The Raspberry Pi and amateur astronomy

First published by Astronomy Technology Today

Background to the Raspberry Pi.

Back in 2009 a team at Cambridge University's Computer Laboratory decided to do something about the decline in Computer Science. They set up a charity to promote basic computer science in schools and then to develop access to low-cost computers. The first Raspberry Pi was launched in 2012.

With sales now approaching 40 million units and the computer now in its 4th generation, it is certainly achieving the original objectives. The success comes from its low price, wide versatility and reasonable processing power.

Raspberry Pi 4B

Amateur astronomers have been quick to find many applications for the Raspberry Pi, and many of these have taken advantage of its low power consumption and the ease of adding hardware.

This ability to bring a small computer right up to the telescope and mount, even in a field, has enabled new techniques especially in astrophotography. Commercial manufacturers have recognised this, and a number of mount side computers are now available, some even incorporating the Raspberry Pi as the processor.

Key to its uptake has been the relative ease of programming the computer, primarily using Python with a growing, open source and free repository of previously written code modules. The Raspberry Pi was also designed to allow the easy addition of hardware such as switches, sensors, LEDs and other displays.

ScopeDog

After a career in science and engineering I had built an observatory, a couple of telescopes and many accessories. Being an 'early adopter', the Raspberry PI was attractive, and I had played with a couple over the years, but no serious projects.

A few years ago, I decided to make a Dobsonian mount for my 18" mirror set that I had been using on a driven equatorial fork mount in my observatory. Having the new Dobsonian mount track was a priority. Craig Colbert, who I had met at the Texas Star Party had the same desire and so we set about a joint project. I would do the mount and telescope design and he would do the software.

While I designed and made my telescope, Craig set about the task of coding. As an IT professional this wasn't too difficult for him, but as he lived 5500 miles away in Santa Monica, bringing the hardware and software together was going to be challenging. Once we

had chosen the encoder system and drive stepper motors, we were both able to make reasonable progress separately. At about this time we realised that a full ‘GoTo’ solution was very little more complicated than a tracking mount, so that became our new goal. Although stepper motors can be driven from the Raspberry Pi with very little additional circuitry, we decided to use proprietary driver modules from Phidgets. This would put the cost up a bit but would greatly simplify the task and additionally provide protection to the motors and circuit boards.

We chose to use the Nexus DSC product from AstroDevices to drive the encoders and communicate with a tablet computer running SkySafari, which would be our primary telescope control mode. Our telescope controller, now named ‘ScopeDog’, would communicate with the Nexus DSC to get current telescope position and any ‘goto’ commands being issued. The main task of the ScopeDog computer would be to convert the RA & Dec output by the Nexus DSC into Azimuth and Altitude, and to calculate and command motor drive speeds in these axes. This turns out to be quite a computational task which has to be done continuously at some speed. The telescope was to have a small hand control box and this needed to be communicated with too.

ScopeDog control box with lid removed

Craig soon established that a Raspberry Pi would be capable of achieving the workload, but it would need to run fast and so he chose Java as the program language. While he developed the code, I built my new Dobsonian mount and added the Nexus DSC and the motors with their drive belts. I also built a ScopeDog control box, complete in every way except for no code! However, Phidgets provide some coded test routines for their modules and I was able to use these to make sure the motors would move the telescope satisfactorily.

My telescope was the first built and so the integration of his code and my mount would be on my telescope in the UK. At first progress was difficult, but we learnt how to communicate better, needing to describe problems and solutions with more clarity. Video calls were a great help and often I would be holding my phone camera next to the motors so Craig could see and hear exactly what his code was doing on my hardware.

Original ScopeDog hand box mounted near the eyepiece

It took about a month to solve the main bugs and get a complete system running well enough to use for extended periods. It wasn't all fixing bugs, as we also added new features and refinements during this time, and for a while afterwards. An example of this is the action of the hand-box joystick. At fast and medium slew speeds the telescope moves in the direction the joystick is moved. When slow speed is selected, we realised we would be looking through the eyepiece and so for this speed only we reversed the azimuth direction, so the motion is correct when viewed through the telescope including its diagonal secondary mirror.

We went further and made a large number of set-up parameters easily available to the user via a web browser, (the Raspberry Pi is set to generate its own Wi-Fi hot-spot). These include telescope slew speeds, gear ratios and drive directions, joystick characteristics, and motor current.

ScopeDog and accessories mounted on my telescope base

The telescope slew is achieved by careful acceleration and deceleration of the motors which makes for a satisfying and quiet experience with minimal stress on the telescope mount. GoTo performance is generally within a few arc minutes, being limited by the initial two star alignment and telescope mount inaccuracies and flexure.

My friend has now built his telescope, (which I had the pleasure of 'borrowing' at a Texas Star Party!). Ours have been running very well for about 6 years, and I have built a few more ScopeDog systems for colleagues after they saw mine in action.

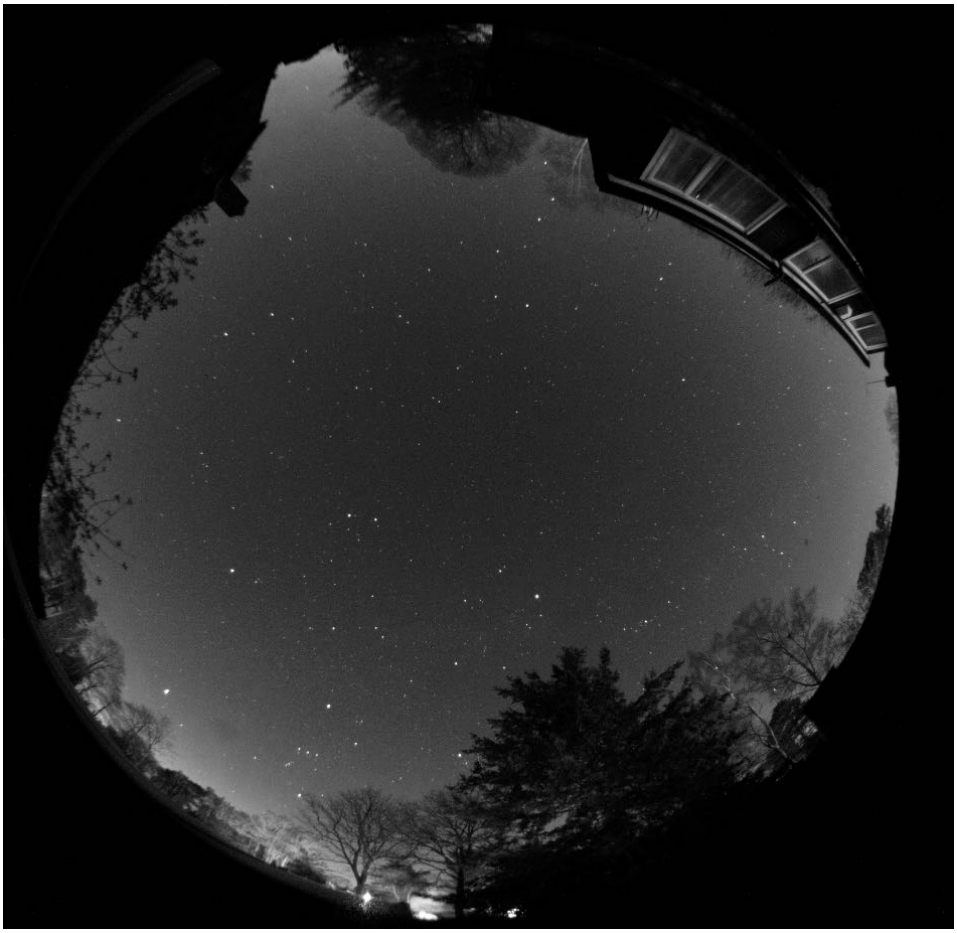
All-Sky Camera

My second Raspberry Pi project is a little more recent. Browsing the vendors at the last Kelling Star Party in the UK, I came across Allsky Optics who were offering parts and kits to make an all-sky camera. I already had a suitable ccd and so I bought an acrylic dome and a heater to make one for myself. This didn't take long, but I soon found that the ideal mounting position at home was too far from my desk to easily use USB, but someone suggested using a Raspberry Pi as a 'server' by the camera and using wifi to communicate back to my desk. This then triggered a whole research cycle, where I belated learnt what many astro-photographers have known for years. Having a small computer on the telescope mount can be very convenient, simplify connectivity and provide more functionality. My preferred desktop package to manage the all-sky camera was to be ALLSkEye which has many useful features, not least automatic meteor detection. This led me to base my remote Raspberry Pi server on the Indi/Indigo standards. Indi stands for 'Instrument Neutral Distributed Interface' and provides a bridge between computers and equipment, either locally or over a network, and for Windows pc's, Mac or Linux computers.

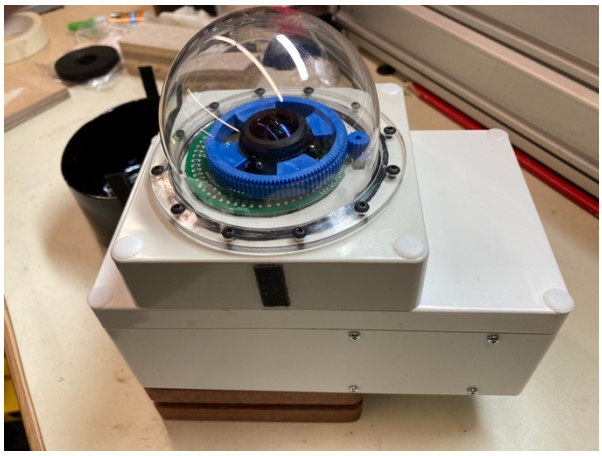
All sky camera mounted on roof

Setting up a Raspberry Pi as an INDIGO Sky server has been made very simple, taking literally only minutes. I installed the Raspberry Pi in my camera box and connected the ccd and powered up. Fortunately, there was just enough signal from my home wifi LAN reaching the camera box and I was able to set up the Indigo server to connect to my home LAN. Back at my desk I was delighted to find that almost every astronomy program I ran, it was able to discover the Indigo server and see my ccd attached.

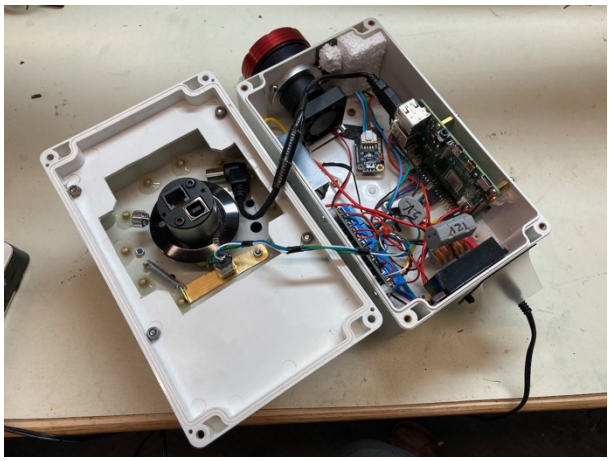
That night my all-sky camera did its first all-night run, working perfectly. Whilst the software and camera were working very well, over the next couple of days I discovered setting focus was critical and also there was a problem with condensation. I rigged up a simple motor driven focuser and used the Raspberry Pi GPIO pins to send ½ second long pulses to the motor. I could now sit at my desk and using the built in Indigo GPIO driver to accurately set focus on a live image. I added a fan to the camera box to help distribute heat from the dome heater and the electronics. These are also remotely controllable.



All-sky camera image



All-sky camera , showing heater and focus drive



Inside the all-sky camera housing

The all-sky camera has achieved two goals. I can keep an eye on the cloud cover, whether at home or at a star party and have no excuses for missing clear skies. Also, if I'm busy, or just not up to observing, I can let the camera watch for meteors.

eFinder

Many astro-photographers are now routinely using plate-solving to help align their telescope with exactly the target they want. The telescope is slewed to the target and an image captured. A computer then analyses the image and works out exactly where the telescope is pointing by comparing star patterns with reference index files. The telescope can then be adjusted if necessary so as to centre the object on the ccd. This feature is increasingly being built into astrophotography programs, and many commercial goto telescopes now use this technique to completely automate the two-star alignment process. Much of this is possible through the advances in the processing power of small battery powered low-cost computers.

Browsing the Cloudynights forum on-line I came across a couple of projects where amateur astronomers had built their own systems based on Raspberry Pis. This was intriguing and with the third Covid lockdown just starting I needed a new challenging project to keep me occupied.

My current favourite style of observing with my 18" Dobsonian is to see how far I can push the limits. I will deliberately pick targets that any sane person would consider impossible and spend perhaps an hour seeing if I can observe it (or not!). My scope encoders and drive will get me to within 15 arcmins typically, the errors arising from inaccuracies and flexure in the mount and telescope. Usually, I need to use high powers to see the faint and/or small objects and being sure I'm looking at the right field can take up a lot of time. Having seen the object (or not) I'll need to widen the field of view so I can double check I am seeing the target, and not a 'decoy'. Wouldn't it be nice if my scope would point *exactly* at the target, giving me the confidence to spend more time experimenting with filters, magnification, etc!

There were a lot of unknowns – how long would the exposures need to be, what aperture finder scope would I need, how sensitive need the ccd camera be, and how long would the plate-solving take? These are all interrelated and so I started by building a test rig out of parts I already had. Using the inevitable Raspberry PI as the core, I found that there was a very good ready built suite of programmes under the Astroberry name. Aimed at astro-photographers it includes a planetarium program, mount and camera controllers, image sequencing tools and importantly, the excellent Astrometry.net plate-solver is included.

eFinder camera & Raspberry Pi

The ccd from my all-sky camera (a QHY5L-M-ii) would provide a sensitivity reference point, but I also wanted to try out the new Raspberry HQ ccd. This was a very affordable 12MP camera ideally suited to pairing with the Raspberry Pi, but it does have very small pixels and hence might not be sensitive enough. Worth a try through as I would always find another project to use it on.

My first prototype wasn't very pretty but it worked well enough to capture a range of images with varying exposures. From these images I used a photo editing application to generate an even bigger range of test images with varying field of view and numbers of stars. These I fed into the plate-solver. I was surprised by the results. Given that I can seed the plate solver with the approximate position of the image, I found that with a 50mm aperture and 1 degree field of view I could typically solve in around 2-5 seconds. This was possible after exposures of only 2-3 seconds with the QHY ccd and 5 seconds with the RPi HQ camera, although that was with 4x4 binning.

Spurred on by these results I set about building a fieldable system. It had to be rugged, stable and resistant to dew. It would also need to communicate with the rest of my ScopeDog telescope control system on the 18". I also didn't want to rely on a computer screen for it to work, (I am a visual observer after all!), so I would need to write my own computer program to control the camera, launch the plate-solver, calculate required telescope offset, and communicate with the ScopeDog telescope drive. I had a little programming experience, but none with modern object orientated languages. I dived straight in though using Python, using the plenty of on-line resources available.

My code started off being disorganised and inefficient, but after a couple of weeks of learning I did a major rewrite, and the finished program now works well enough. The core of the code is a fast loop that scans the control panel buttons and refreshes the small red alphanumeric display. When commanded the code accesses sub-routines to capture an image, read telescope position, plate solve, convert RA & Dec to Az & Alt, calculate telescope pointing error, and finally to command the scope to move to close gap. This sequence is automatic.

The small display can be scrolled through to see results and to view and change various parameters. It defaults to a simple display showing the difference between scope actual position and required position (from SkySafari & the Nexus DSC), in arc mins. One button, the 'select' button is illuminated and has three key functions: a short press results in a measurement and display of scope position error, but doesn't move the scope, a little longer press causes the scope to move to position and a fresh measurement of scope position displayed. A really long press commands the eFinder to work out where it is pointing assuming no input from the telescope encoders – like a "where am I". This requires the plate solver to potentially check all 3Gb of index files and can take a minute or so.

Taking breaks from struggling with writing good Python code, I assembled the Raspberry PI and HQ camera into a plastic ABS box with all the connectors I would need. I combined the eFinder display and buttons with my original ScopeDog controls into a new hand box to be mounted at the eyepiece.

eFinder & ScopeDog display & controls

I'm still not sure the RPi HQ ccd will be good enough, and so my code recognises if an alternative ccd has been plugged into the RPi USB3 port and uses that instead.

A feature I added towards the end of the project was to automatically store the results of every plate-solve measurement, along with date, time, azimuth and altitude. I am expecting over time this will build up a picture of how and when the errors occur. Some will be down to poor initial two-star alignments and so be different for each session, and some will be due to errors in the mount geometry and telescope flexures and will be similar for all sessions.

After just a couple of nights of using the eFinder on my 18" scope the impact it has made is amazing and just what I had hoped for. The confidence in knowing my view in the telescope eyepiece is exactly where I wanted it to be within an arc minute, is just what I needed to look for the most challenging objects.

The eFinder integrated with ScopeDog on my telescope

I had quite quickly concluded that the Raspberry PI HQ ccd camera module, whilst an affordable and excellent camera, was not best suited to this role. The pixels were too small and exposures needed to be quite long. I bought instead an ASI120mm-S mono which proved to have excellent performance and was relatively easy to control from my Python code. This now gave me more flexibility in placing the Raspberry PI as previously the HQ ccd needed to be in the same box. I made a new box to house the Raspberry Pi and LCD display and button module, along with a 12V to 5V converter.

I try to maintain a blog on my website, and Bentley Ousley in Kansas came across it and contacted me, as he had wanted to make a similar electronic finder. He had got stuck on the code and asked for help. Bentley on the other hand had used his 3D printer to make a great 72mm f2.7 finder scope for the project. Collaboration was obviously needed! Bentley has a 20" New Moon Dobsonian driven by ServoCat and I would need to do some changes to my code. He had a spare lens and would 3D print me a housing, in return for me writing his code. A deal!

Although like mine his scope was using a Nexus DSC, the ServoCat uses both its ports (serial & USB) which I had been using to access scope position data. I contacted the ever-helpful Serge at AstroDevices and he suggested using the Nexus DSC wifi as an alternative route to gaining the outputs I needed. So now I had to learn how to open in my Python code a 'TCP/IP Client Socket' or whatever that was. The internet is a wonderful resource, and it wasn't long before I had this working. In fact, it worked so well that I changed my own eFinder to use wifi and hence eliminated a cable.

Using wifi to access the Nexus DSC also enabled me to send the 'move scope' commands to ServoCat. The Nexus DSC can use the classic LX200 communication protocol over wifi and this gave me everything I needed. Cleverly, the Nexus DSC can serve multiple clients over the wifi and so a tablet running SkySafari or similar in LX200 protocol will continue to control the telescope alongside my eFinder.

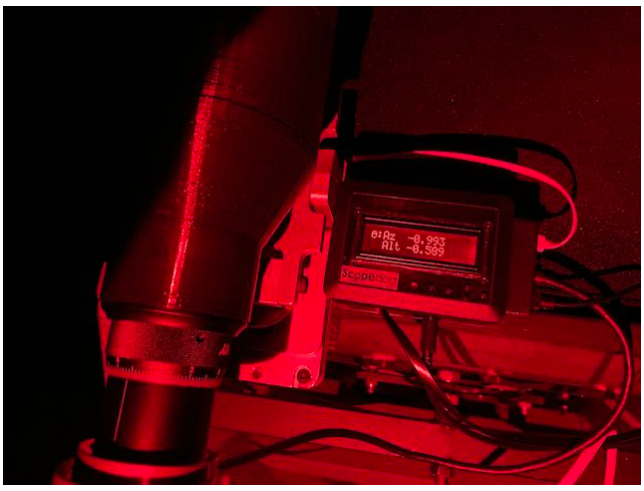
So, everything was set. Bentley assembled a Raspberry Pi and display module ready for the new code. With my help he configured the Raspberry Pi, its operating system and a few extra modules/drivers that were needed. He printed me a bespoke finder housing that would fit on my dovetail mount and along with the 72mm lens, despatched it to me across the 'pond'.

The housing arrived just in time for me to use it at an observing week I had planned at Kelling Heath in the UK. This was starting on the very first day we were allowed to stay 'out' after our latest Covid lockdown. I had 7 clear nights out of 7, almost unheard of in the UK! I had plenty of time to adjust settings and get the eFinder running smoothly. First step each session is to align the eFinder with the main scope. There is a handy App available for ASI ccd cameras that produces a rapidly updated image on an android tablet, complete with cross wire. The eFinder is mounted on an adjustable Geoptik dovetail bracket, and alignment to a reticule eyepiece in the main scope can be achieved within a fraction of an arc minute, and just takes a few seconds. For telescope elevation angles above about 15 degrees, I have found that this alignment stays true well within an arc minute.

The 72mm f2.7 lens turned out to be a game changer. With my ASI120MM-S ccd I only needed a 1 second exposure to capture more than enough stars for a reliable plate solve. Refining the input parameters to the plate-solve resulted in most solves taking only around 2-3 seconds. I suspect most of this time is used in accessing various index files on the SD card and my next task will be to pre-load the most commonly used ones into RAM.

The new 3D printed finder scope mounted on the Dobsonian.

At this stage the eFinder operates only partly automatically. This is so I can understand how well it is working and make changes. Current sequence is; do a 'goto' from either SkySafari or Nexus DSC, then on a short press of the eFinder button it reads the Nexus DSC to get this 'reference position', it takes an image, solves it, and displays the error in arc minutes between intended target and the centre of the field of view. Usually, this error is around 10-15 arc minutes, resulting from initial 2-star alignment tolerances, telescope mount inaccuracies and flexure.



A longer press of the button causes the scope to then be moved by the calculated amount, and a new image capture, solve and display of pointing error. Depending on the direction of the previous goto movement (hence backlash), this new pointing error is typically 1-3 arc mins. Another longer press of the button repeats the iteration, but now the backlash has been used up, the final indicated pointing error is usually less than 1 arc minute!

Typical eFinder display of final pointing result.

By about the 3rd night at Kelling Heath I was getting pretty much 100% solve success rates. Any failures were down to me nudging the scope when pressing the button, so it wasn't settled enough prior to image capture. The solution to this was to move the eFinder control box from the eyepiece, down to the bottom of the scope.

I was really happy with how it was performing, and it got me thinking! About 20 minutes later I had modified the code to add an 'autotrack' function. A really long press of the

eFinder button now started an ‘image – solve – move’ repeated loop. I was really pleased when first time I tried it, the scope ‘locked’ on to the sky position and followed within an arc minute or so. Each loop was taking about 5 seconds as I had inserted a pause period to let the scope settle before imaging. The corrections were very small as the scope is quite able to track quite well anyway. Tongue-in-cheek I purposely messed up a 2-star initial alignment big time by using the wrong stars and tried it again. This time the eFinder autotrack had to work harder and it was very satisfying to hear the corrections being applied every few seconds, keeping the sky position rock steady. My own Dobsonian has been built with ball bearings on both axes and so there is almost no stiction. The response of the scope to the motor drives is almost instantaneous and precise. Bentley is still struggling with bad weather and hasn’t yet been able to commission his system yet. I suspect on his more classic Teflon pad bearings the pointing accuracy may not be so good. We will see and perhaps another update article will be needed!

Painted eFinder in its new padded case.

In the meantime, once I got back home I rewarded my eFinder with a paint job and a foam lined custom case.

For now, while I am still developing the system, every image captured, error calculation, scope position and time, is stored on the Raspberry PI SD card. When reviewing the images I was amazed to find I could see M13 fairly well resolved. It just goes to show what fast optics can achieve in a 1 second exposure!

I am extremely happy with how the eFinder has turned out with performance way above my expectations. I suspect I may have the only ‘large’ Dobsonian able to accurately point to an arc minute or so.

I am also very pleased that I have been able to share the project. In fact, most of my projects end up getting shared. Usually, it is just giving help and advice in setting up similar systems, but in this case it was particularly satisfying to be able to bring complementary skills and resources together.

1 second eFinder exposure of M13.

Keith Venables FRAS