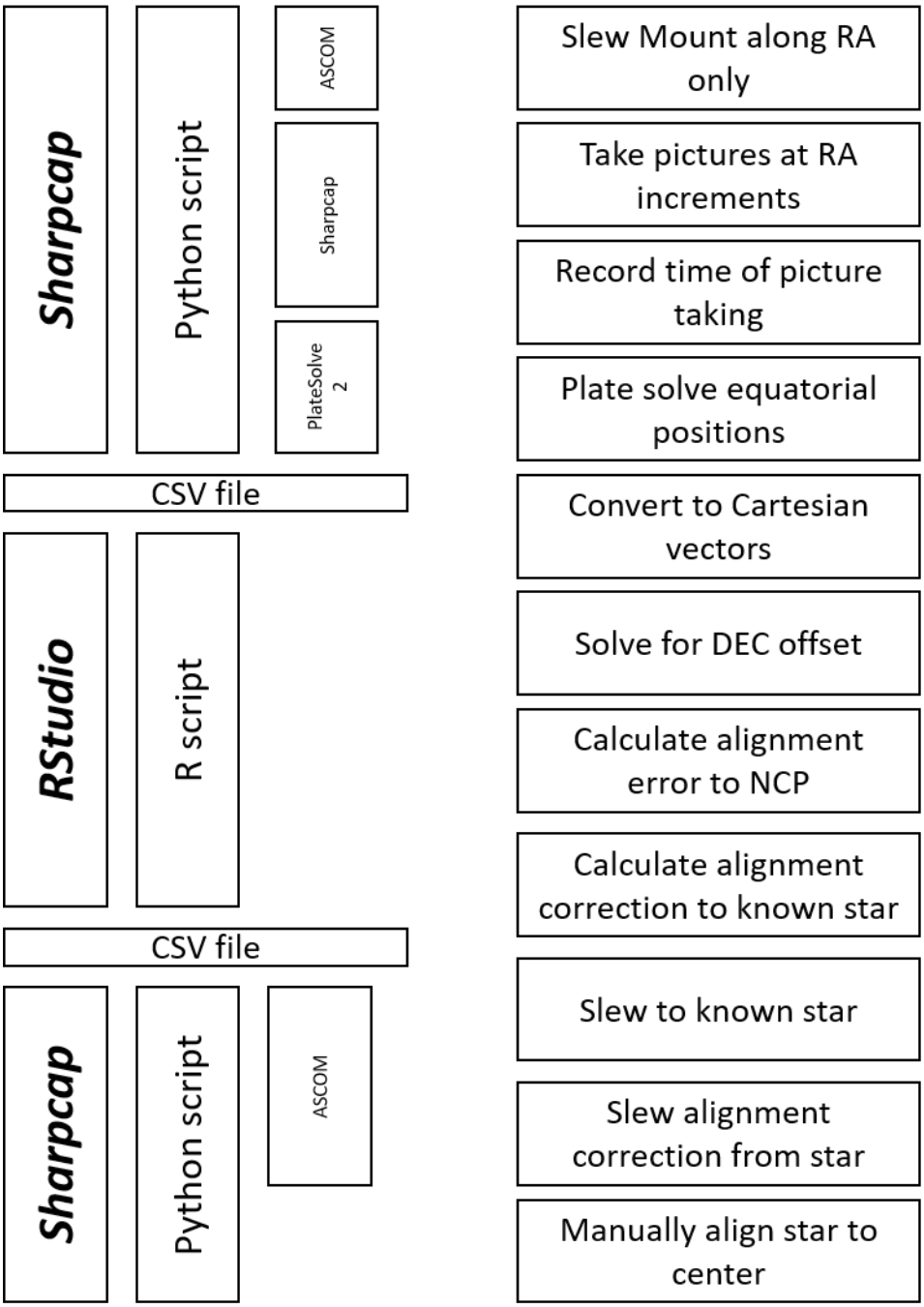
Reference mount vector:

x	y	z
0.00000000	0.66804038	0.74412502

Resulting Azimuth correction in deg:
-1.086

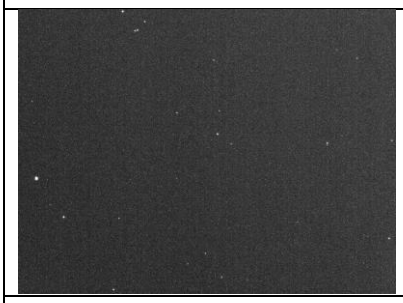
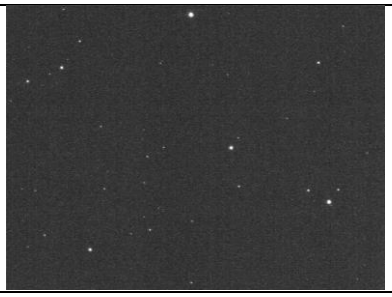
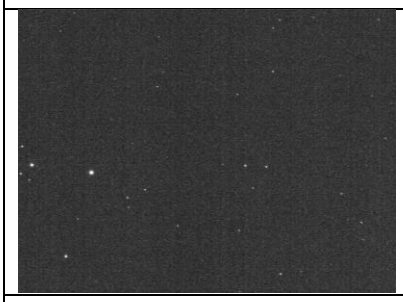
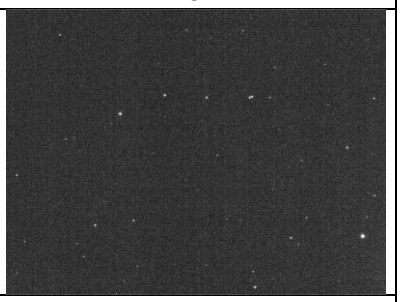
Resulting Altitude correction in deg:
-1.408

Program flow:



Pictures of Example:

The following is actual recorded data as an example.

1	2	3
		
4	5	6
		
7	8	9