A Raspberry Pi Project

Hubble Pi

Telescope Imaging and Skyguiding System

Santiago Rodriguez

8.June 2020

Contents

1 Introduction and Motivation

First of all, hello and thank you for reading my protocol about this little project of mine! My name is Santiago Rodriguez and I'm the creator of this homebrewed telescope imaging and skyguiding system I decided to call Hubble Pi after the american astronomer Edwin Hubble, whose work in the field of extragalactic astronomy and observational cosmolgy provided some of the first evidence of the universes expansion, as well as the Hubble Space Telescope also named in his honour, which has produced some of the most breathtaking images of the cosmos during its service lifetime and is due to be succeeded by the James Webb Telescope due to be launched next year.

Since I'm currently a physics student with a very huge interest in astrophysics, I'd wanted to work on a way to capture images/recordings with my telescope as well as easily finding objects in the night sky since a long time. Having recently taken up an introduction to astrophysics as a winter semester lecture and after getting started this year with Linux as well as Python, I recently came up with the idea of making such a setup with the Raspberry Pi 4 and the newly released HQ 12Mp camera module an esteemed friend of mine told me about.

This was ideal, since the Raspberry Pi also has the future potential of being programmed for autoguiding as well as doing amateur stellar spectroscopy with the camera module and filters like the Star Analyser 100 due to being able to output the camera readings directly as Raw Bayer data captures or the more numerical YUV raw format for processing with spectrography software. In addition, the ability to fully and manually control the camera module using Python, with its high quality and high resolution Sony IMX477 sensor, meant these kind of applications would definitely be viable and highly adaptable for conventional astrophotography too thanks to the flexibility of the Picamera package as well as Python for scientific and numerical applications.

2 Getting Started - Parts, Software and Requirements

For this project, the following parts and software were used;

2.1 Parts

1. **Raspberry Pi 4 2GB**; any Raspberry Pi will do, really, as there is also no need for more than 2GB of RAM and the HQ camera module fits any of the models. The Raspberry Pi 4 is just the current model at the time of writing.

2. **HQ 12MP Camera Module**; the core of this build and a very impressive piece of hardware with a standard C- and CS-Mount for camera lenses. Only the sensor is needed for this project, as the telescope will essentially be acting as a supersized lens for it.

3. **3.5 inch 320x480 Touchscreen**; a small, low-power, resistive touchscreen for displaying the Kstars software and AstroCam GUI, comes with a pen and is powered solely by the Raspberry Pis GPIO pins.

4. **1.25 inch to C-Mount Telescope Adapter**; as the standard size for telescope eyepieces is about 1.25 inches, this adapter will fit the HQ camera module with its C-Mount onto the telescope. There are also other adapters for less standardized telescope ocular sizes, so searching for a different one is also possible if the telescope requires it.

5. **Telescope Phone Mount**; there are many mounts in the wild for attaching the phone besides the eyepiece in order to focus its camera on the output and take picture. Such a mount can be used here for keeping the Raspberry Pi board besides the ocular, as long as it also doesn't cover the eyepiece.

6. **32GB SD Card and 8GB Mini USB Stick**; the 32Gb SD card is for installing the Raspberry Pi OS as well as the required software for this project, the 8GB USB stick for later transferring the image data to a computer. Larger storage units can also be used, but are not necessary for the immediate uses in this project.

2.2 Software

1. **Raspberry Pi OS (Buster 32-bit Desktop Only)**; at the time of writing, the latest stable release of Raspberry Pi OS and both lightweight as well as flexible enough te serve as the backbone of this project. The recently released 64bit beta didn't work with the Raspberry Pi camera module for some reason, but should also work in future, stable releases as well.

2. **SSH-XRDP Remote Desktop and Autohotspot**; for accessing the Hubble Pi from a laptop while on the field or at home, SSH and XRDP allow for a remote desktop session to be established while the devices are on the same network, thus removing the need for a dedicated keyboard+mouse when more control is needed. Coupled together with the Autohotspot setup script by RaspberryConnect, which turns the Hubble Pi into a WiFi hotspot when no available connections are detected, one can access the Hubble Pi from a laptop, tablet or phone wirelessly anywhere without an access point.

3. **KStars**; the KDE software for simulating the night sky and finding celestial objects in a live representation of it. It comes with the Ekos astrophotography suite and potential for further integration with astronomy tools through INDI.

4. **AstroCam**; a Python 3 script with a Tkinter GUI I developed for controlling the camera sensor and quickly adjusting astrophotography relevant settings on the touchscreen. It's based on the code of Erik here for streaming the camera output as a preview and will be further expanded on this projects Github repository as I test it out in the field.

5. **Camera Preview**; a quick desktop script for executing the raspistill preview terminal command. Can act as a viewfinder when setting up the scope and is a little bit more stable than the stream-based AstroCam preview.

6. **Drivers and Packages**; depending on the touchscreen model, drivers might be necessary in order to enable the touchscreen interface. The Python 3 Tkinter and Picamera packages are also required in order for the AstroCam script to run.

2.3 Requirements

1. ***Telescope***; as a matter of fact, any telescope which accepts 1.25 inch eyepieces should do. The picture quality however will largely depend on the quality of the optics, the light gathering power and thus the aperture D, as well as if the telescopes mount can compensate for the Earths rotation so as to allow for longer exposure times t_e of the camera sensor without image breaking trailing showing up.

2. ***Power Delivery Method***; a method for powering the Hubble Pi while setting it up at home or while using it on the field will also be necessary. Since the Raspberry Pi 4 board powering the Hubble Pi accepts a USB Type C input for power delivery, any modern phone charger should do while at home, whereas on the field there is the option of using either a phone powerbank or a laptops USB port, as long as they are capable of delivering the required power draw by the Raspberry Pi 4 board while powering the screen and the camera module.

3. ***Setup Interface***; for the first setup, the Hubble Pi will need an external interface like a mouse, keyboard and monitor or another computer to SSH into the Raspberry Pi OS terminal and set up a remote desktop access with XRDP. After the remote desktop and Autohotspot is set up, the Hubble Pi can be accessed and further configured from any device capable of a WiFi connection, like a laptop, tablet or phone.

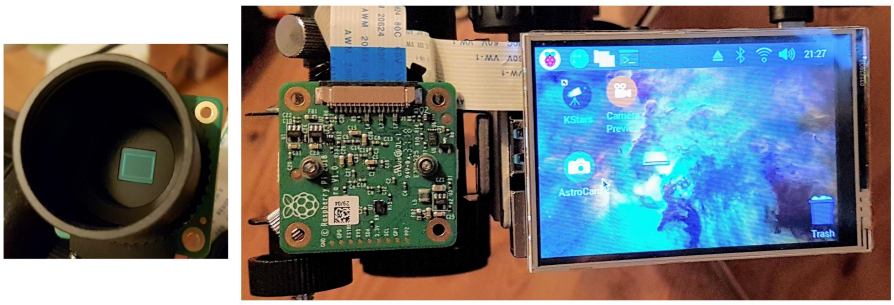

Figure 1: Camera Sensor with C-Mount Telescope Adapter and Hubble Pi Board

3 First Setup and Initial Testing

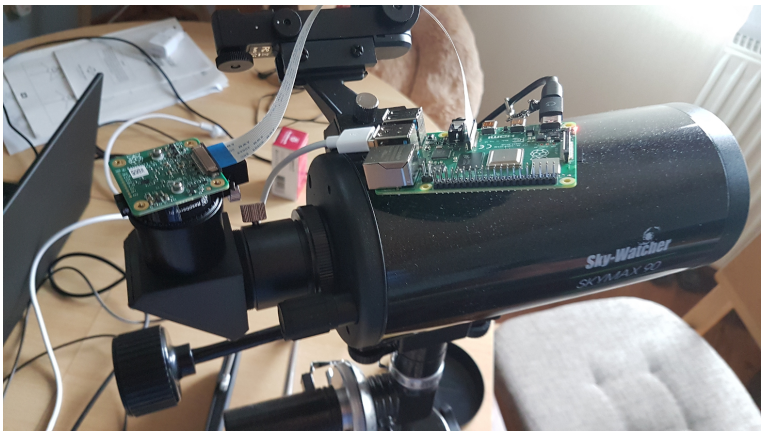

Figure 2: Initial Setup for Testing

Once all the required parts are assembled, the Raspberry Pi OS image can be flashed to the SD card on a computer with the Raspberry Pi Imager. By putting an empty ssh file into the boot directory along with a wpa_supplicant.conf file for the network (for more information click here), one can then access the Hubble Pi directly from a computer and install XRDP for controlling the board through a remote connection or with a monitor and keyboard-mouse combination. After connecting and enabling the camera module, an initial mounting onto the telescope with the 1.25 inch to C-Mount adapter allows for some first test shots and focusing adjustments.

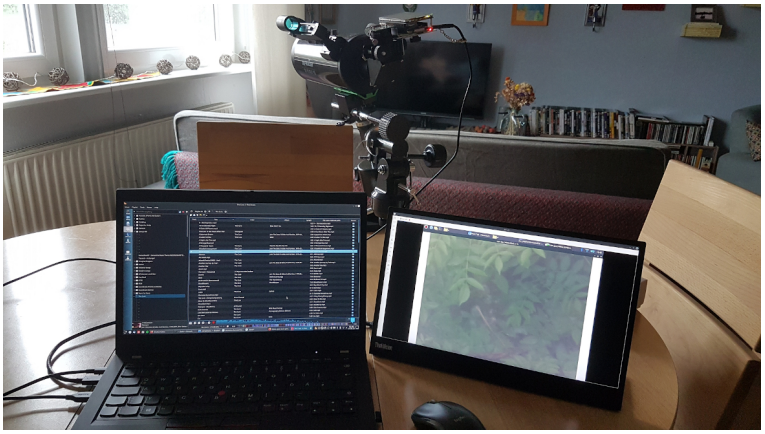

Figure 3: Image Testing with Raspistill and Remote Desktop

For initial testing, the **raspistill** and **raspivid** command line tools can be used. By typing the code below into a terminal

raspistill -o Desktop/image.jpg

a quick 5 seconds preview of the camera (not visible in a remote desktop session) is displayed before saving a picture to the desktop, while on the other hand with

raspivid -o video.h264 -t 10

a small 10 seconds video is shot by the sensor. In order to adjust the image focus, a permanent preview can also be called through the command

raspistill -t 0

and viewed through the 3.5 inch screen once installed and set up with its drivers. By pointing the telescope towards a far away object like a distant tree and adjusting the focusing knob, a sharp image should come into focus on the preview of the camera sensor.

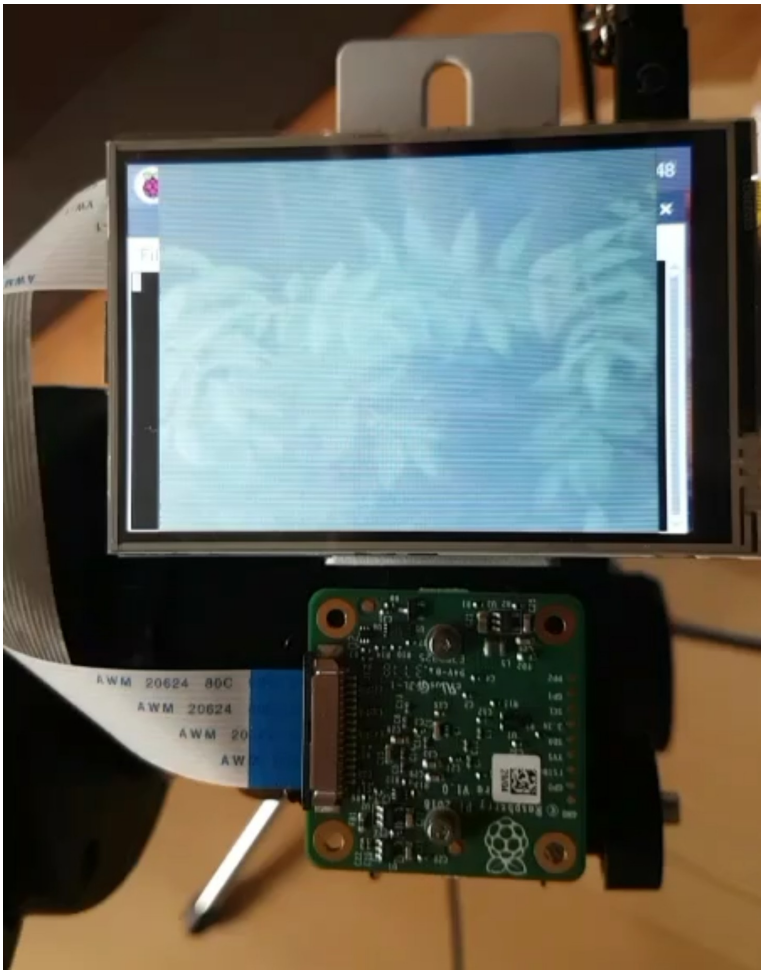

Figure 4: Camera Preview on the 3.5 inch screen
(foggy due to shooting through a thick window on a rainy day)

Note: The new HQ camera module is capable of shooting at a max resolution of 4056 x 3040 pixels and can thus require more VRAM to draw on for longer camera or video shooting commands. In case of performance issues due to running out of video memory, the amount of memory available to the Raspberry Pi 4s GPU can be increased from the standard 128mb to 256mb in the Raspberry Pi Configuration Tool under Memory split and in this use case should not negatively impact any other functionality except for models with less available memory.

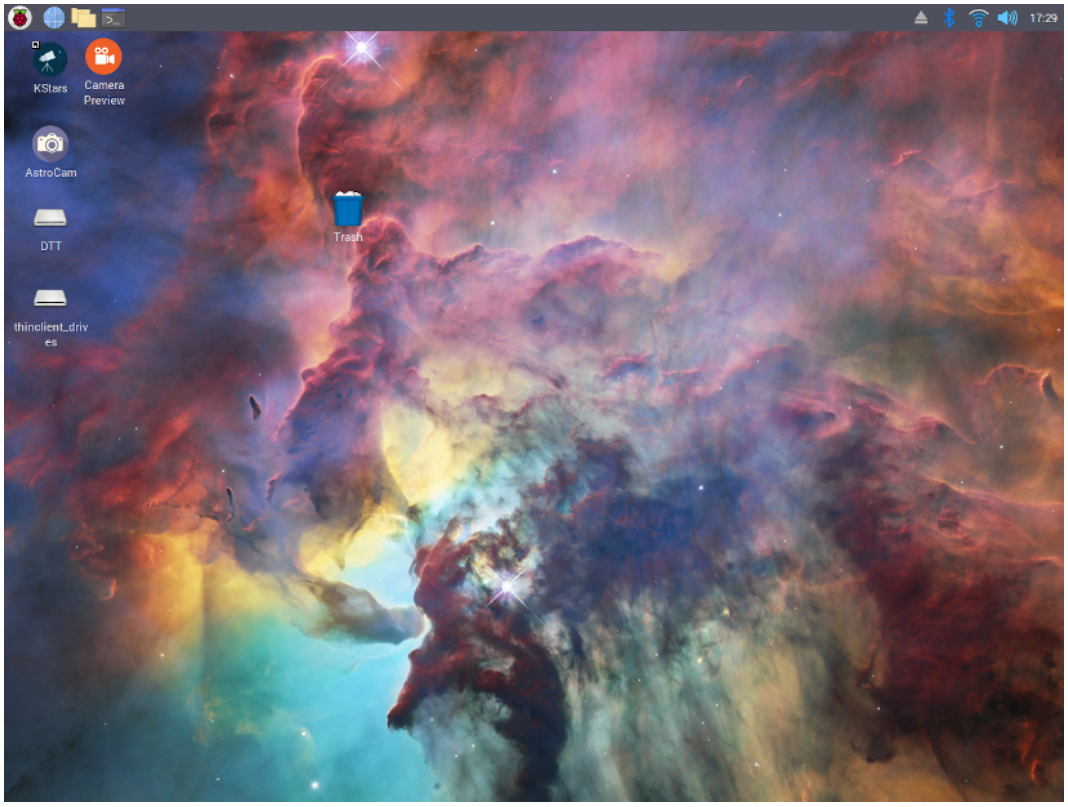

Figure 5: Hubble Pi Desktop

Once the initial testing is done and the integrity of all components has been verified, the desktop can be set up with a personalised background like the one shown above as well as desktop shortcuts for the different programs which will be installed on the Hubble Pi. The KStars shortcut can be obtained from the program list after installing KStars through the commands

sudo apt-get update

sudo apt-get install kstars

and by then changing the icon to the one provided for it in the Github repository. The other two shortcuts can be set up by pasting the following two text files from the Github respository on the desktop and subsequently making them executable through the command line with

sudo chmod +x /PathToShortcut

After that, the only remaining thing left to do is to create the folders

/home/pi/AstroCameraApp/PythonScripts/

/home/pi/Pictures/Astrophotography/H264

/home/pi/Pictures/Astrophotography/JPG

/home/pi/Pictures/Astrophotography/RAW

then put the AstroCam.py in the PythonScripts folder and make it executable with the chmod command. If needed, one can install the Python 3 packages Tkinter and Picamera should they be missing, as well as the Autohotspot according to the installation guide provided on Raspberry Connect. Lastly, it is also possible to enable one click file execution instead of double click by going to the file manager settings, so as to further optimize the Hubble Pi desktop for a touchscreen interface.

4 AstroCam and the PiCamera Python Module

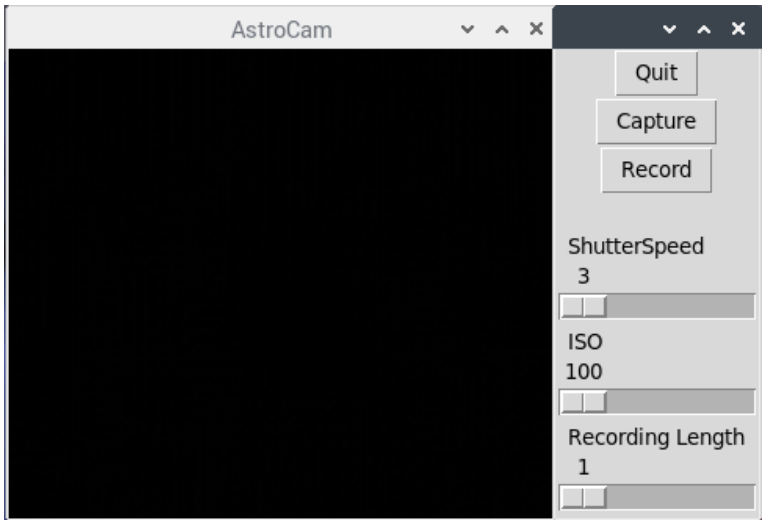

Figure 6: AstroCam Touchscreen GUI

Note: Under heavy development

The AstroCam program provided in this Github repository was adapted from the code of Erik here and repurposed for use in astrophotography sessions with the HQ camera module and a touchscreen. For this, it shows both a preview of the pictures the camera would be taking as a live stream, as well as the three main settings seen above for controlling the sensors shutter speed and thus exposure time, the light sensitivity by means of the ISO and lastly the recording time of each recording session. The interface is configured for a 320x480 touchscreen; if controlled through remote desktop, there's also an advanced settings window right below which allows for modification of further settings like brightness and contrast. Other GUI arrangements for different screen resolutions and sizes can be set up by modifying the corresponding Tkinter values in the Python script.

The Capture button is programmed to take a picture according to the set parameters and output it as a RAW image in the standard RGB format as well as optionally a JPG (for this, just uncomment the corresponding lines in the Python script) to the previously set folders. However, if the recording lenght is above 1, then the Capture button will instead make a capture sequence of the specified recording lenght. The Record button takes a video recording at a FHD resolution during the chosen recording lenght in seconds.

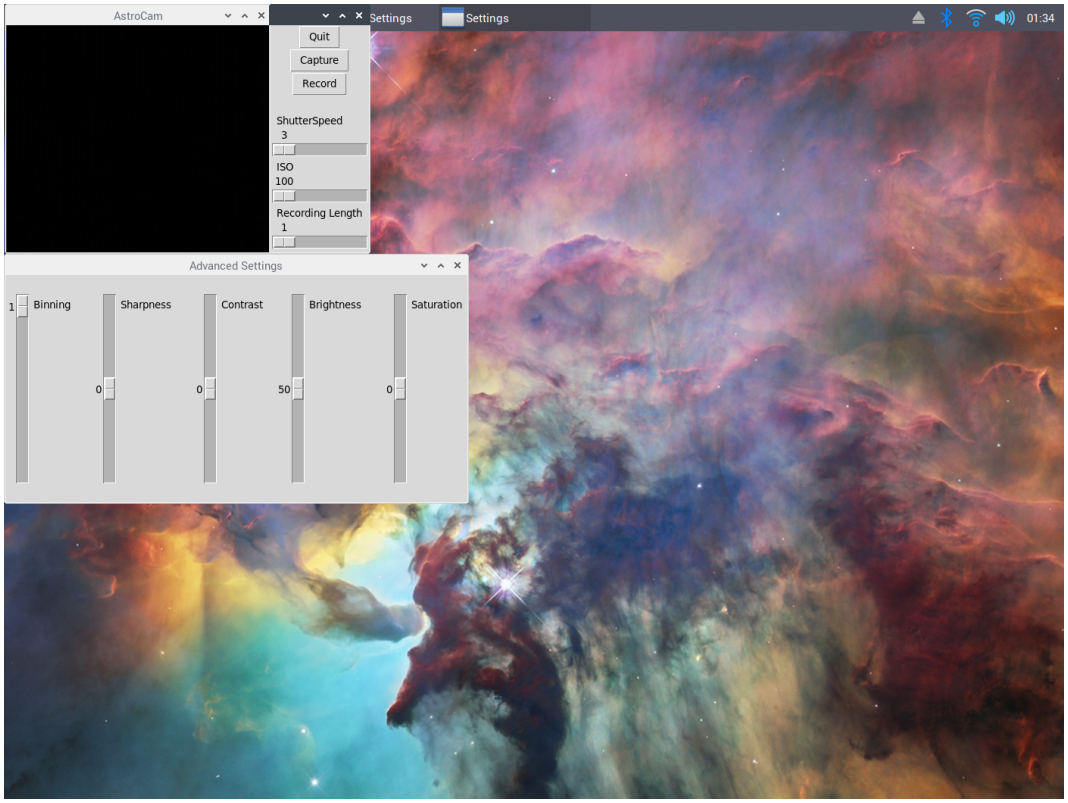

Figure 7: AstroCam Remote Desktop GUI

4.1 AstroCam Functions and Settings Overview

For a more comprehensive overview of all the functions and settings in the AstroCam software, a list with the PiCamera Python package attributes being set by each along with a short explanation will be provided below;

4.1.1 Function Buttons

1. ***Capture***; executes the **picamera.capture()** command and saves a timestamped RAW image in the RGB file format (along with a JPG if uncommented in the Python script) to the directories */home/pi/Pictures/Astrophotography/RAW* and */home/pi/Pictures/Astrophotography/JPG*. For recording lengths above 1 it does the same, only with a **picamera.capture_sequence()** command of the selected length and a full window preview during the capture session (only on the touchscreen).

2. ***Record***; starts a video recording with the **picamera.start_video()** command and keeps recording for the specified recording length. The Shutter Speed setting is ignored, resolution is fixed at 1080p/30fps and exposure mode is set to auto as well as the white balance. It also shows a direct preview of the video being recorded (also only on the touchscreen). Once finished, it saves the video in a H264 format to the */home/pi/Pictures/Astrophotography/H264* directory.

3. ***Quit***; self-explanatory. Quits the programm and shuts down all its subprocesses.

4.1.2 Touchscreen Settings

1. ***Shutter Speed***; sets the **picamera.shutter_speed()** attribute in centiseconds to a fixed value between 1 (which would be 1/100 of a second) and 200 (2 seconds, the maximum for the HQ camera module). It's set to 1/33 of a second by default, so as to allow a framerate of around 33fps to settle, and only affects the Capture function, as the Record function fixes it automatically according to the exposure speed. Used to adjust the exposure time of each camera capture by the sensor.

2. ***ISO***; adjusts the **picamera.iso** setting between the allowed values for the HQ camera module of 100 and 800. Higher values yield higher light sensitivity, but also increase noise. Used to change the sensors light sensitivity and increase information capture for dimm objects in the night sky.

3. ***Recording Length***; sets the recording length in total captured frames (Capture function) or recording time in seconds (Record function). Set to 1 in order to execute regular single **picamera.capture()** commands with the Capture function and set to higher values for the **picamera.capture_sequence()** command to be carried out with the specified amount of captures.

4.1.3 Advanced Settings

1. ***Binning***; divides the value of the **picamera.resolution** attribute for the Capture function by the specified denominator in a procedure known as pixel binning, which sees the data from various, adjacent pixels combined into one by reducing the resolution so as to increase light sensitivity. A /1 denominator will keep the Capture resolution at the IMX477 maximum value of $(4056, 3040)$, whereas a /2 denominator will halve this resolution down to $(2028, 1520)$ and combine the information of two adjacent pixels. Up to /4 denominators are supported with a resulting 4 adjacent pixel binning. Like the Shutter Speed setting, this does not as of yet support the Record function, which uses a fixed $(1920, 1080)$ resolution.
   Note; All this setting does is reduce the camera resolution, which in theory should lead to the sensor firmware automatically recognizing the possiblity of pixel binning and thus start automatically doing so. Requires further testing and more specific PiCamera documentation for the HQ camera module to be released, as sensor modes and pixel binning were only covered there at the time of writing for the V1 and V2 camera modules (PiCamera release 1.13, 2018)

2. ***Sharpness***; sets the **picamera.sharpness** attribute to values between 0 and 100, with 0 being the cameras default. Affects all functions.

3. ***Contrast***; sets the **picamera.contrast** attribute to values between 0 and 100, with 0 being the cameras default. Affects all functions.

4. ***Brightness***; sets the **picamera.brightness** attribute to values between 0 and 100, with 50 being the cameras default. Affects all functions.

5. ***Saturation***; sets the **picamera.saturation** attribute to values between 0 and 100, with 0 being the cameras default. Affects all functions.

4.2 The PiCamera Python Package and HQ Camera Sensor

The PiCamera Python Package package provided by the Raspberry Pi Foundation provides the backbone and interface for the AstroCam software to communicate with the HQ camera sensor distributed by the same foundation. It is important to understand here, however, that the PiCamera package -while very flexible in its design and camera control settings for all kind of applications- is still limited at some level by the camera firmware and hardware design inherent to the IMX477 CMOS sensor's nature as a camcorder chip. The purpose of this section is to give a short comment on these limitations and discuss their effects on the AstroCam Python code structure as well as the HQ camera module's potential for astrophotography.

As explained under the Camera Hardware Section 6 of the PiCamera documentation, thr Pi's HQ camera module works basically like a phone camera sensor. This means that unlike the typical DSLR sensors employed by most astrophotographers, it does not have a physical shutter for controlling things such as exposure time by preventing light from falling onto the chipset and instead uses a digital "rolling shutter" which reads the constantly information-streaming pixel rows in the sensor and resets their values after each shutter cycle. This translates into two crucial design decisions for the camera sensor;

Firstly, it is constantly streaming images even while idle down to the Raspberry Pi 4 board for processing and using that data to set the white balance gains as well as the exposure modes automatic digital and analog gain control for the camera. This is important for turning the automatic white balance and exposure mode to 'off' in order to capture consistent image sequences; as *these last two values [analog and digital gains] are not directly settable, and default to low values when the camera is first initialized. Therefore it is important to let them settle on higher values before disabling automatic gain control otherwise all frames captured will appear black.[1].* Thus they can't be manually set in the AstroCam software and must be left to automatically settle for a reasonable amount of time before fixing them (this is done automatically before each capture session initialized by the Capture function).

Secondly, the digital "rolling shutter" essentially limits the maximum exposure time - a precious feature for all those deep-sky and dimm sky objects photographies- to the slowest speed at which the sensor can be made to read and reset the values of each pixel row after initializing the sensor (or in other words, the minimum framerate at which the sensor can be made to output images). Based on page 110 of the official Camera Guide, this minimum framerate is achievable at the full sensor resolution and lies at around $0.005 frames/second$, resulting in a maximum exposure time of around $\frac{1 frame}{0.005 frames/second} = 200 seconds$ and thus leading to the maximum shutter speed value of 2 minutes in the AstroCam software. This value is far below the exposure time proper astrophotography DSLR and CCD cameras are capable of, owing to the IMX477 sensors nature as a camcorder chip, but can still be usable for photographing galaxies and other deep-sky objects bright enough which don't require exposure times longer than that, specially when combined with image stacking and proper post-processing of the RAW files.

4.3 Notes and Future Development

After only one week of development since I first started the project, my AstroCam Python app is still far from finished and has only now barely reached a state of useability I would consider sufficient to upload it to a GitHub repository. Future improvements of the GUI such as managing the settings across different tabs in the same window or allowing for direct numeric input of the settings through a virtual numpad are already in the works, and some tinkering with as well as further testing of the camera module on the field should allow for some deeper insights into the pratical limitations caused by the design decisions discussed above, which could be potentially accounted or even made up for with extra lines within the Python code.

Long term features which are already under consideration include;

1. ***AstroSpectra***; an extension for doing amateur spectroscopy by capturing and storing data taken with filters like the Star Analyser 100 in a more numerical format like YUV or Numpy arrays, maybe even directly in the Raw Bayer data recorded by the sensor, as well as processing it into a format which other spectrography programs can work with.

2. ***AstroGuider***; an extension for autoguiding a motorized scope by using the preview image data as reference for the Hubble Pi to control the GoTo mount and keep a given sky object in focus. Could potentially be integrated with a GPS module and other astronomy software with built-in sky catalogues such as Kstars to also automatically point the telescope at any given object in the night sky and subsequently keep track of it.

For improvements to the Python code as well as implementations of potential new features, the GitHub repository containing the AstroCam script will also be open to pull requests.

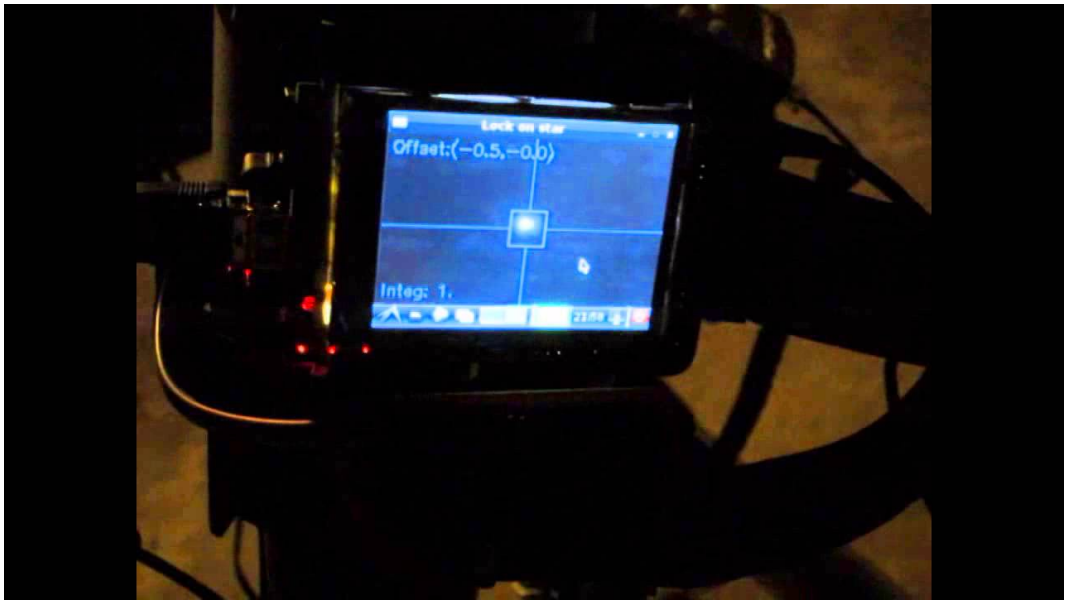

Figure 8: Python Based Raspberry Pi AutoGuider by Samson Yang

5 KStars and INDI Integration

Figure 9: KStars on the Hubble Pi Touchscreen

KStars is a free, open source astronomy software by KDE used in the Hubble Pi for its skyguiding tasks. It shows a live, graphical simulation of the night sky at the current, input location and is thus very usefull for pinpointing the location of celestial objects in the night sky, as well as predicting their future locations for plannification purposes. Although the small touchscreen doesn't allow for easy input of more complex tasks like FoV displaying or altitude-time plots, which KStars is also capable of performing, it's still perfectly sufficient for use in the field to quickly find the current position of an object in the skyand observe it. Combined with its catalogue of *up to 100 million stars (with additional addons), 13,000 deep sky objects, constellations from different cultures, all 8 planets, the Sun and Moon, and thousands of comets, asteroids, satellites, and supernovae*, KStars turns the Hubble Pi into a standalone planetarium attached to the telescope and capable of imaging the observed sky object.

Unfortunately, not all its features work out of the box; Ekos, a completely integrated astrophotography solution that comes built into KStars and which can control INDI-compatible devices including numerous telescopes, CCDs, DSLRs, focusers, filters and so on, doesn't seem to be compatible with the HQ camera module from my testing, hence my necessity to programm the AstroCam app to control it. This has probably to do with the camera module connecting to the Pi through a ribbon cable and being controlled by either the Raspistill command lines or the PiCamera Python package commands, neither of which are very streamlined camera control interfaces. I don't have much experience yet with Ekos though, as this is my first astrophotography setup, and maybe there's a way to interface the camera module with INDI through some kind of Python script. A guide I found on INDI integration here details how to set up an INDI server on the Raspberry Pi for otherwise full telescope and DSLR/CCD remote control if desired.

6 Sample Images

For illustration purposes a short selection of mostly unprocessed images taken with the HQ camera sensor will be shown below. They were taken mainly for testing purposes during relatively foggy summer nights in Germany from a mildly light polluted town with a 90/1250 Maksutov-Cassegrain telescope, so proper postprocessed images taken under better conditions will probably deliver a lot of further details.

Figure 10: Mare Crisium

Figure 11: Oceanus Procellarum and Crater Aristarchus/Vallis Schröteri

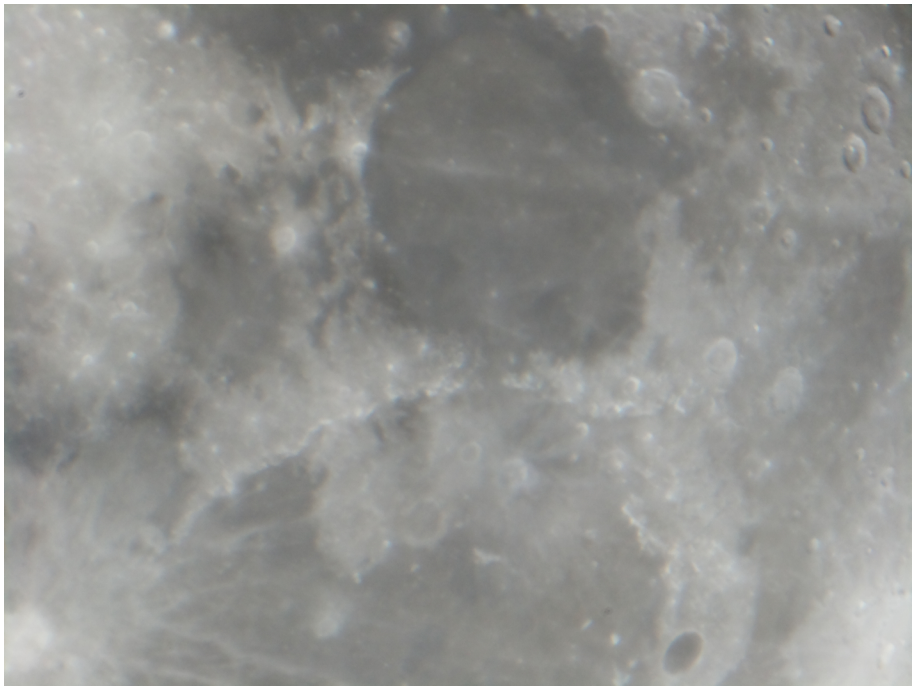

Figure 12: Mare Serenitatis anpd Mare Imbrium

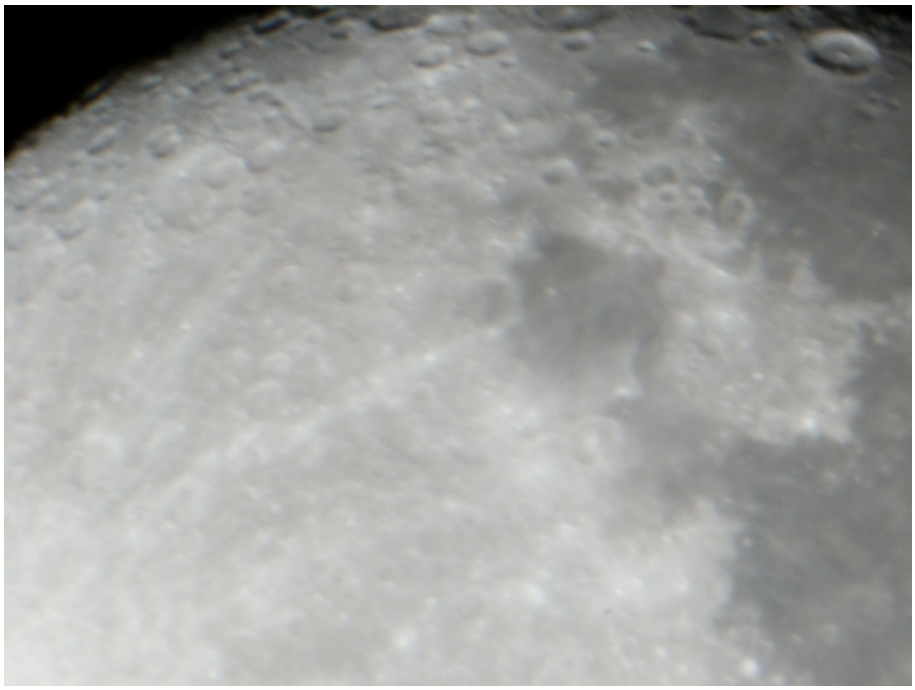

Figure 13: Mare Nectaris and Mare Fecunditatis

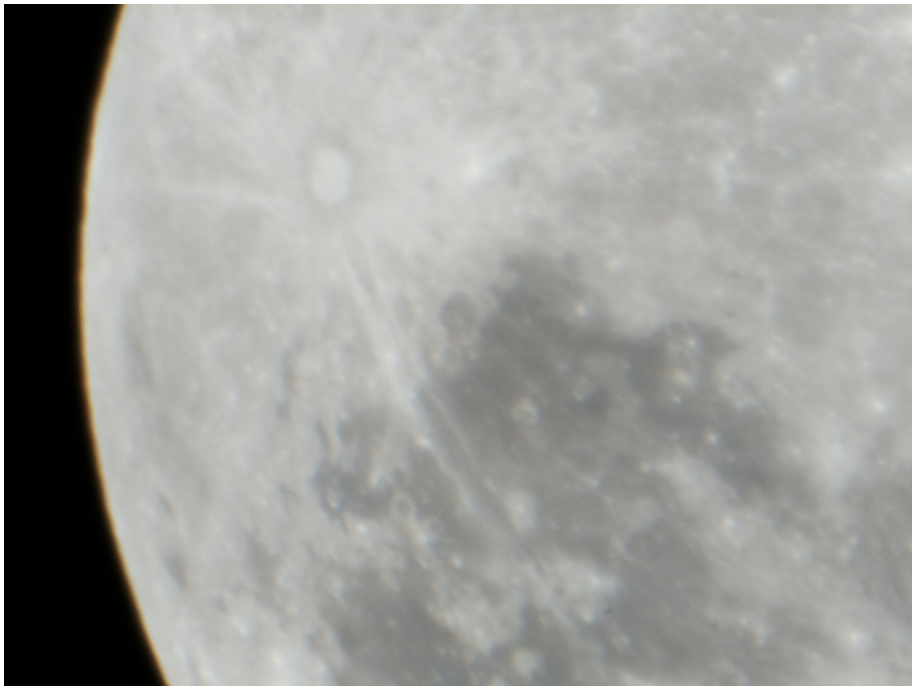

Figure 14: Crater Tycho and Mare Nubium

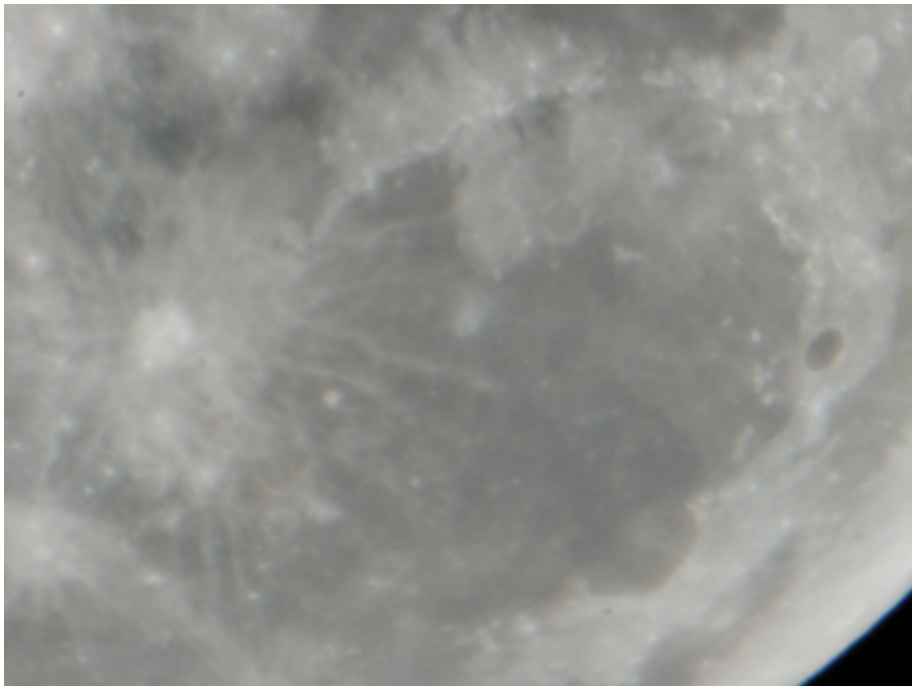

Figure 15: Mare Imbrium and Crater Copernicus

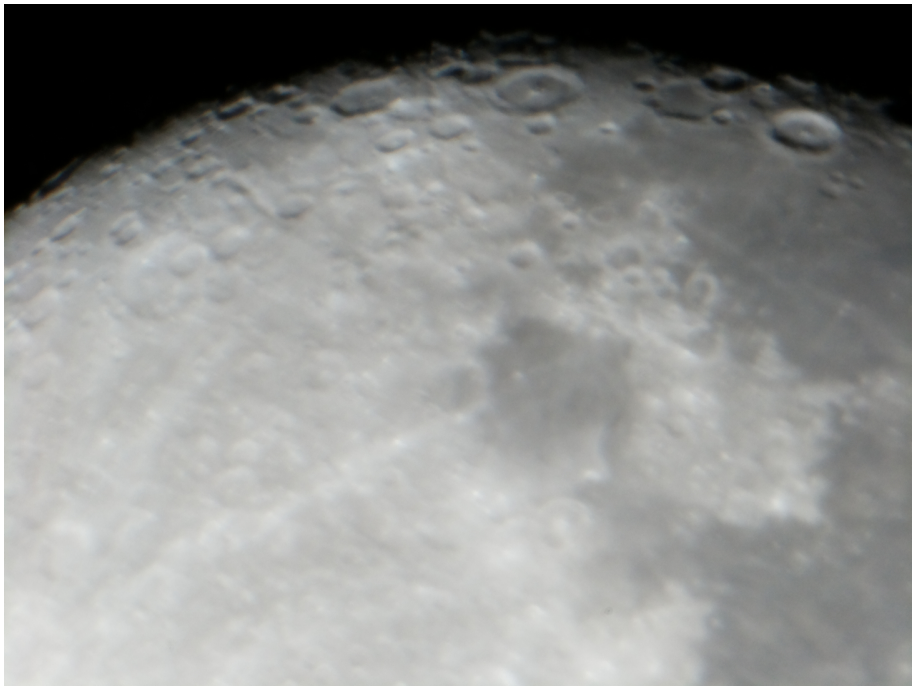

Figure 16: Mare Nectaris and Mare Fecunditatis

Figure 17: Oceanus Procellarum on the Touchscreen Camera Preview

Figure 18: Jupiter (Cropped Image)

Figure 19: Jupiter Full Size Image

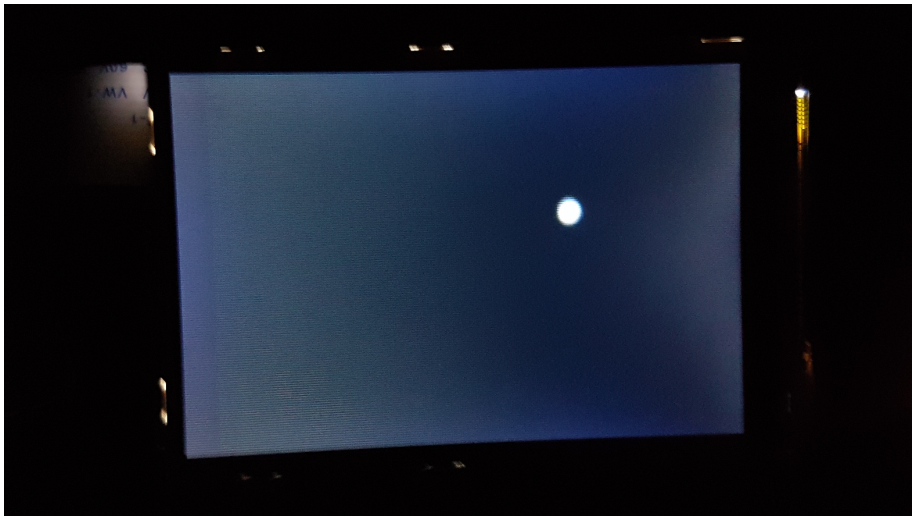

Figure 20: Jupiter on the Touchscreen Camera Preview

Figure 21: Saturn (Cropped Image)

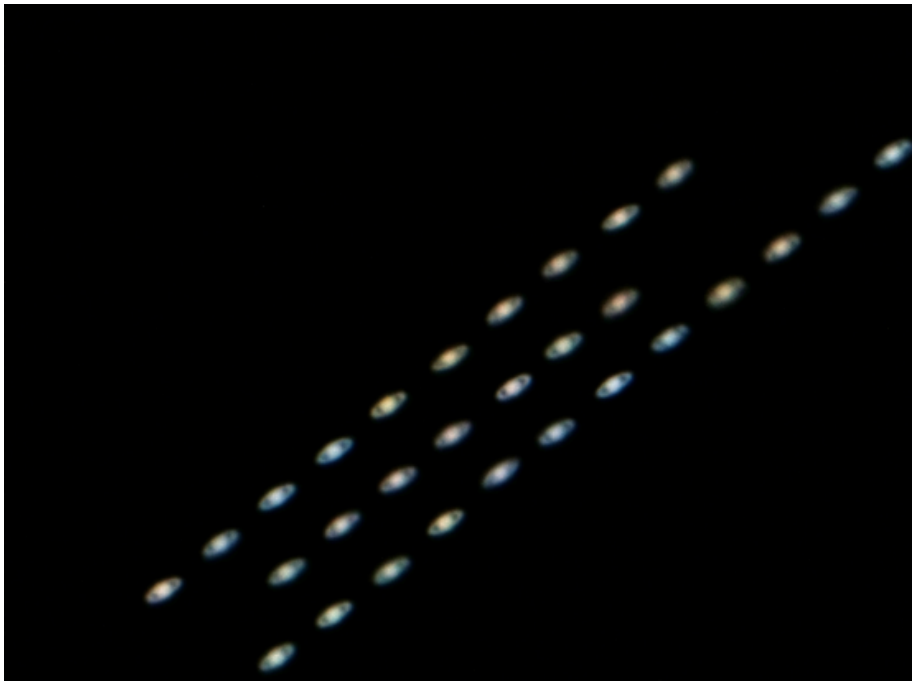

Figure 22: Saturn Stacked Images

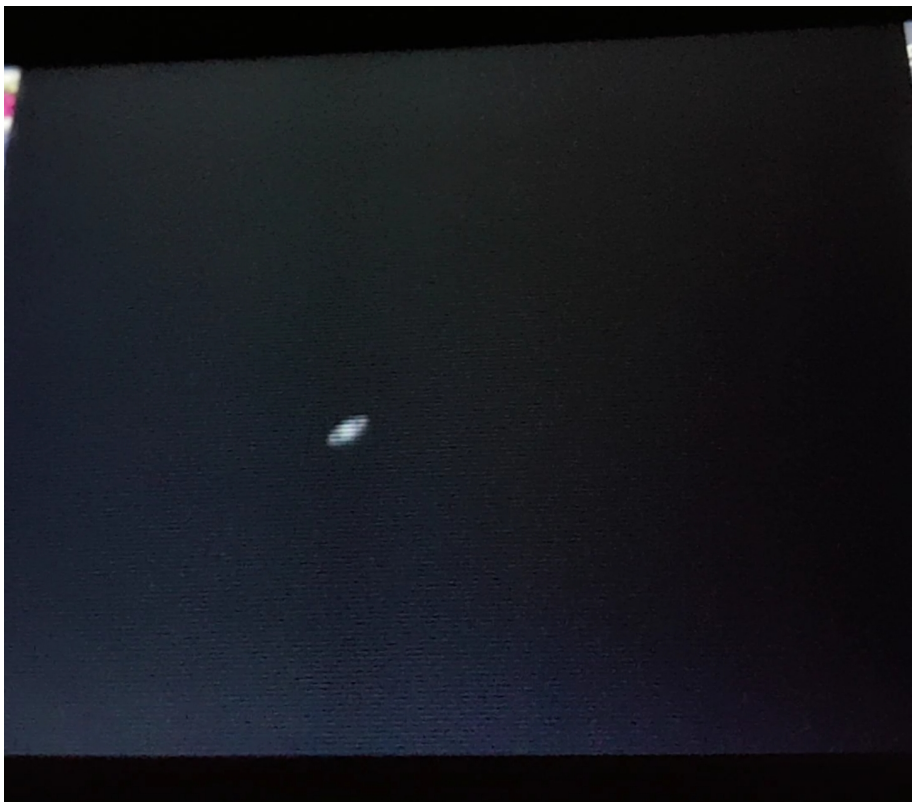

Figure 23: Saturn on the Touchscreen Camera Preview

7 Afterword and Credits

Although still a work in progress and far from finished, I want to give credits to all the open source work done by others before me which made this project possible in the first place. This includes the Raspberry Pi Foundation's Raspberry Pi 4, Raspberry Pi OS and HQ camera module as well as PiCamera Python package, the Python programming language and Tkinter package behind my AstroCam GUI, Eriks code upon which I based it as well as RaspberryConnect's AutoHotspot script for remote desktop access to the Hubble Pi while on the field, and last but not least of course the KStars and INDI software made available by KDE and the incredible astronomical community respectively. To all of you, none of this would have been possible without your efforts and for that you have my utmost gratitude.

"If I have seen further than others, it is by standing upon the shoulders of giants." - Isaac Newton