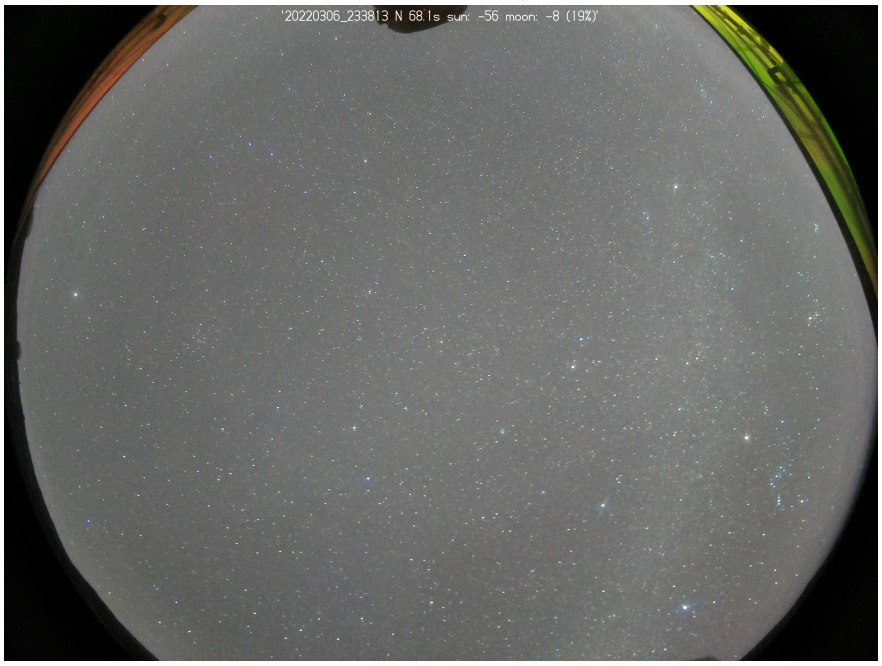
skyWATCH: Construction and Development of an Autonomous All-Sky Camera

Progress Report 2021/22

George Hume
ghume1@sheffield.ac.uk



Tuesday 27th September, 2022

1 Project Overview and Objectives

The skyWATCH (Sky Weather AsTro Camera tHingie) devices are autonomous all-sky cameras made inexpensively using Raspberry Pi microcomputers and off-the-shelf components. The devices take all-sky images during the night using a fish eye lens attached to a Raspberry Pi HQ camera [1]. They can also have a range of sensors which can be used to monitor the temperature of the surroundings, CPU, and sky, as well as the humidity and pressure.

Since CONCAM (the ING's previous all-sky Camera) was taken offline, the Isaac Newton Group (ING) does not have any all-sky monitoring capabilities at either of its telescopes (the INT and WHT), unlike other telescopes, such as the Liverpool Telescope or the Gran Telescopio Canarias (GTC) [2, 3]. Therefore, the skyWATCH devices, if officially acquired by the ING, would be a cost effective solution to this absence, allowing real-time and archival sky and weather monitoring.

There is a range of prototype devices used for different degrees of testing. The most complete of these, such as a device located in the WHT DIMM tower (codenamed *murdoc*[1]), are enclosed within repurposed transparent waterproof domes, originally intended for scuba divers' cameras. You can see the images and readings from any active skyWATCH devices at skywatching.eu. The main goal of the project has been to develop these prototypes designed and built by Richard Ashley[2]. Hopefully, these improvements will allow them to be presented to ING management to approve their official use.

The parts of the project that will be expanded on in this report cover three main areas:

- **A New Capture Script - Section 2.1**
  This regards the creation of a new script for capturing all-sky images. It needed to replicate the outputs already being produced on the skyWATCH devices, but instead of taking an image at a fixed cadence the requirement was for it to expose continually (i.e., as soon as one image is captured the next starts).
- **Dome Detection - Section 2.2**
  The second aim, specifically related to the *murdoc* skyWATCH device, was to develop a method to detect when the WHT DIMM tower dome was closed above it. If the dome was detected to be closed then this could be reported and the capture cadence reduced to save resources.
- **Image Overlays - Section 2.3**
  The final aim was to overlay constellation maps onto the captured all-sky images (like in the Liverpool telescope's all-sky camera) [2]. This would involve finding the mapping between the positions of targets on the sky and their positions on the all-sky images. Using this information the pointing of the WHT could then also be overlaid onto the all-sky images (like in the GTC all-sky camera) [3].

How all these parts can be expanded and improved on is discussed in **Section 3**.

2 Progress

2.1 New Capture Script

As explained in **Section 1** the new capture script needed to replicate the output of the original skyWATCH script but also capture constantly during the night-time. It was chosen to write the script in *Python* as this had the best compatibility with the Raspberry Pi HQ camera, and was what the previous script was written in. The latest version of the script is available on GitHub.

When first running the script it will ask for the device name, location name, latitude, longitude, elevation, and format the images will be saved as (JPG or PNG). These parameters are then saved as a JSON file called `setup.json` and if the script detects that this is already present it won't ask for this information again.

Once the script is running it will check the altitude of the sun every minute using the Python library `skyfield` along with the latitude, longitude, elevation and UTC time [4]. If the sun is over 6 degrees below the horizon (i.e., the end of civil twilight in the evening, or the start of it in the morning), it switches to

[1] *murdoc* is named after Murdoc Niccals the bassist of the animated band Gorillaz.
[2] Richard can be contacted via rashley@ing.iac.es.

'night-time' mode. This mode starts capturing the all-sky images, but if the opposite is true then it remains in 'day-time' mode, which involves not exposing and continually checking the sun's altitude.

The capture process involves using the `raspistill` imaging software to take the all-sky images which can be saved as a JPG or PNG [5]. To obtain all-sky images which are not under or over exposed a simple auto-exposure method is used to predict the exposure time. This method takes a low resolution (825x640 pixels, or 0.5 MP) temporary image and calculates its median pixel value, $\tilde{x}$. If the median value is not within a specified upper and lower bound, then the ratio of half the max pixel value (255) and the median all to the power of 1.6 is calculated[3]. The product of this and the previous exposure time (t_n) is then used as the new exposure time (t_{n+1}), i.e.,

$$t_{n+1} = t_n \left(\frac{128}{\tilde{x}} \right)^{1.6}$$

However, the maximum value of t_{n+1} is set to 90s to prevent stars from trailing much. Otherwise, if the median pixel value is within the bounds, the exposure time remains the same.

A final full-resolution image (4056x3040 pixels, or 12.3 MP) is captured using the new exposure time and is annotated with the device's name and location, the UTC time of the capture, and its exposure time. An example of one of the all-sky images captured using this script running on *murdoc* is visible in **Fig.1**. The image is captured in greyscale, which is used to remove a pink hue present in the long exposure images. This is due to the removal of the infrared filter from the Raspberry Pi HQ camera which allows more photons to be captured, therefore improving the camera's night-time performance.

Along with each image captured a JSON file is saved with the same name and contains: the device's name; the UTC time of capture; the image's filename, exposure time, width, and height; the location's name, latitude, longitude, and elevation; the time of day (either nautical twilight, astronomical twilight, or dark time); the altitude of the sun; and the moon's altitude, phase, and illumination. The latter are also calculated using the `skyfield` Python library [4].

During 'night-time' mode, a log is produced indicating when all these actions occur and the filename of each image captured is saved to a chronological list. At the end of the night, a time-lapse video is produced and saved using the images from the night. This can also be reproduced later using the list if the video failed to be produced.

2.2 Dome Detection

One of the objectives specific to the skyWATCH device *murdoc* was to detect when the WHT DIMM tower dome is closed above it. The method which was used for this is called *Template Matching* and uses the Python library `openCV` to achieve this [6, 7]. The principle behind template matching is to match a few standard templates of the DIMM dome to an image taken by *murdoc*. If there is a good match then it is likely that the dome is closed. The templates in question are sections of a standard image of the DIMM dome taken by the *murdoc* which had pronounced features (such as the gaps or joins between the dome sections).

Before any matches can be made the all-sky images are processed to a similar contrast by equalising a histogram of the pixel values of each image [8]. To extenuate the features that need to be matched between the templates and the image, the pixel values of the image were inverted and then clipped so all the pixel values below 150 (where the maximum pixel value is 255) were set to zero. This meant that any final post-processed images of the DIMM Dome were mostly black with the features highlighted in white. This same processing had to also be done preemptively to the templates so they would match with the images after post-processing. You can see the processed templates used overlaid in their correct positions on top of an un-processed image of the dome in **Fig.2**.

The algorithm that performs the template matching is from `openCV` and outputs the coordinate of the top left corner of the section on the all-sky image which it predicts the template matches with [7]. If these coordinates are within a pre-defined set of limits (centred on the average template coordinates predicted from a large set of dome images taken with *murdoc*) then they are said to match.

[3]This relationship was found by fitting a power-law curve to a scatter plot of the median pixel values of images taken using the Raspberry Pi HQ camera at a range of exposure times.

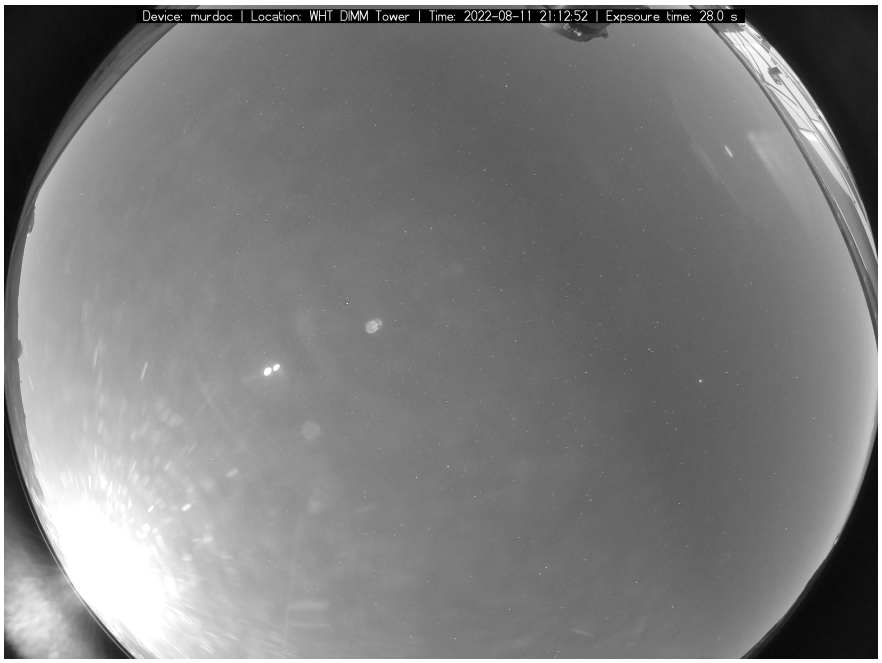


Figure 1: *An all-sky image taken using the new capture script during a test on* ***murdoc*** *on the 11th August 2022. You can see the information about the image in the annotation at the top.*

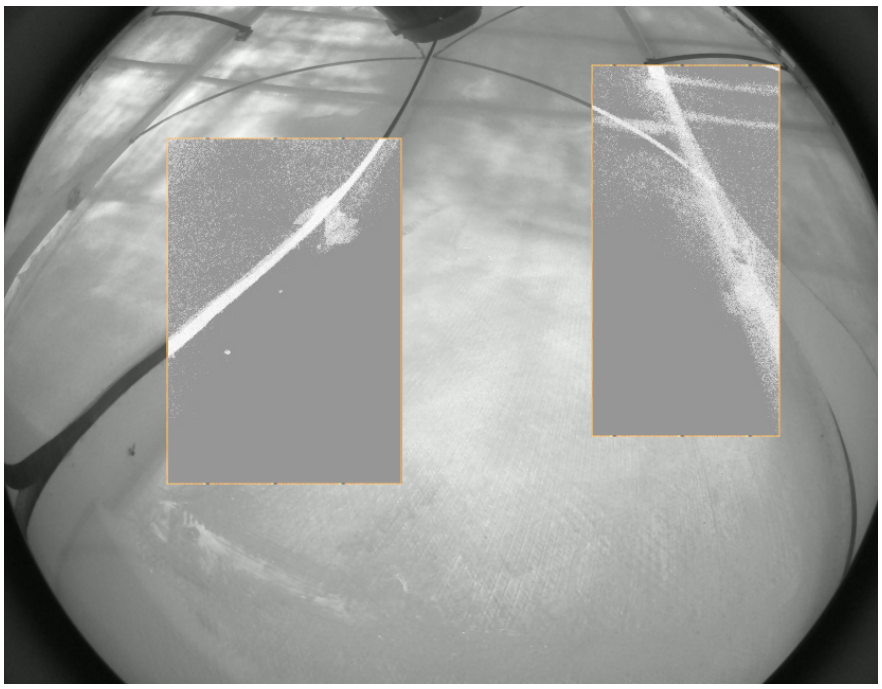


Figure 2: *An image of the WHT DIMM Tower dome taken on* ***murdoc*** *with the processed templates used in the dome detection overlaid.*

After testing this dome detection method on multiple sets of images taken on different nights from *murdoc* a few improvements to the method were made. First, to reduce the number of false positives and false negatives (i.e., detecting the dome is closed when it is open, and detecting the dome to be open when it is closed respectively), both templates have to be matched when going from open to closed, but either

can match when staying closed. This was implemented because, in testing, it was seen that sometimes one template could match with images of the sky causing false positives. Therefore, this criterion was introduced as the chance of both templates matching on an image of the sky is much lower. In contrast, when the dome is closed sometimes the brightness is not enough for both templates to be detected, causing false negatives, so the decision was taken to reduce the criterion when the dome was closed. To further reduce the number of false results, if the predicted location of the templates on the image caused the templates to overlap then the result would be false (i.e., the dome is open), as this is not physically possible.

Another event observed during testing was that sometimes the templates were not matching even though they were very close to being within the limits. To try and negate this, after a positive match the centre of the matching limits for that template would be updated to match these newly predicted coordinates if, and only if, the overall result was positive (i.e., the dome is detected to be closed). Therefore, in theory, these limits could dynamically change and therefore hopefully reduce the number of near misses that could cause false negatives. However, currently, not enough tests have been implemented using these dynamic limits to evaluate their effectiveness.

2.2.1 Integrating dome detection into the new capture script

Most of the initial development behind the dome detection method used pre-existing all-sky images from *murdoc*, but after this, the majority of its testing and development was when it was an integrated part of the new capture script (see **Section 2.1**). This meant that when 'night-mode' was active, before capturing a full resolution image, a temporary lower resolution image (0.5 MP/825x640 pixels, compared to the full resolution of 12.3 MP/4065x3040 pixels) was captured without any annotations. This temporary image was then used within the dome detection function to predict if the dome was open or closed. If the function predicted that the dome was open then the capture script would continue and capture the full resolution all-sky image. If not, then a placeholder image is made by overlaying text indicating the dome is closed along with the device's name, the location, and the UTC time onto the low resolution image (see **Fig.3**). This placeholder is saved and is used in the time-lapse instead of a blank frame. The fact that you can see the image behind the text can be used to double-check that the prediction of the dome detection method is correct. After creating the placeholder, there is then a minute pause in operation before the script loops and checks if the dome is closed again.

2.3 Image Overlays

One of the key aims of the project was to try and overlay stars, constellations and telescope pointings (mainly of WHT) onto the all-sky images. To do this the positions of objects on the sky needed to be translated to their position on the image accounting for any distortion caused by the Raspberry Pi HQ camera and its lens. Three different methods were attempted to achieve this during the year, however, unfortunately, none achieved the necessary accuracy.

2.3.1 Camera calibration

The first method involved using the Python library `openCV` to calculate the camera parameters of the skyWATCH camera [6, 9]. These are such properties as the focal length and optical centres which can be contained within a matrix. This matrix along with any rotations or translations performed on the image can be then used to account for the distortion of the image. These coefficients can be calculated using a range of images of a well defined reference pattern (in this case a chessboard pattern) from different angles and distances. The resulting matrix can then be used to undistort images taken with the camera, see an example in **Fig.4**.

The plan was to use the matrix made in the camera calibration to do the reserve and distort a sky map to match the distortion of the all-sky image. The sky map could then be overlaid onto the all-sky image and, in theory, the positions of the constellations on it would line up with the correct stars in the image. This sky map was sourced from *VirtualSky*, where an HTML sky map can be created for a specific location and time [10]. So, using a Python script, a query to the site was made to obtain the HTML map and then convert it into an image using the Python library `imgkit` [11]. The image of the map was then cropped to match the dimensions of the all-sky image and finally, transformed to match the distortion of the all-sky image using

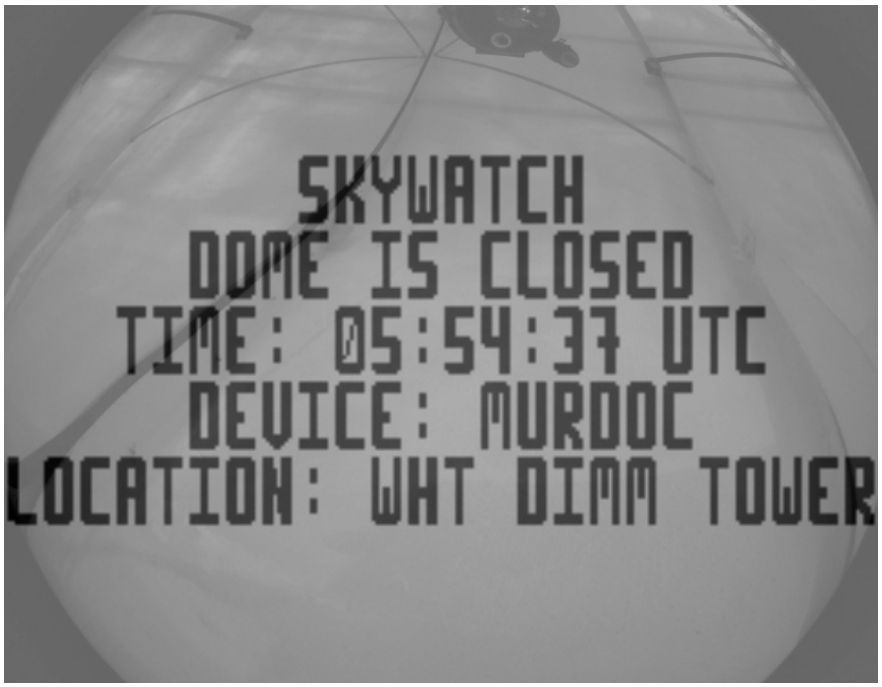


Figure 3: *An example of one of the placeholder images produced by the new capture script when the WHT DIMM Tower dome is closed.*

the inverse of the camera calibration matrix. Unfortunately, each time this was performed and the distorted map was overlaid on the all-sky image, the constellation maps and real stars never accurately lined up.

2.3.2 Polynomial transformations

The second method explored was to use polynomial transformations of the sky map to match the all-sky image. This involved finding the x and y coordinates of around 40 corresponding stars in a reference all-sky image and a corresponding sky map from *VirtualSky*. Relationships between the corresponding x and y coordinates of these stars in the image and map were found using second-degree polynomial fits which can be seen in **Fig.5**. These two polynomial functions could then be applied to each of the pixels in the sky map to change their position to match the reference frame of the all-sky image. The newly transformed map was then overlaid onto the all-sky image, but, again, the positions of the constellations and the stars, unfortunately, did not match. This is likely due to not accounting for the outliers and not using enough stars for accurate fits (especially for the y coordinate fit - see the right of **Fig.5**).

2.3.3 Polar transformations

The third method attempted did not involve using any sky maps from *VirtualSky*, instead, it involved manually extracting the x and y coordinates of over 150 stars from a reference all-sky image. The x and y coordinates were then transformed into polar coordinates with the origin at the centre of the image instead of the top left. This transformation was performed under the assumption that the polar coordinates - radial distance (r) and polar angle (ϑ) - could more easily be fitted to the altitude and azimuth of the stars. The altitude and azimuth were calculated using the Python Library `skyfield` from the Right Ascension (RA) and Declination (Dec) of the stars (which could be easily found as most of the stars were part of constellations) along with the time and location the image was taken [4].

As an all-sky image can be thought of as a 2-dimensional projection of the sky onto the ground, the geometry of the celestial sphere can be used to map between the altitude and azimuth, and r and ϑ. Both the azimuth ($az.$) and the ϑ are angles measured from the zenith in the plane of the image, so are almost analogous ($az. \sim \vartheta$). The altitude, however, is the angle from the horizon to the star. If the sky is considered

Figure 4: *The image on the left is an image taken with a fish-eye lens. The image on the right uses a matrix found during the* `openCV` *camera calibration to undistort the left image. Note that the undistortion crops the full field-of-view of the original image.*

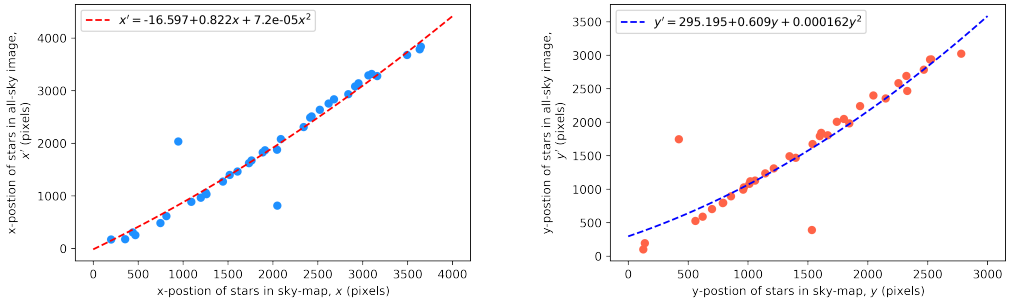


Figure 5: *A polynomial fit of the x coordinates of stars in the all-sky image and sky map (left) and the same for the y coordinates on the right. Note there are a few outliers and, to the eye, the y coordinates line of best fit does not appear as good as the corresponding for the x coordinates.*

a hemisphere that meets earth at the horizon then $r \sim d\cos(alt.)$, where d is the distance to the horizon from the centre of the image and $alt.$ is the altitude.

Instead of applying these relationships described above exactly a linear fit was calculated between $az.$ and ϑ and a sinusoidal fit between the $alt.$ and r. These fits were then used to predict the x and y positions of 5 stars on a different test image taken with the same device from their RA and Dec. You can see the results of one of these tests in **Fig.6** and, unfortunately, the predictions of the positions of the stars were not accurate enough. This is likely due to the method not accounting for the image distortion properly. However, this method does seem to have the most potential as it would allow for the pointing of WHT to be easily displayed on the image as it can be directly converted from RA and Dec to x and y image coordinates. Maybe in conjunction with one of the other methods or doing fits where r depends on both the azimuth and altitude, this could greatly improve the accuracy by accounting for the image distortion.

3 The Future

As is evident from the previous sections, there are still a lot of aspects relating to these objectives that need to be built upon and developed before they can be considered to be finished. This section aims to discuss these shortcomings and suggest improvements that could be made in the subsequent year by any new students that join the project.

Currently, the new capture script is still in the testing phase, but these tests are mainly related to refining the dome detection part of the software using *murdoc*. Unfortunately, at the end of this academic year, it was not possible to connect to *murdoc* as the device fell off the ING network, so the latest version (version 0.8) of the new capture script, which does everything that is described in **Sections 2.1 & 2.2**, could not

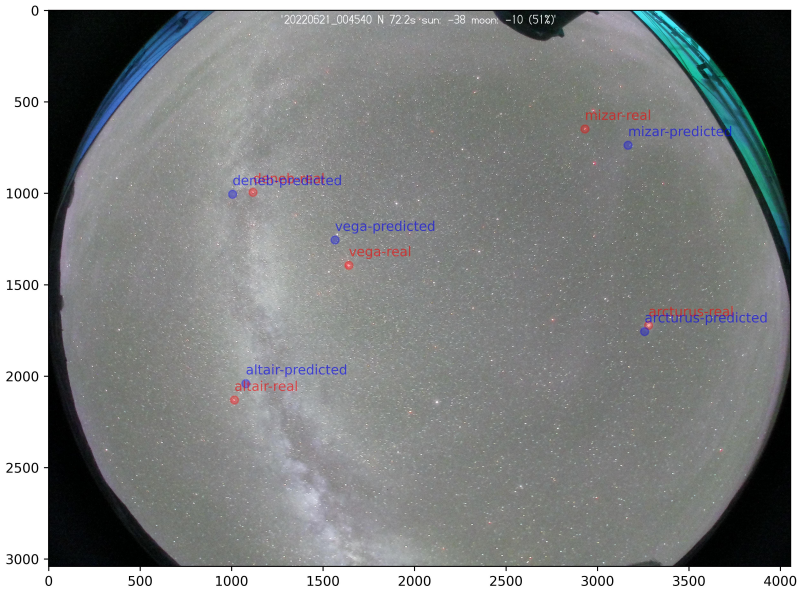


Figure 6: *An attempt at trying to predict the position of stars in an all-sky image using best fits between the azimuth and altitude of stars to their radial distance and polar angle in the all-sky image. You can see the predicted positions in blue and the real positions in red.*

be tested on it. Thankfully, this issue has now been resolved, so others can hopefully resume this testing. If the tests of version 0.8 are successful then it, or parts of it, can be implemented into standard skyWATCH software. Otherwise, further adjustments will be needed to refine the dome detection method. A useful addition to the new capture script could be implementing a method to reduce the overheads when capturing full resolution, long exposure all-sky images. This could be done by using the new Raspberry Pi camera software included in the latest Raspberry Pi OS (Bullseye) called `libcamera`, which is set to be the standard moving forward [12].

The objective that has raised the most issues is definitely the implementation of image overlays onto the all-sky images, as all the methods used (camera calibration, polynomial transforms and polar transforms) could not achieve the necessary accuracy. Issues regarding the camera calibration method may be due to the distortion matrix that was used to undistort images from *murdoc* was obtained from the camera calibration of a skyWATCH device which was believed to have the same camera set-up as *murdoc's* but it may not have been completely identical. Additionally, improvements to the accuracy of the polynomial transform method could be made by using a larger amount of stars and removing any outliers.

The most promising technique explored for these image overlays was the polar transform method as it does not rely on an external service for its overlays (*VirtualSky*) and can instead directly map a value of RA and Dec onto the image, making it very useful for indicating the pointing of the WHT (however a method to query its pointing still needs to be devised). However, the testing of this method also shows inaccuracies between the predicted positions of the stars and their actual positions. Hopefully, with a larger number of reference stars over multiple nights, a better fit could be found. Further improvements could be made if the fitting of the radial distance (r) and the polar angle (ϑ) both depend on the azimuth and the altitude of the stars as this should account for any image distortion. Once this is working to the necessary accuracy, maps of the constellations would have to be created using the RA and Dec so they can be overlaid onto the images, otherwise, the method would be limited to showing only telescope pointings and star positions.

4 Conclusion

In summary, this report discusses the development of a new capture script for the skyWATCH devices, the creation of a technique to detect when the WHT DIMM tower dome obscures the *murdoc* skyWATCH

device's view of the sky, and the exploration of different methods to overlay stars, constellations, and telescope pointings onto the all-sky images that the devices capture. The success has varied between these aims; the first and second are almost operational (only a few more tests are needed to refine the dome detection), but the third needs more development as all three methods explored have failed to produce the accuracy necessary for these overlays. Hopefully, in the next academic year, the new capture script and dome detection can be implemented into the standard skyWATCH software, and the polar transform method of overlaying, which has the most promise, can be built upon until the necessary level of accuracy is achieved.

References

[1]Raspberry Pi Foundation, *Raspberry Pi High Quality Camera*, https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/ (visited on 09/06/2022).

[2]Astrophysics Research Institute - Liverpool John Moores University, *SkyCam*, https://telescope.livjm.ac.uk/TelInst/Inst/SkyCam/ (visited on 09/06/2022).

[3]GRANTECAN S.A., *Webcams at the GTC*, http://www.gtc.iac.es/multimedia/webcams.php# (visited on 09/06/2022).

[4]B. Rhodes, *Skyfield: High precision research-grade positions for planets and Earth satellites generator*, July 2019.

[5]Raspberry Pi Foundation, *Raspicam commands*, https://www.raspberrypi.com/documentation/accessories/camera.html#raspicam-commands (visited on 06/22/2022).

[6]G. Bradski, "The OpenCV Library", Dr. Dobb's Journal of Software Tools (2000).

[7]OpenCV Team, *Template Matching*, https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html (visited on 05/31/2022).

[8]linuxtut, *Normalize image brightness*, https://linuxtut.com/en/9c9fc6c0e9e8a9d05800/ (visited on 06/01/2022).

[9]OpenCV Team, *Camera Calibration*, https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html (visited on 05/31/2022).

[10]Las Cumbres Observatory, *VirtualSky*, https://virtualsky.lco.global/embed/custom.html (visited on 04/26/2022).

[11]jarrekk, *IMGKit: Python library of HTML to IMG wrapper*, https://github.com/jarrekk/imgkit (visited on 09/06/2022).

[12]D. Plowman, *Bullseye camera system*, (Nov. 2021) https://www.raspberrypi.com/news/bullseye-camera-system/ (visited on 09/06/2022).

The majority of the code referred to in this report can be viewed on github.com/george-hummus/skyWATCH.