Professorship of Lunar and Planetary
Exploration Technologies
Prof. Dr.-Ing. Philipp Reiß

Term Paper

Hard- and Software Development of a Moon Guiding System for Telescopes

LPE SA-2023/07

Author:
Tobias Kurz

Supervisor: Prof. Dr. Detlef Koschny
Professorship of Lunar and Planetary
Exploration Technologies
Technical University of Munich

Professorship of Lunar and Planetary
Exploration Technologies
Prof. Dr.-Ing. Philipp Reiss

LPE-Nummer:	SA-2023/07
Titel der Arbeit:	Hard- and Software Development of a Moon Guiding System for Telescopes
Autor:	Tobias Kurz
Matrikelnummer:	03770864

Erklärung

Mir als vertraulich genannte Informationen, Unterlagen und Erkenntnisse werde ich nach meiner Tätigkeit am Lehrstuhl nicht an Dritte weitergeben.

Ich erkläre mich außerdem damit einverstanden, dass meine Bachelor-, Semester-, Master-, oder Diplomarbeit von der Professur auf Anfrage fachlich interessierten Personen, auch über eine Bibliothek, zugänglich gemacht wird, und dass darin enthaltene Ergebnisse sowie dabei entstandene Entwicklungen und Programme von der Professur für Lunare und Planetare Explorationstechnologien uneingeschränkt genutzt werden dürfen. (Rechte an evtl. entstehenden Programmen und Erfindungen müssen im Vorfeld geklärt werden.)

Ich erkläre außerdem, dass ich diese Arbeit ohne fremde Hilfe angefertigt und nur die in dem Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Ottobrunn, den 31.01.2024

Unterschrift

Zusammenfassung

Diese Semesterarbeit befasst sich mit der Hard- und Softwareentwicklung eines Systems zum Nachführen des Mondes mithilfe eines motorisierten Teleskopstativs. Ziel ist es, die präzise Nachführung des Mondes, mithilfe kommerziell erhältlicher Komponenten, für mehr Menschen möglich zu machen. Basierend auf der Auswertung von Bildern einer Kamera mithilfe eines Raspberry Pi wird der Mond im Bild erkannt. Bei Abweichungen der Position wird ein Steuersignal mithilfe der ST-4 Schnittstelle and die Montierung übermittelt.

Eine umfassende Bauanleitung der Hardware, sowie eine Anleitung zur Nutzung der Software sind zur Verfügung gestellt. Umfangreiche Tests zu unterschiedlichen Mondphasen wurde durchgeführt und die Zuverlässigkeit des Systems nachgewiesen. Verschiedene Profile wurden anhand einer Parameterstudie erstellt und getestet. Das System ist flexibel und modular gestaltet und ermöglicht somit einen hohen grad an Anpassbarkeit sowohl in der Software als auch in der Hardware.

Abstract

This term paper focuses on the hardware and software development of a system for guiding a motorized telescope mount to follow the Moon. The objective is to enable precise lunar guiding with commercially available components, making it accessible to more people. The system identifies the Moon in images captured by a camera by utilizing a Raspberry Pi. When deviations in the Moon's position are detected, a control signal is transmitted to the mount via the ST-4 interface.

Comprehensive construction instructions for the hardware along with a guide for software usage are provided. Extensive tests were conducted during various lunar phases to demonstrate the system's reliability. Different profiles were created and evaluated through a parameter study. The system is designed to be flexible and modular, offering a high degree of adaptability in both software and hardware aspects.

Contents

List of Figures

List of Tables

Symbols and Formulas

Symbol	Unit	Description
α_d	rad	viewing angle
α_H	rad	horizontal viewing angle
α_V	rad	vertical viewing angle
β	rad	angle of sensor diagonal
d	mm	sensor diameter
f	mm	focal length
l_H	pixels	horizontal pixel count of sensor
l_V	pixels	vertical pixel count of sensor
ϕ	rad	misalignment angle

COTS	Commercial Off-The-Shelf
DEC	Declination
FPS	Frames per Second
GPIO	General Purpose Input/Output
HDMI	High Definition Multimedia Interface
LPE	Professorship of Lunar and Planetary Exploration Technologies
RA	Right Ascension
RJ12	Registered Jack (standardized 6-pin connector)
ST-4	Star Tracker version 4

1 Introduction

The exploration of celestial bodies, particularly the Moon, has been a subject of significant interest in the field of astronomy and space science. This thesis presents the development of a Raspberry Pi based system designed to guide amateur-sized telescopes for the observation of the unilluminated side of the Moon. The primary objective of this research is to facilitate the detection of impact flashes caused by small asteroids, a phenomenon observable from Earth.

While existing commercial systems offer capabilities in tracking the stars, they lack of functionality for guiding telescopes to follow the Moon's trajectory. This gap necessitates the development of a specialized system capable of providing precise and consistent guiding of the Moon. The system developed for this thesis aims to address this need by integrating a Raspberry Pi with a camera to autonomously guide a telescope mount, thereby keeping the unilluminated part of the Moon within the field of view for extended periods. The methodology involves a comprehensive approach encompassing hardware design, software development, and empirical testing. The hardware component entails designing and integrating a system, which can be attached to a standard telescope mount. On the software front, a Python-based tool is for detecting the Moon and issuing corrective commands to the mount is developed.

The system provides a cost-effective and efficient solution and therefore making the research on asteroid impacts on the Moon more accessible.The implications of this study extend beyond observational astronomy, potentially aiding in the understanding of near-Earth space and its asteroid environment.

2 Hardware

Motorized telescope mounts are used by both amateur and professional astro-photographers to capture sharp images of the night sky. This is made possible due to the mount rotating in a direction opposite to that of Earth's rotation, thereby compensating for it. Without this counter-rotation, stars would appear as streaks in images with exposure times of just a few seconds. The research team for Lunar and Planetary Exploration Technologies (LPE) intends to use such a telescope mount to guide a camera aimed at the Moon. As the Moon orbits in the same direction as Earth's rotation, it appears to move slightly slower across the night sky compared to the background stars. Therefore, some commercial mounts, lacking a dedicated Moon mode, would lose track of the Moon after some time. Additionally, minor misalignments of the mount from the Earth's rotational axis, as well as the fact that the orbit of the Moon is slightly tilted, can lead to significant long-term guiding errors. This is where the Moon guider system comes into play. Using the standard ST-4 interface, which most commercial mounts are equipped with, correction signals will be transmitted to maintain the Moon in the frame during tracking periods extending over several hours. This leads to a set of specific requirements that need to be defined.

2.1 Requirements

Affordability
The System should be comprised of commercial off-the-shelf (COTS) components. To enable data collection through crowd sourcing, access must be made easy. Ideally, components that are already available within the astro-community should be utilized.

Usability
To maximize the system's user-friendliness, achieving a compact design is imperative. The guiding hardware should be designed for seamless mounting onto a standard telescope mount, requiring minimal modification to the mount itself. By consolidating all components within a small form factor, the end-user should be able to effortlessly secure it in place, connect the ST-4 cable, and be ready to operate without any complications.

Simplicity
The system should not be overly complex. Users should have the ability to easily replace or integrate individual components as needed. The system's functions must also be clear and structured, such that it can be used by users without specialized background knowledge.

Precision
To ensure precise tracking of the lunar movement, the system should not deviate too much from the true position. This way the camera keeps the Moon in sight, even after

long durations of guiding. As a rule of thumb, a maximum deviation from the moon center of a few percent of the lunar diameter is aimed for.

2.2 System Overview

The system consists of a camera and lens combination to capture pictures of the Moon in regular time intervals. A computing unit (Raspberry Pi) then uses this image to find the position of the Moon. If a deviation from the predefined position in the image is detected, a correction command is sent to the switching unit (relay board). This position is set by the user via a button press. The command is then turned into a signal pulse and transmitted through the ST-4 port of a commercial telescope mount, where is interpreted as a steering correction. Figure 2–1 shows a schematic of this process.

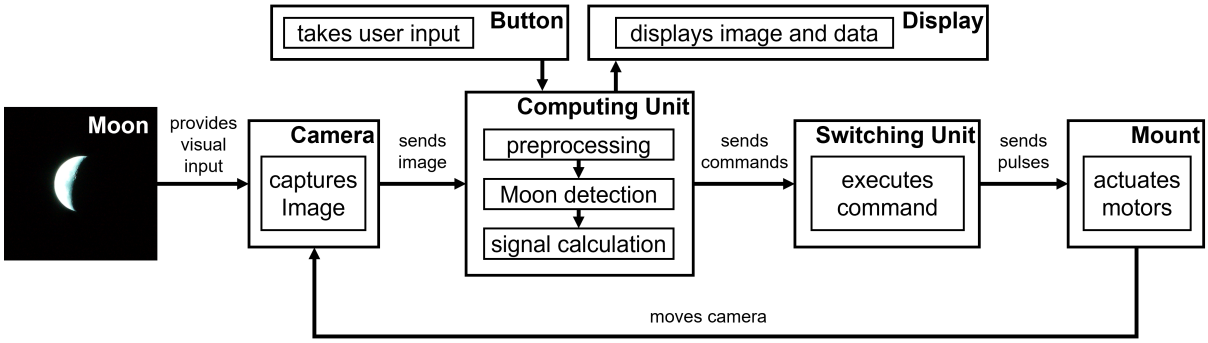


Fig. 2–1: Diagram of the system and its operations

The following will explain the components comprising the system in detail and the key decisions that significantly contributed to their selection.

2.3 Computing Unit

To control the guiding mount, a computing unit capable of processing high quality images is required. It should also allow for a straightforward connection of external devices such as relays and buttons. A Raspberry Pi fulfils both of these requirements and is additionally simple and cost-effective. When guiding out in the field, it can even be powered by a mobile power bank. For this project, the decision was made to use the Model 4B, as it provides sufficient memory as well as a reasonably fast processor. Specifically, an older model was intentionally chosen to avoid potential driver compatibility issues that might arise with newer models. However, newer models can still be utilized if needed.
Caution is advised when using older models, as weaker processors may potentially lead to reduced system performance due to the computationally intensive image processing tasks.
Considerations to use an Arduino based system have been made but quickly dismissed to to the low performance of the processor.

2.4 Camera and Lens

Since the lunar tracking relies solely on optical feedback, the quality of the image received by the computing unit is crucial. Both the camera's resolution and the magnification factor of the lens play a significant role.
While there are numerous camera options available for the Raspberry Pi, most are designed for surveillance or similar applications with a low image quality and wide field of view. As a result, the selection of high-quality cameras is limited to only a few manufacturers. For this project, the decision was made to utilize the Sertronics Raspberry Pi High Quality Camera, which features a resolution of 4056 x 3040 (l_H x l_V) pixels and allows for the mounting of various lenses due to its use of the standard camera C-Mount.[1] This also plays into the requirement of affordability as the user may potentially be able to utilize pre-existing lenses and thus save on additional equipment purchases.
To achieve the required precision, the angular diameter of the apparent Moon on the night sky has to be taken into consideration. This angle changes with the time of day and year since the Moon's orbit is slightly elliptic and the earth's rotation contributes to a distance change to the Moon as well.[2] As an approximation, a median apparent angular diameter of 2000 arc seconds shall be used. To design the system, a precision of 1% of the apparent Moon diameter and therefore a precision of 20" is aimed for. When reviewing different lenses, which fit on the HQ Camera, the viewing angle has to be considered to fulfil this design choice. The viewing angle α_d of a lens can be calculated by using equation 2–1.

$$\alpha_\mathrm{d} = 2 \cdot \arctan\left(\frac{d}{2 \cdot f}\right) \tag{2–1}$$

Here d is the diagonal length of the sensor in millimetres and f is the focal length of the lens, also in millimetres.
Furthermore the horizontal angle α_H and the vertical angle α_V can be calculated by using the angular relations, with β being the angle of the diagonal of the sensor:

$$\alpha_\mathrm{H} = \sin\beta \cdot \alpha_\mathrm{d} \tag{2–2}$$

$$\alpha_\mathrm{V} = \cos\beta \cdot \alpha_\mathrm{d} \tag{2–3}$$

$$\beta = \arctan(l_H/l_V) \tag{2–4}$$

Using the dimensions of the sensor with the requirement of at least one pixel per 20", the maximum viewing angles of the camera and lens combination can be calculated to α_H, max = 81120" and α_V, max = 60800".
Solving formula 2–1 for f and inserting either α_H or α_V results in a minimum focal length of about 15.75 mm with the provided sensor diameter d of 7.9 mm.[1]
After these considerations a lens with a focal length of 50 mm was chosen. This creates a margin of error in the moon detection of about 3 pixels of uncertainty to reach the aimed-for precision of one percent of the Moon diameter, which seemed like a reasonable amount.

2.5 Switching Unit and Wiring

To control the guiding mount, its ST-4 input is utilized. This is an interface commonly found on most astro-cameras and guiding mounts. ST-4 stands for Star Tracker version 4 and dates back to the 1980s, where this kind of guiding interface was first introduced.[3] It is used to steer the mount to one of the four directions (RA+, RA-, Dec+, Dec-) to make small adjustments to the mount's movement and keep the target in the desired position. Since it has never been properly standardized, the exact wiring of the connection has to be adapted to the specific guiding model. In the following the most common architecture is provided, which is also used for this project.[3]

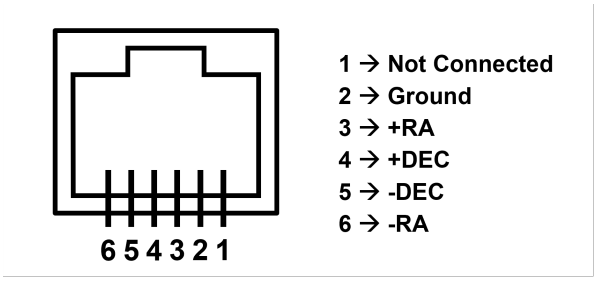


Fig. 2–2: Overview of the most common ST-4 connection layout, adapted a RA change in the northern hemisphere

The ST-4 interface features a standard 6-pin RJ12 connector. Only five of these pins are used by the interface, with four for each direction and one being the common pin (ground). To make a correction in a specific direction, the corresponding pin has to be connected to the common pin, closing the circuit. An overview of the most common layout is provided in figure 2–2. Note that the RA is swapped when using the system in the southern hemisphere.
To establish the connection between the direction pin and the common pin as needed, a relay board is used, which can be controlled by the computing unit. The requirements for this board are that it should have four relays, be compatible with the Pi and have a compact form factor. Additionally, it should be capable of a reasonably fast switching frequency to enable smaller direction adjustments. For this project a 5 V 4-channel opto-coupler relay module was used due to its low price, high availability and compatibility with the Raspberry Pi. Other relay types like semiconductor relays might also be used due to the low operating voltages on the connections.

The Raspberry Pi communicates with external expansion boards most easily through the integrated GPIO pins located on top of the board. These pins can be controlled as needed by the Pi and appropriate software. However, there are certain limitations here as not all pins offer the same capabilities and possibilities. Additionally, for this project, a significant portion of the pins is already occupied by communication with the screen. While the camera input is received via the flat ribbon interface, the remaining pins are available for connecting the button and the relay board. Since the display occupies all of the voltage supply pins of the GPIO interface as well, a second connection had to be soldered to one of the 5V-pins to supply the relay board with power (see 2.8). Due to the low power requirements of both the board and the display, the Pi can easily

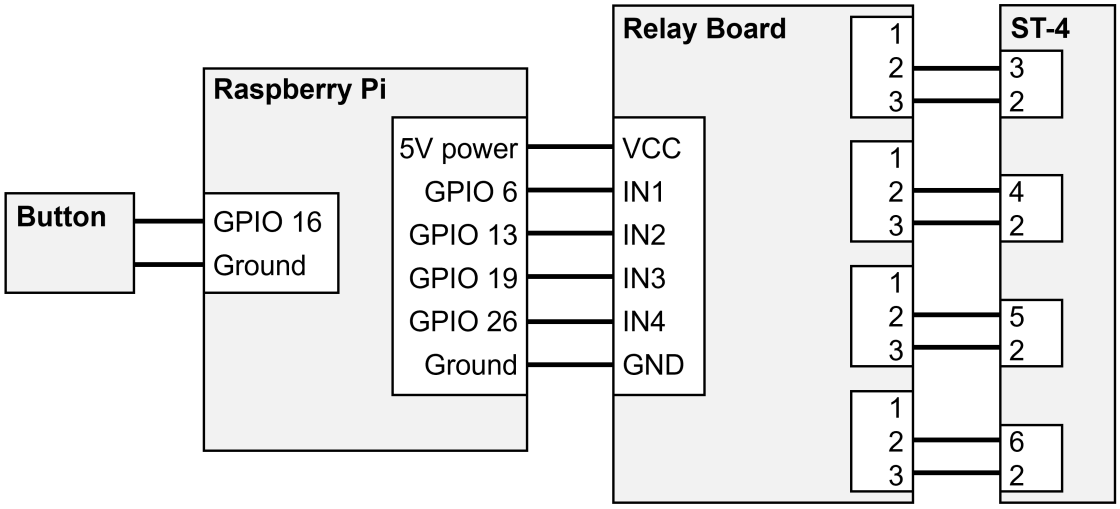


Fig. 2–3: Schematic of the wiring connections from the Raspberry Pi via the relay board to the ST-4 connector. The numbers at the ST-4 interface refer to the connections shown in figure 2–2

supply power to both extensions from a single connection without any issues. Figure 2–3 provides a schematic of the wiring used in this project. Other hardware choices may lead to a different pin layout. Here the relays connect position 1 and 2 when not powered. If the relay is activated, position 2 and 3 are connected, closing the circuit for a given direction.

2.6 Display

To view the current camera output as well as the telemetry of the moon detection some a screen is used. While a normal monitor can be attached to the Raspberry Pi via HDMI cable, the system should provide its own screen to be usable in more remote locations. There are many screens available to choose from so for this project a small 3,5 inch model was used.[4] This is the same size as the Pi itself, which makes it fit nicely into the case.
The display will be covered by a protective sheet of acrylic glass in the end so the touch-screen functionality of the display will not be of much use. However since almost all of the displays with this size feature touch capabilities, there is no difference in cost.

2.7 Case

For the selection of the case, the decision was made to opt for an aluminium case from the brand Innomaker. The robustness of the material allowed for drilling holes to accommodate the button and camera attachment while still maintaining sufficient strength to securely fasten the entire case to a standard telescope rail. Plastic cases might not be sturdy enough to support the weight of the camera and lens. The design of the case allows for upward expansion through the use of spacer screws, although this was not necessary in this particular case. In addition, the enclosure provides enough airflow to prevent the electronics from overheating.

To meet other requirements, such as different mounting mechanisms or alternative hardware, it is advisable to consider a custom 3D-printed case.

2.8 Hardware Integration and Preparation

The hardware integration for this project proved to be a challenge. As the entire system is intended to function as a single unit, all the hardware needs to fit inside the casing. The construction of the case requires mounting the Pi at the bottom. Logically, the screen has to be mounted on top for visibility. Consequently, there is no other space available for the rest of the electronics, including the relay board, except between these two components. The use of a GPIO pin extender for the Raspberry Pi allows for creating a gap of approximately 2 cm between the Pi and the screen, which is just large enough to accommodate the relays. Likewise, the cables, button, and the head of the screw, which is used to secure the camera to the case, can fit inside the enclosure.
The connection to the ST-4 interface of the mount is designed in such a way, that the male connector is already provided by the guiding system. This eliminates the need for an additional cable to connect the device and it can simply be plugged into the mount for further use.
Preparing the relay board for assembly, the wire connections from the relay channels to the ST-4 cable have to be established first. This was done by cutting standard Dupont connector wires to length in order to use up as little space as possible. Instead of routing all four ground wires separately to the ST-4 ground, they are connected directly at the relay board. A close-up image of this can be found in appendix B, figure 2–1. The colors of the cables are chosen to match the colors of the RJ12 interior cables, aiding with connecting the ends later. Note that the colors of the wires do *not* match with the connections from the relay board to the Raspberry Pi. Except for the black ground-wire, the order of the colors is predefined by the attached Dupont-ribbon and is therefore coincidental.

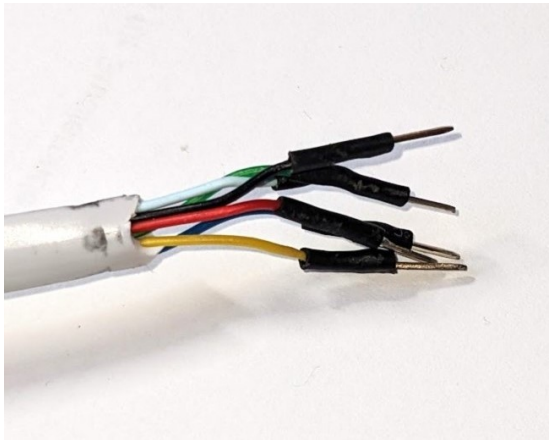


Fig. 2–4: Picture of the modified end of the RJ12 cable for connection to the relay

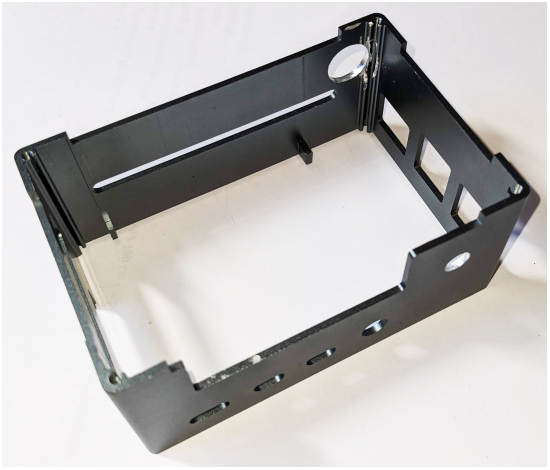


Fig. 2–5: Picture of the Moon guider casing with holes for button and camera screw drilled

To fit the relay wires, the exposed ends of the RJ12 cable wires are fitted with standard Dupont pins. Shrink wraps are used to seal the solder joints (see figure 2–4). The unused connection (here white) is sealed off.
Likewise, pins are also soldered to the wire ends of the button (see 2–8, part no. 8), making them easier to connect to the Raspberry Pi.

As previously mentioned, two holes for each the camera screw and the button have to be drilled into the side of case. In figure 2–5 the holes can be seen on opposite sides of the case. For the button hole, some of the inside case edge is removed as well to fit the tightening nut of the button. This is done by using a dremel or small files. Furthermore the gap between two of the ribs of the bottom plate has to be widened to fit a second camera screw (see 2–8, part no. 4), for securing the case to a standard telescope rail. Drilling this hole is a challenge since the ribs of the plate tend to separate over the drill as soon as pressure is applied. Using two wooden boards and clamping them with the plate sandwiched in between fixes the issue.

To supply the relay board with power one of the 5 Volt GPIO pins of the Raspberry Pi is forked off. This is done by soldering an extra pin in an angle to the corresponding location on the GPIO extension. Additionally, a protecting shrink wrap is fitted around the neighbouring pin to prevent accidental contact. Since the extension is only used to bridge the connections up to the display, the remaining pins are sawed off to provide room for the relay board to be connected. Figure 2–6 shows the modified part.

Lastly a piece of acrylic glass is cut to size as a replacement for the opaque original top cover of the case. For this, the cover of an old CD-case was re-purposed. Holes were drilled into the edges for the mounting screws to attach. Cutting and drilling under water can prevent cracking of the thin plastic. The finished glass plate is depicted in figure 2–7. While this is optional for the functionality of the moon guider, it serves as a protection for the components.

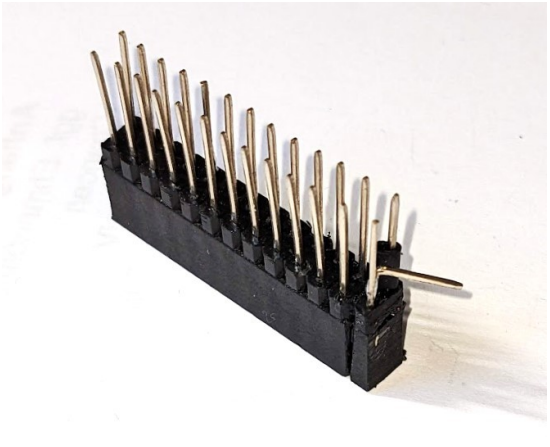


Fig. 2–6: Picture of the modified GPIO extension with soldered additional pin

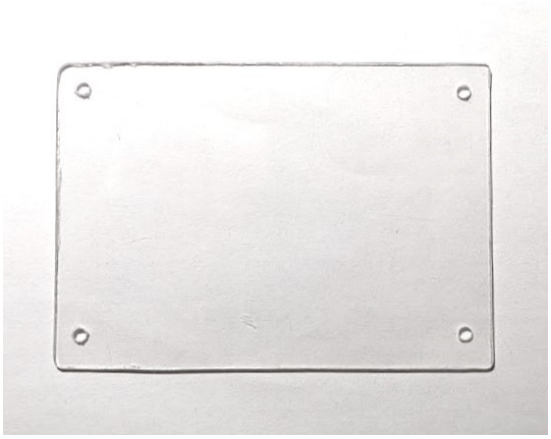


Fig. 2–7: Picture of the custom made acrylic glass top cover

After modifying the parts as described above, the assembly of the Moon guider system can be performed. The multitude of all the parts needed for this are laid out in figure 2–8. In the top down view the parts are as numbered:

1. Case outer wall
2. Raspberry Pi
3. Camera, lens and ribbon cable
4. Case bottom plate with screw hole
5. 4-channel relay board
6. 3.5 inch display
7. Dupont cables
8. Button with soldered Dupont connectors and tightening nut
9. SD card for the Raspbarry Pi
10. Modified GPIO extension
11. Spacer screws and nut
12. Small screws to attach the Pi to the casing
13. 5/16 and 1/2 inch camera screw
14. Modified RJ12 cable

2.9 Assembly

The modular architecture of the case makes it easy to assemble from the bottom up. In the following, the assembly process will be guided through step by step. Reference pictures of the hardware after a completed step are given in figures 2–2 to 2–5 in appendix B.

Step 1: Firstly the bottom plate of the case **(4)** is screwed to a standard telescope rail using the 1/2 inch camera screw **(13)**. A piece of insulating tape is added to the head of the screw to prevent accidental short circuits of the Raspberry Pi, since it will be very close to the underside of the circuit board. The short spacer screws **(11)** are tightened to the bottom plate using the small screws **(12)**. These are included with the casing and will later secure the Raspberry Pi to the case.

Step 2: Before inserting the Raspberry Pi **(2)** into the case outer wall **(2)**, it is recommended to put in the 5/16 inch camera screw **13** first, as it can be hard to fit it in once the circuit board is in place. The stack can now be placed on top of the assembled bottom plate from step 1, such that the spacer screws fit the holes in the Raspberry Pi board. They are then screwed tight with the remaining spacer screws. The spacer near the USB-C interface of the Pi is extended by the spacer nut. It will later be used

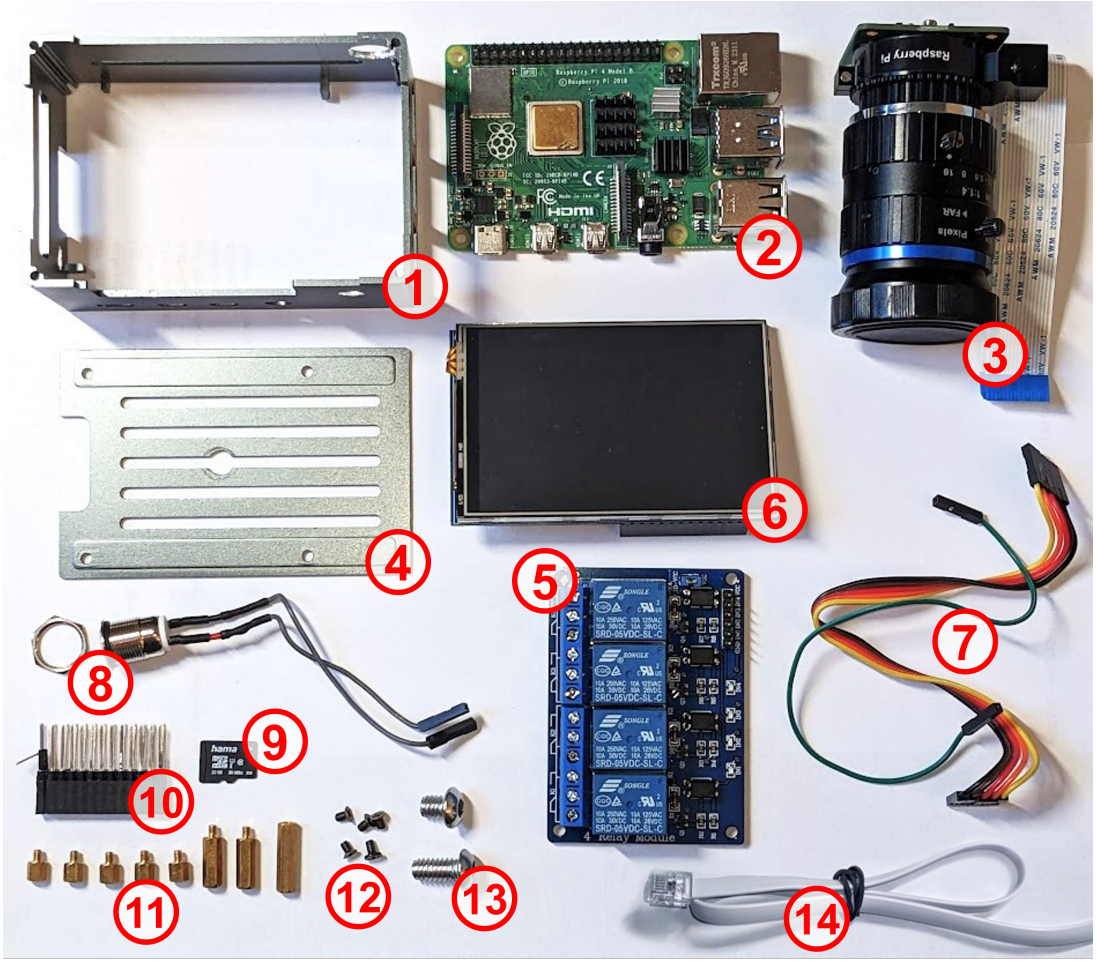


Fig. 2–8: Top view of all the essential parts of the Moon guider system laid out and numbered.

to hold the relay board in place.
Now the button **(8)** can be inserted and tightened. The chosen case provides an opening with the right shape to fit the RJ12 cable **(14)** after some minor filing. Through this hole the cable is threaded. This might take some effort since the assembly of pins is wider than the hole, so they need to be put through one by one. Lastly the modified GPIO extension **(10)** can be plugged in. Due to the small margins it is recommended to plug in the connector **(7)** for the 5 V supply (here green) first.

Step 3: Now the camera **(3)** can be screwed in and tightened. The relay board **(5)** is connected to the GPIO pins as well as the RJ12 pins. The button and the ribbon cable of the camera are also connected to their corresponding interfaces on the Raspberry Pi. For the exact wiring see section 2.5. In order to place the ribbon cable flat along the bottom, the connector may need to be bent 90 degrees. Note that the relay board is placed upside down on the rest of the assembly.

Step 4: By placing the rainbow-coloured Dupont-cables to the side between the relay blocks and the GPIO extension, the relay board can be mounted flush into the casing. When everything is tidy and the pin extensions are visible next to the relay board, it can be fixed in place by screwing it into the spacer nut in the corner. As this is a very tight fit, caution is advised to not break off any pins or pull out cables in the process.

Step 5: Now the screen **(6)** can be plugged into the GPIO extension, making the system usable. To finish the assembly, the custom made acrylic glass plate is screwed to the top of the case. Figure 2–9 shows the assembled product.

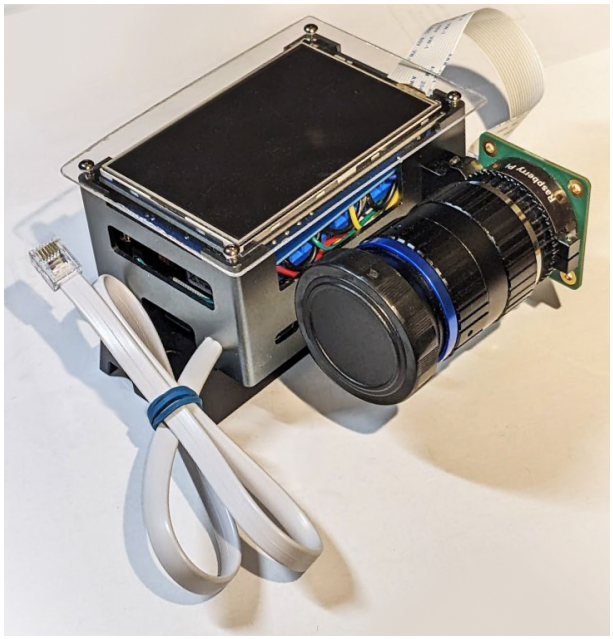


Fig. 2–9: Picture of the Moon guider with the relay board fully assembled

3 Software

To detect the Moon and send the adequate correction signals to the mount, several intermediate steps are required, necessitating the use and development of tailored processing software. Initially, the image of the Moon captured by the camera must be analysed. The goal is to determine the position of the Moon on the image with the highest precision possible. This determination should work reliably under various conditions, including clouds, haze, changing lighting conditions, and different lunar phases. To speed up the process, the images undergo preprocessing, such as noise reduction through blurring and removal of color channels, not needed for the detection.

Based on the Moon's position in the image, its deviation from the chosen reference point is calculated and an appropriate signal is generated for the relay board. The relay board then implements the signals from the Pi by connecting or disconnecting the pins of the ST-4 interface of the mount accordingly. The mount then starts moving the camera in the desired direction, providing visual feedback to the camera and the cycle repeats. Meanwhile, both the camera image and useful data are transmitted and displayed on the screen. A schematic representation of this process is depicted in figure 2–1.

For signal processing and programming the functions, the programming language Python is used. It is highly compatible with the Raspberry Pi and offers pre-built modules that are utilized in the processing software. Furthermore, it is easy to use and widely adopted, which favours the adaptation and further development of this system.

3.1 Software Architecture

To keep the software structured, a modular approach is realized. The individual functions are grouped thematically and incorporated into separate Python files. This approach makes testing very convenient, as functions can easily be swapped and used individually in a testing environment. It also helps keep the code organized for future use and makes it easier to identify and fix errors.
Based around the main file, the circle detection, relay actuation and input parameters are handled in separate entities. For ease of use, the user can specify these parameters in the `config.ini` file. Here different profiles can be created and edited. The configuration file is then read by the program, which creates a class to incorporate all of the chosen parameters. They can then be easily accessed in each of the respective files.
The input parameters cover a wide variety of different functionalities and can be categorized into four groups: Guider settings, camera settings, image detection parameters and general settings. By using this wide range of setting options, the program is flexible and adaptable, making suitable for easy adaptation to possible variations in hardware architecture. Table 3–1 provides a list of these settings.
For a more detailed look into all of the different components and documentation of the software, see appendix C.

Tab. 3–1: Setting parameters for the Moon Guider. Can be specified in `config.ini`

Parameter	Default	Description

`pin_ra_down`
`pin_ra_up`
`pin_dec_down`
`pin_dec_up`
`pin_button`
	19
	13
	6
	26
	16
		Pins of the Raspberry Pi GPIO for connecting the relay ($\pm$ DEC, $\pm$ RA) and button. Pin numbers can be found in the Raspberry Pi user manual of the respective version. Depends on hardware set-up and wiring.

`margin`	0.3	Margin of pixels in which the target can move without relay activation
`pulse_multiplier`	0.1	Multiplication factor for calculating the duration of the relay pulse (deviation in pixels · `pulse_multiplier`)
`cloud_mode`	`repeat`	Determines how a loss of target due to clouds or other obstructions will be handled. `None`: No cloud handling, guiding will pause `repeat`: Last set of pulses is repeated
`record_buffer`	100	Number of recorded steering signals to iterate. Only relevant when cloud mode is set to repeat.
`rotate`	90	Rotates the directions of the detected deviation clockwise to account for a angled camera. Possible rotation values are 0, 90, 180 or 270.
`image_width` `image_height`	4056 3040	Dimensions of the image sensor in pixels
`in_scale`	1	Scales the camera image before processing.
`image_buffer`	6	Size of the image buffer of the camera stream.
`blur`	5	Kernel-size of the median blur applied to the image.

`dp`
`param1`
`param2`
	2
	300
	50
		Parameters for the `HoughCirlces` detection algorithm. See section 3.3.

`buffer_length`	20	Number of target locations which will be averaged to smooth out the guiding
`overlay`	`True`	Show white bar with information
`out_scale`	0.12	Scales the output image to adjust to different screen sizes. Has no effect on accuracy.
`show_cam_feed`	`False`	Show high frame-rate camera stream on start-up. Can be useful so point the guider.
`do_relay_test`	`False`	Perform a relay test on start-up to check for correct wiring and configuration.
`export_to_excel`	`False`	Ability to save guider data to excel file after exiting.

3.2 Camera Feed Integration

The primary input for the Moon guider system is the camera image of the Moon, which is captured using the Raspberry Pi camera. The camera is controlled using the `Picamera2` module, a pre-built Python library. This module allows for the configuration of specific profiles for the camera, which then adjust settings for the subsequent camera output. To ensure the highest precision of the guider, it is advisable to read the maximum resolution of the sensor. Due to the slow movement of the Moon, at just under 15 arc seconds per second, the system does not require an extremely high processing speed. Most of this movement is even compensated by the standard motion of the mount.
To increase the responsiveness of the system, the processing speed can be improved by adjusting the image buffer count in the configuration. A higher buffer count enables faster processing of image data and consequently quicker adjustment of the overall system. However, the size of the buffer depends on the available memory of the computing unit. In the case of the Raspberry Pi 4B used here, the optimal buffer size is at approximately 6 frames. With a larger buffer, there is a risk of memory overflow, leading to program crashes. For other models this value may vary accordingly.
Conveniently the image capture function provided by the `Picamera2` module, returns the image as a `numpy`-array, which can be directly used by the Moon detection software.

3.3 Moon Detection

Detecting the Moon in the captured image takes multiple processing steps.
Firstly, the image is transformed into the grey scale format, removing the color channels. This is the required format for the subsequent lunar detection algorithm and additionally speeds up processing due to less amount of data being processed. Then a median blur is applied to the image. This blur helps by removing noise from the image. Depending on the resolution of the image, the kernel for applying the blur may need to be resized. Tests have shown that a slight blur of around 4 pixels seems to bee the sweet-spot in smoothing out the edge of the moon, leading to a more consistent target detection. Too much blur introduces an uncertainty for the edge detection algorithm.
The entire process takes place within the `preprocessing` function, which takes the original image as input, processes it based on the provided configuration, and then outputs the processed image.

To do the image processing as well as the Moon detection the python module `cv2` was used. The module `cv2` is part of the OpenCV (Open Source Computer Vision Library), a tool from the field of computer vision and image processing. This library offers a wide array of functionalities for real-time image processing applications. One of the prominent functions in `cv2` is `cv2.HoughCircles`, which is a function for detecting circular shapes in images. This function is ideal for applications like the Moon guider system, where the primary objective is to identify the a circle in an image. The `cv2.HoughCircles` function implements the Hough Circle Transform.
The function works by scanning the image for points that form a curve and mapping them to potential circle centres in an accumulator space. When these mapped points

converge significantly, it indicates the presence of a circle.[5] While this is a very brief explanation for method of this function, the details are beyond the scope of this term paper.

Key parameters of `cv2.HoughCircles` include:

- **`dp`**: Inverse ratio of the accumulator resolution to the image resolution. Affects the size of the accumulator array which is used for the processing. A lower value results in a more precise detection but increases computation.
- **`minDist`**: The minimum distance between detected circle centers. Adjusting this parameter prevents multiple detections of the same circle. For this application the minimum distance was set to the width of the picture as only one circle per image should be detected.
- **`param1`**: Threshold value for the `canny()`-edge detection, the function is used to get the boundary of the circle.[6] Increasing this value helps with noise but may not detect the Moon in low contrast situations like haze or clouds.
- **`param2`**: Accumulator threshold for the circle detection. This determines how "circular" the circle needs to be for the function to detect the circle.[7] A high value improves performance when the Moon is close to full but may lead to no detection at new Moon.
- **`minRadius`** and **`maxRadius`**: Minimum and maximum radius of the detected circle. Adjusting this to the current apparent Moon size can increase detection accuracy and reduce false positives. In the code this is done by detecting the moon with more liberal bounds first and then locking the radius by pressing the guider button.

After detecting the circle in the image, the function returns the x- and y-coordinate of the center in the image as well as the radius. These parameters can then be used as the target coordinates for the guider. Depending on the time of day and month, the captured picture of the Moon can vary substantially. Due to varying brightness, clouds, haze and especially different Moon phases, the `HoughCircles`-detection algorithm's performance will not be constant for the same set of parameters. For this reason, different parameter profiles have been created to fit those situations best. An overview is given in table 1–1 in the user manual. An extensive report on how these values were chosen can be found in section 4.1.

A different approach to detection the Moon and specifically its deviation is to take two pictures and look for differences between them. Using this technique a deviation vector can be calculated. However this method was dismissed due to its high susceptibility to haze and clouds. Moving cloud sheets can easily throw off the motion detection, if they are big enough. Using the Hough-method the Moon can be detected even at reasonable cloudy and hazy conditions with satisfactory precision.

3.4 Signal Output

To steer the guider in the right direction, a correction signal for the mount is required. As mentioned earlier, the mount is already moving at lunar speed automatically, so the entire movement doesn't need to be guided manually. Slight deviations in the guider's alignment can lead to significant inconsistencies in the long-term. It is precisely these deviations that the Moon guider aims to compensate for. Performing small course corrections is done by sending short pulses of movement commands to the guider, as long duration commands have proven to lead to unpredictable behaviour (see section 3.6). After calculating the deviation of the Moon from the target position, the software first checks if this deviation is inside the defined `margin`. If not, a signal will be sent to the relay board. To make the steering input more sensitive, as the target is approached, the length of the signal is dependent on the amount of deviation. The bigger the deviation, the longer the signal, up to a maximum of three seconds. The signal length is calculated by multiplying the deviation in pixels times the `pulse_multiplier`, a value set by the user. The optimal value of this parameter is dependent on the specific mount which is used since not all mounts have the same amount of correction movement when using the ST-4 interface. In this specific case, using the Skywatcher Star Adventurer GTI, a `pulse_multiplier` value of around 0,1 worked best.

3.5 Error Mitigation

Since the Moon Guider relies solely on visual feedback from the camera, the consistency of the Moon detection is crucial. But some external influences can not be controlled by the software. Clouds and fog are the main factors contributing to detection errors. During an otherwise clear night, clouds or fog patches can frequently obscure the view of the Moon for a certain period. During this time, the guider cannot locate the Moon correctly, leading to a loss of precise tracking. However, the general direction should be maintained to allow for resumption when the Moon becomes visible again. Additionally, structures within illuminated clouds can be recognized by the software as circles, leading to false positives. To address these situations, various mechanisms have been developed.

3.5.1 Handling Clouds

In the program, two ways of handling clouds were implemented to choose from. This setting can be addressed via the `cloud_mode` parameter. If no Moon is detected in the image for five times in a row, the cloud mode is activated.

`cloud_ mode = None`
As soon as the cloud mode is activated, the guider is paused and remains inactive until the Moon is detected again. The guiding then resumes automatically. As this is the simplest form of cloud handling, it requires the least amount of computation. For longer obstructions of view, the Moon guider will inevitably loose track of the target and not be able to recover.
Using the viewing angle of the camera (on the RA axis) of 25600" (as measured by

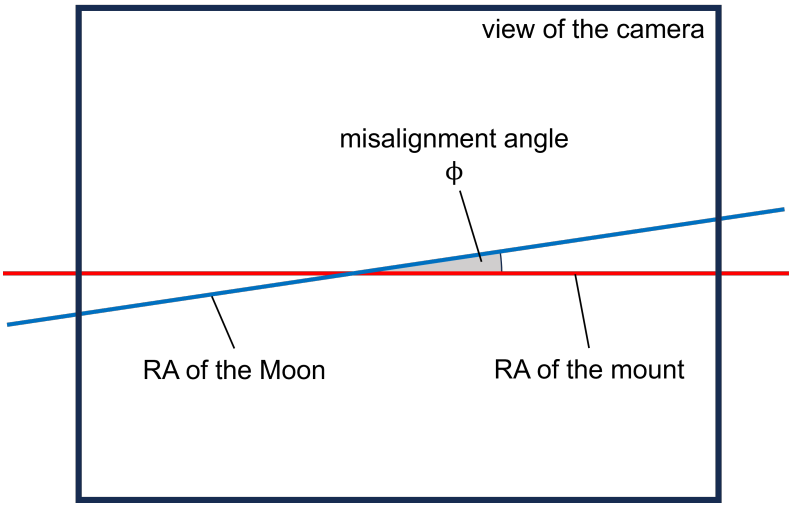


Fig. 3–1: Illustration of the effect of a misaligned altitude axis during un-guided operation of the mount.

hand) together with the apparent speed of the Moon in the night sky of about 15” per second, we can calculate that the Moon would take just over 14 minutes to leave the view of the camera. This assumes that the Moon was at the center of the image at t_0, at which point the mount motors were to fully turned off. Now considering that the mount’s RA movement continues even if no commands are sent to the ST-4 interface, the only way for the guider to loose sight is due to a wrong alignment of the mount. While the mount can be precisely set in the cardinal direction using a compass, the altitude alignment of the guider is not so trivial. Uneven terrain and poor visibility can quickly lead to deviations here. Assuming a small misalignment angle ϕ, this results in a tilted RA axis. The misalignment now induces a deviation from the Moon in the DEC direction by the sine of the misalignment angle between the RA of the mount and the true RA of the Moon, which is illustrated in figure 3–1. For small deviations, this effect is therefore minimal. It is safe to assume that for ϕ of a few degrees, the guider will be able to regain the target after well over 14 minutes.

This behaviour has been confirmed during tests with the live Moon. Figure 3–2 shows a plot of the x and y position of the Moon in the frame. The camera is aligned with the RA and DEC axis due to the mount being oriented and therefore correspond to the x- and y-axis of the picture respectively. During this test the camera is covered at around 110 seconds, which makes the guider switch to cloud mode. This was set to `None` in the test so the guiding is inactive during this time. After 140 seconds of no relay activity the cover is removed and the target is regained. The dashed lines show the linear approximation of the drift during this time where only the native motion of the mount guides the system. While the deviation of the x-axis (RA) shows close to not drift at all during the inactive phase, the deviation in the y-direction (DEC) is rather significant at about 55 pixels. The great difference in drift of the two axis is most likely due to the effect described above and depicted in figure 3–1. At this amount of misalignment of the mount, the guider would be unrecoverable after about one hour.

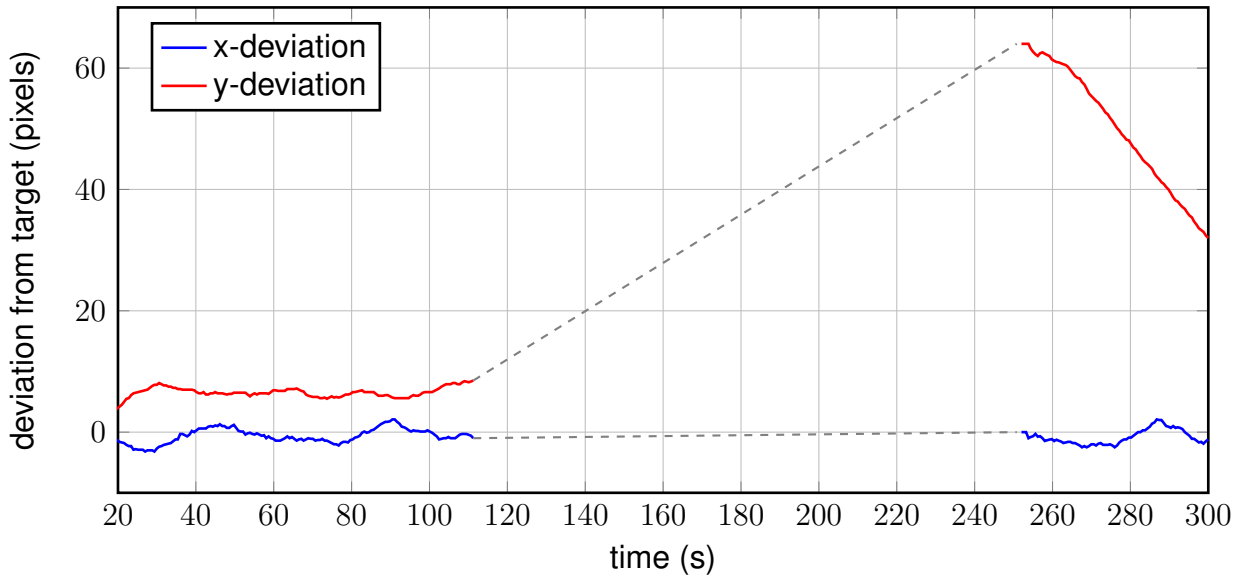


Fig. 3–2: Plot of the Moon position in the frame during a live test with `cloud_mode = None`.

`cloud_ mode = repeat`
When the cloud mode is set to `repeat`, a history of relay activation signals is recorded. The number of entries in this recording can be set via the `record_buffer` parameter. When the target is lost, the relays execute this history until the target is back in sight. To predict the movement of the Moon using this method, the assumption is made that the Moon moves in a linear path uniformly across the night sky. Since the path describes more of a sine wave, this approximation is only valid for limited periods.[8] Beyond a certain duration of loss of contact, the guider will inevitably deviate from the path of the Moon. However, for smaller cloud patches and fog banks, the corrections are still sufficient.
Figure 3–3 shows this mode in action. In the test, similarly to the one performed with cloud-mode `None`, the guider follows the live Moon for around 105 seconds when the camera view is blocked. The guider then goes into `repeat`-mode until the cover is removed at the 250 second mark. During this time the recorded pattern of 100 signal activations is repeated. Over the total of about 145 seconds the drift in the x-direction (here RA) and y-direction (here DEC) only amount to around 2.5 and 15 pixels respectively, as indicated by the dashed lines. At this rate the Moon would slip out of sight at the top of the frame after almost 3 hours, provided a centred Moon in the beginning. Since the purpose usage of the guider presumes a clear night in the first place, a cloud overhang of this duration should not occur. The precision is therefore sufficient.

3.5.2 Other Error Mitigation Measures

Further measures have been taken to make lunar tracking as precise as possible. As previously mentioned it is of utmost importance for the precision of the entire system, that the visual lunar detection is as precise as possible. Even though the parameter

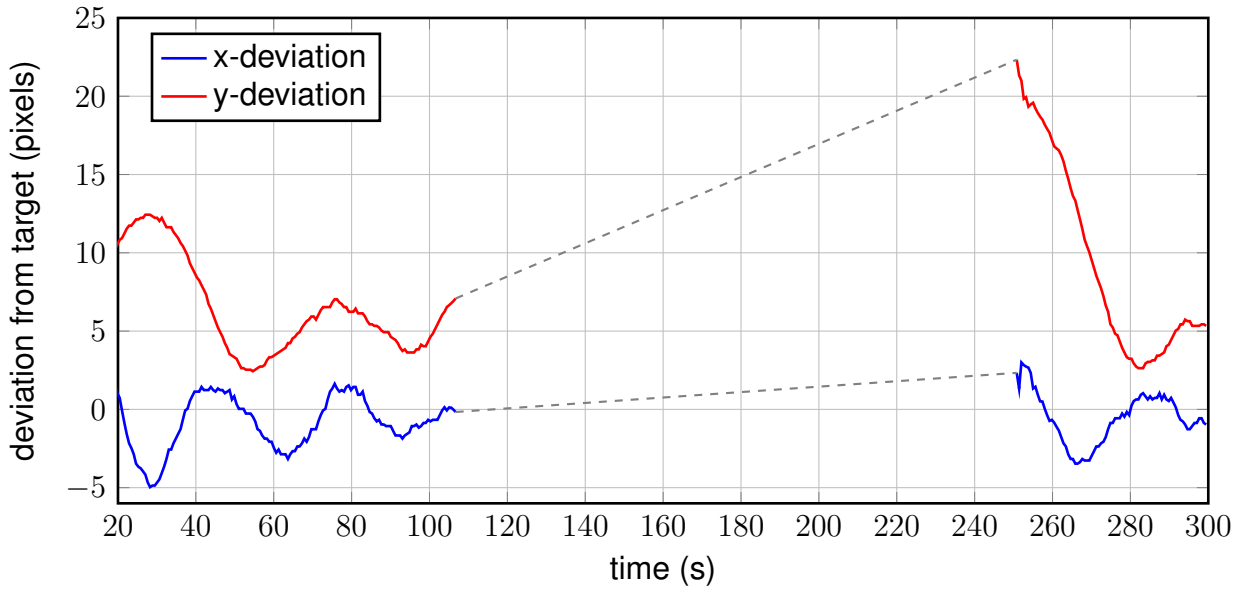


Fig. 3–3: Plot of the Moon position in the frame during a test with `repeat`-mode. `record_buffer` was set to 100 frames, at an average FPS of 1.7

study already provides the optimal settings for the `HoughCircles`-algorithm (see section 4.1), slight fluctuations remain. These are attributable to air fluctuations, clouds, camera noise and other influences. Even a tiny change is sufficient for the detection of the Moon to be shifted by a few pixels.
To mitigate these fluctuations, a buffer and averaging function has been added to the guider. This buffer, the length of which can be adjusted via the `buffer_length` parameter in the program, collects the determined target positions. Instead of using the actual target values from the current frame, the guider uses the average of the target buffer for the lunar position. In this way, small fluctuations are compensated over time. It has to be noted that this averaging results in a slight delay of the movement of the guider to the actual Moon. The amount of this delay can be approximated by the following formula, again assuming a linear static motion of the moon:

$$\Delta_{\text{buffer}} \approx \frac{1}{2} \cdot \texttt{buffer\_length} \cdot \Delta t_{\text{frame}} \cdot \omega_{\text{moon}} \tag{3–1}$$

Here Δ_{buffer} is the deviation of the system in arc seconds from the actual Moon, ω_{moon} is the angular velocity of the apparent Moon in the sky in arc seconds per second and Δt_{frame} is the average calculation time it takes for a frame to be analysed in seconds.
While this averaging induces an offset of the guiding system lagging behind the Moon, it ensures that the random jitter of the Moon detection is mostly cancelled out. To account for the offset, the center reference point can be adjusted.

To further improve the detection quality of the Moon in the image, a reference Moon radius can be set. While starting up, the `minRadius` and `maxRadius` are set to high values. A video feed, including a visual marker of the detected Moon is then shown. The user can then confirm the correct detection by pressing the button. This ensures

that only circles of the correct size are taken into account, further eliminating false positives.

As clouds come in many shapes and sizes, sometimes they display ring- or circle-like features which could falsely be picked up by the detection algorithm. To prevent this from happening, the Moon guiding system features a threshold for identified circle positions. Hence, circles which are more than 100 pixels away from the last known target position are filtered out, as they are most likely false positives.

3.6 Challenges

During the development of the Moon Guider, several challenges had to be overcome. The control of the mount proved to be non-trivial, in particular. The first approach to steering the mount was by sending a continuous signal to move in a specific direction, until the target was reached. This results in long duration signals, especially during the initial target acquiring stage. During a first test, this approach was implemented and its performance tracked by logging the x and y position of the target on the image. These correspond to the RA and DEC axis of the guider respectively. Figure 3–4 shows the deviation from the target in the x and y axes separately over time.

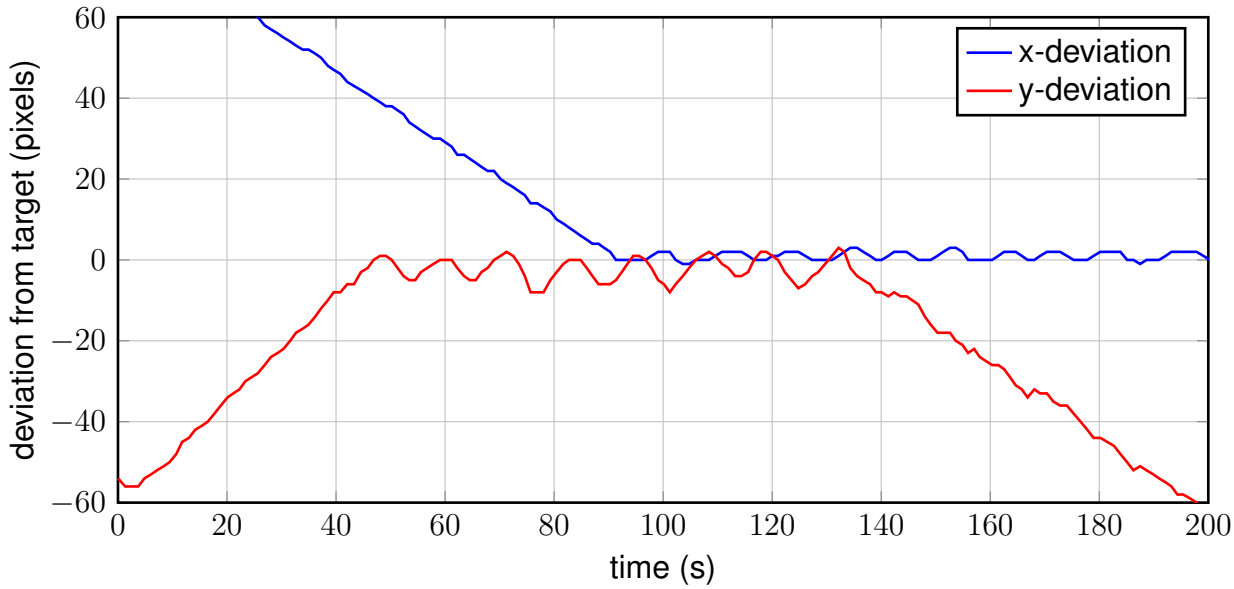


Fig. 3–4: Plot of the error while guiding with continuous direction signals. Failure occurs after 140 seconds.

As can be seen, the deviation from the target position slowly approaches zero and remains in the vicinity, following the target as planned. However, after about 140 seconds, the mount starts to deviate from the target in the negative y-direction, while the x-axis continues to be relatively stable. This deviation in the y-direction was caused by the DEC motor not activating any more, even though the corresponding relay had been activated. Similar occurrences of this happening showed no clear pattern as to why the steering of the guider stopped. The system started out with normal behaviour and

then the deviation in one axis started increasing at a seemingly random point in time, sometimes taking 20 minutes.
Initially, the function of the relays was checked. Theoretically, stuck relays could be a reason for the guider's malfunction. As a first effort to free a possible sticky relay, a new functionality was added to the guider's software. It was able to detect a continuous increasing deviation from the target and send a pulse to the corresponding relay in order to free the mechanics. Surprisingly this method seemed to get the guider back on track, which was shown by tests such as depicted in figure 3–5. Here the error appears two times. The guider then pulses the relays and the steering returns it back to the target. Interestingly this did not seem to work consistently as can be seen from the different maximum deviations for each error in figure 3–5. The software in this test was set up to detect an increasing deviation after 20 steps of continuous divergence, which corresponded to about 30 seconds. In the plot, it can be seen that the first deviation is not corrected until about 60 seconds, when and the guider changes its direction. The initial pulsing of the relays did not seem to have any effect. For the second deviation it even took three attempts.

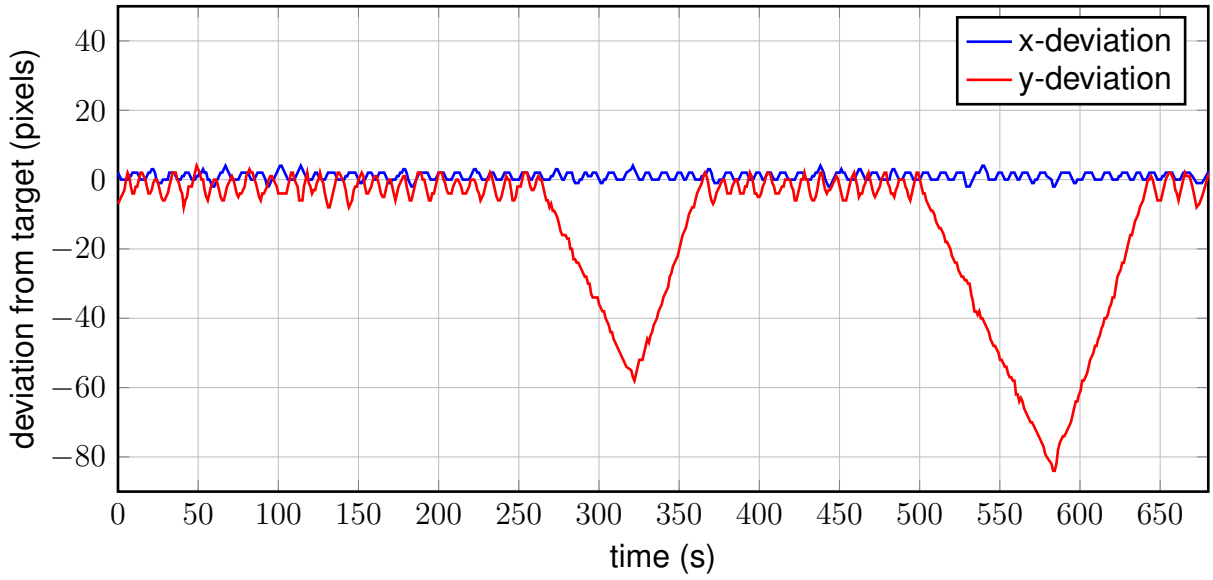


Fig. 3–5: Plot of the same error. The software recognizes malfunction and pulses relay to free up sticky mechanics.

Further investigating this issue, the exact cause had to be identified, as it could still be an issue with the mount as well as the guider. To rule out a sticky relay, additional relay boards from a different manufacturer were connected to the system and tested but the same error was observed. This indicated that the error was most likely to be caused by the mount itself.
After researching and writing with the manufacturer of the mount, it became apparent that the length of the signal seemed to be the problem. Some mounts only expect short correction signals and may be confused by the long duration signals, which were sent by the guiding system. However, since the software did not allow for parallel execution

of various tasks, the minimum pulse time was bound by the speed of data processing. The shortest possible signal was therefore initially limited to about 0.8 seconds, which was the time it took for the guiding system to compute the Moon location in one frame.

One way to activate the relays in parallel with image processing is through threading. Threading in Python allows multiple instances of a program to be created, which can then be executed simultaneously. Since the activation of the relays, aside from the actual control signal, mainly consists of waiting time, this technique can also be handled well with the limited performance of the Raspberry Pi. The software was then rewritten from activating the relays between the frames to pulsing the relays for a specific amount of time. This allows the steering signal to be arbitrarily short. The signal length is now determined by the magnitude of the deviation in the corresponding axis in pixels times a multiplier `pulse_multiplier`. Scaling the duration of the signal ensures a softer, more precise steering in close proximity to the target, while allowing fast movement during initial target acquiring. While reworking the code, the efficiency was also improved to around 0.6 seconds per frame which makes the software much more responsive.
When testing the new `relay_handling`, the error does not occur any more. Various tests over long durations of time have not shown any unexpected behaviour by the mount. For more detailed information about the tests see chapter 5.
Possibly the internal control of the mount did not expect long duration signals and turned the motor steering off, misinterpreting the signal as an error. However this theory has not been confirmed with certainty due to a lack of clarifying information from the manufacturer.

4 Testing and Development

Due to its orbit around the Earth, the Moon spends about half of the month during daylight hours and the other half at night. To test the system optimally, a clear night and unobstructed view of the Moon are required. These limitations make the rapid development of the Moon guiding system challenging. A simulation environment had to be found.
For this, the testing was split into two parts: Moon detection and relay actuation and steering. These two functions could initially be tested and developed independently from each other, before integrating them into a joined system.

4.1 Moon Detection

As previously discussed, the precision of the guider depends heavily on the choice of parameters for the `HoughCircles` algorithm. Finding the best values for these parameters is not trivial, since they all depend on each other as well as the picture which the algorithm is analysing. To find the best combinations a grid search is deployed. This is a search technique normally used for hyper-parameter tuning in machine learning models. It is an exhaustive search in which a list of possible values is defined for each of the parameters in question. The algorithm then checks every single combination and tracks their performances. This way the best performing combination of parameters can be found.
In the context of Moon detection a grid search concerning the parameters `blur`, `dp`, `param1` and `param2` is deployed. Each combination is then given a stack of 15 images of the Moon. The process is then repeated to find the parameters for the other Moon phases.
The images are generated by taking an original image, captured by the Raspberry Pi Camera, and augmenting it by varying brightness, contrast and noise. This ensures that the position of the Moon is the same in every picture of a given set, while generalizing the test for different conditions. Over each combination of parameters and set of Moon images the values of the Moon coordinates and radius are logged. The best performing combination can then be found by choosing the one with the least amount of deviation between the detected values per set. If two or more combinations have the same performance, the one with the lowest computation time is chosen. The results, which are published in table 1–1, are recommended benchmark values which can be used with the Moon Guider system. For more detailed results of the grid search consult the excel-files associated with this paper or visit the GitHub. Due to the size of the tables, they have not been included in this report.

4.2 Relay Actuation and Steering

For the development of the relay actuation and steering function, the Mount is initially fitted with a laser pointer and pointed towards a wall. The relays for the corresponding mount axes are then actuated one by one. By tracing movement of the laser, the

movement of the mount can be detected. Figure 4–1 shows a picture of the connection from the Raspberry Pi to the ST-4 interface in these early tests. Here the white wiring is the common pin (ground) and the coloured wires are corresponding to the directions of motion.

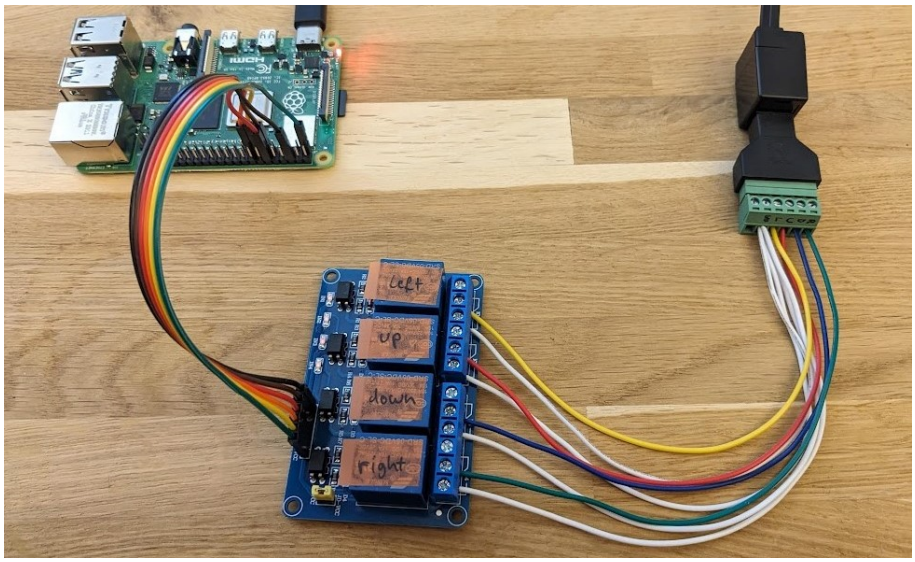


Fig. 4–1: Set-up for testing purposes: top-left: Raspberry Pi; middle: Relay board; top-right: Debug connector to ST-4 interface

After the connection to the mount is confirmed to work as intended, the camera is added to the system. Since now the target can be located and logged, this provides a more detailed look into the ongoing movements. To measure this with maximum accuracy, the target detection must be as consistent as possible. Therefore, optimal conditions are created for the circle detection. As a test set-up, a white paper circle with the correct size is placed into the camera view. If the correct target is found, the mount is shifted such that it has to move back to the target by actuating the motors. The data collected from these kinds of tests, provided insight into the reaction of the mount to the input signals, that lead to the discovery of the continuous signal problem which is discussed in 3.6.
Figures 4–2 and 4–3 show pictures of the system as well as the display output during this test respectively. The single components are held in place using rubber bands and anti-ESD (electro-static discharge) film as it is not yet assembled.
In later iterations the Moon Guider is tested with moving targets. To do this at times where the Moon is not available, a screen is placed 2 meters in front of the guider and the mount is aligned accordingly. This was done by using a smartphone, running star-gazing application. Utilizing the gyro-sensors of the phone, the mount was roughly pointed into the direction of north star. A video is then played on the screen, depicting an animated white disk on a black background. With the disk being exactly the right size and movement speed as the real Moon, all of the functionalities of the Moon guider can be tested in the simulation environment.

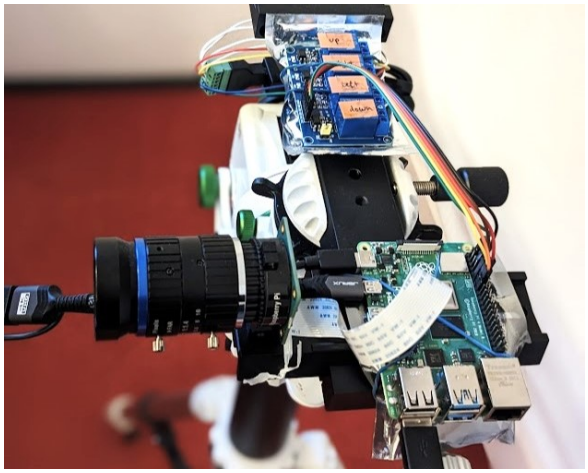


Fig. 4–2: Picture system as it was used for testing, mounted on the telescope stand

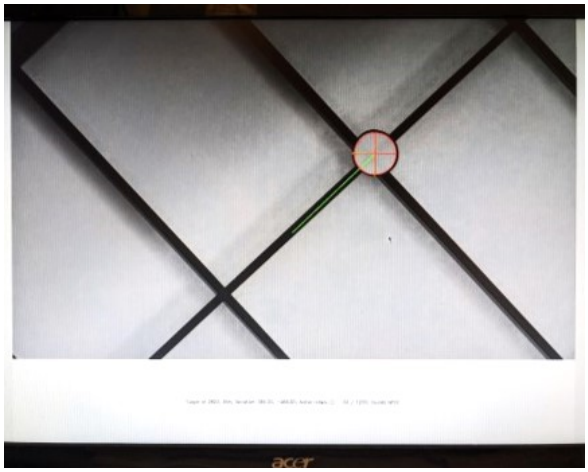


Fig. 4–3: Picture of the display output of an early system test. In the view is a wall with a white paper circle attached to it so Simulate the Moon

4.3 Integrated System

With both of the sub-functions working, full system tests can be executed. The guiding precision is evaluated by measuring the maximum range of the deviation over a fixed period of time. The lower this range, the better the guider is able to hold the target in the desired position. The tests show that the guider enters a control feedback loop when staying close to the target. This leads the guider to oscillate over the target position. This oscillation heavily depends on the `buffer_length` parameter, as was tested in the experiment depicted in figure 3–1 in appendix C.4. Here the simulated moving target was tracked two times with for the same configurations, except for a different `buffer_size` parameter. During the time the deviation from the target was logged. For simplicity, only the deviation on the x-axis is depicted here, however the y-axis deviation looks very similar. A bigger buffer clearly dampens the oscillations of the guider significantly while also lowering the frequency.
The `pulse_multiplier` also changes the behaviour of the guiding. It changes the duration of the computed steering signal. This can lead to overshooting the target, ad a too strong steering motion is applied. An example of this is shown in appendix C.4, figure 3–2, where the same test is done twice, again in the simulation environment, only changing the multiplier. After many tests, a `pulse_multiplier` of 0.1 proved to be the best fit. However this value might change for different models of mounts as they might vary in their steering magnitudes.
The `margin` parameter did not seem to have a significant effect, except for a shifted pointing center in some cases.
To monitor the guider's performance during these tests, the option to log the resulting data and export it to an excel-file was added. This feature is still operational and can be used to review the guider performance after operation.

5 Results

In the simulation environment as well as the live Moon, the guider has been tested extensively. It has been proven to be reliable under real conditions within a margin of a few pixels of deviation from the target.

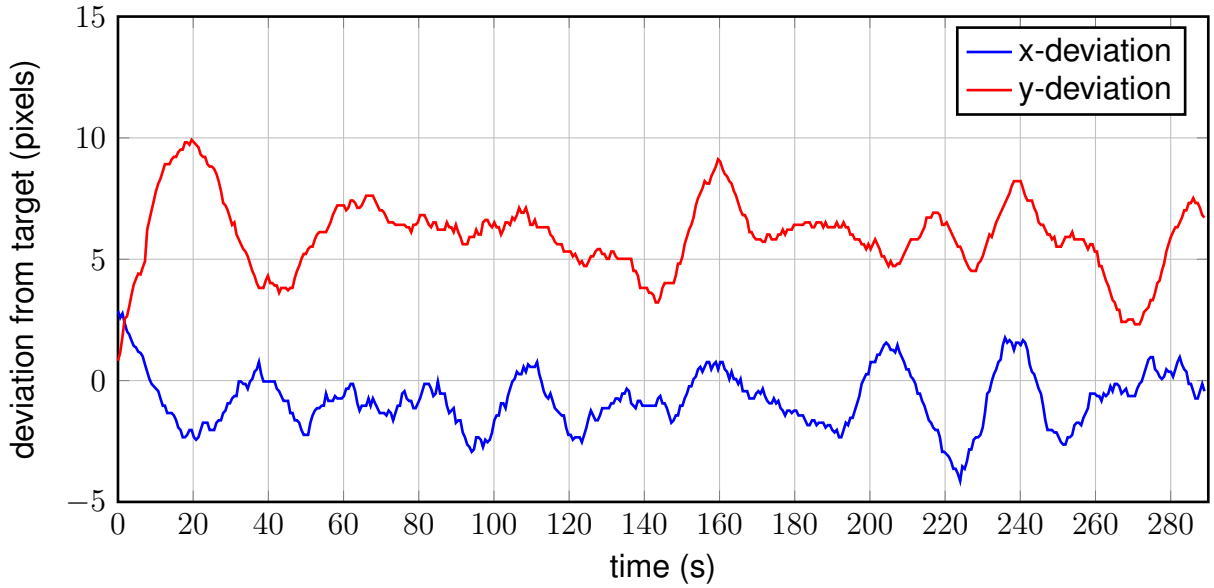


Fig. 5–1: Plot of the a successful guiding test with the live Moon. `buffer_length` = 20; `pulse_multiplier` = 0.1

Figure 5–1 shows the plot of a test performed on the live Moon. As can be seen, the deviations fluctuate up and down throughout the test. An oscillation occurs at times, where the guider corrects and then overshoots the target. Since this overshooting is within a range of a few pixels, it is acceptable for the application. It has to be kept in mind that this deviation refers to the position of the Moon calculated by the guider. As a result, it is subject to constant minor fluctuations in detection that the Moon does not actually have. In reality, the deviation is smoothed out. It is noticeable that the y-deviation does not approach the zero point but oscillates at about 5 pixels of deviation. Depending on how it is positioned, the induced deviation from one of the axis due to misalignment of the mount may be stronger. Since the length of the steering signal for both axes scales equally, one axis might not catch up. In this case it it the y-axis. However, this can be easily corrected for later use by resetting the reference point.
To check for any malfunctions that might only appear after long durations of time, the guider has been tested for up to one hour. The behaviour of the target guiding seems to coincide with the short duration tests. As can be seen in figure 5–2, the Moon is kept within a corridor of about ± 2.5 pixels in either direction. Additionally, the long duration test indicates that the guider does not drift off on one axis. Although oscillating around the target, the guider keeps the Moon in the desired area.

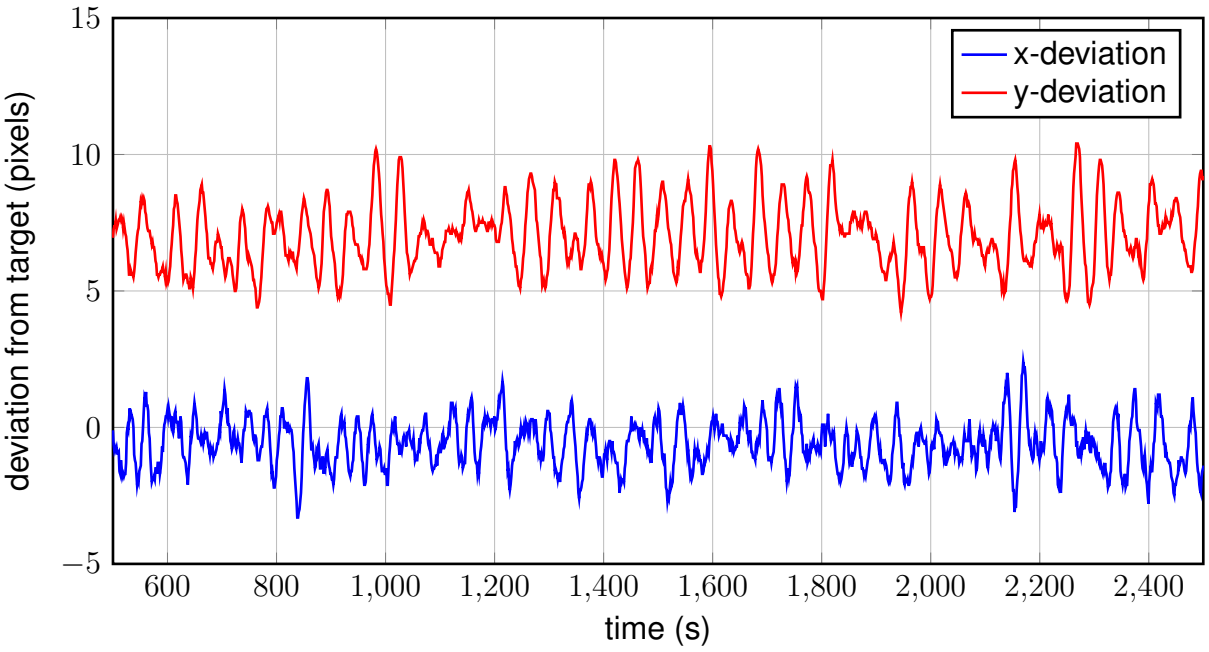


Fig. 5–2: Extract of a plot of a 50 minute test during half Moon in a clear night.

5.1 Discussion

Apparently the time delay between the steering signal and target deviation change in combination with the generally slow reaction of the ST-4 steering creates a feedback loop in the guider. This feedback loop than leads to oscillations around the desired location. It is possible that these oscillations can be overcome by designing the relay activation function in a different way or by adding a dampening component to it. However in the scope of this thesis, this simple approach was deemed sufficient. Because the response to a steering signal strongly depends on the model of the mount, complicated control loops might not generalize well to other models. In a project, which should be usable with many different hardware set-ups, simplicity is key.
The tests have shown that the guider performs well in a variety of different conditions. This includes various moon phases, as well as cloudy and varying lighting conditions. Since not every phase of the Moon could be tested in reality (in part due to the cloudy weather during the German winter), simulations had to be utilized. These simulations were used to find the appropriate parameters for the design of the system. In the conditions tested live so far, the guider was able to achieve equally good performance using these parameters. However, in systems that process data from the outside, there can always be discrepancies between the simulation environment and the chaotic environment of the real world. These discrepancies can be approximated as closely as possible within a simulation but can never be completely ruled out with absolute certainty. Because of this the performance of the guider will always be subject to unexpected environmental conditions to some degree.

6 Conclusion and Outlook

The system developed during this term paper is capable of guiding a standard motorized telescope mount to follow the movement of the Moon in the night sky for extended periods of time. A sufficient target holding precision to guide the mount for capturing impact flashed of the Moon has been demonstrated. The software is adaptable to various sky- and Moon-conditions as well as different hardware set-ups and can be configured and changed by the user. Extensive documentation on the software and hardware is provided to make the application more simple and user friendly. Furthermore the guiding system archived its goal of utilizing COTS-components as well as commercially available astronomy gear. When the system is adopted by more people, more data will be gathered about the performance of the Moon guider. Although any models of telescope mount equipped with a ST-4 interface should be able to use the guider in theory, it is yet to be tested and confirmed. As is often the case with open-source and community projects, they will continue to grow and adapt to new conclusions.

Further work on the Moon guider software could include a more advanced control loop for the steering corrections. To archive maximum guiding precision a predictive controller may be effective. Improvements of the user interface could also be made, to eliminate the need for a keyboard altogether.

Bibliography

[1] BerryBase, “Raspberry pi high quality kamera.”

[2] F. Momeni, H. Papei, R. Jamshidzadeh, S. Bereliani, A. Miri, A. Aghababaei, N. Nikjoo, E. Mehdizadehi, M. Akbari Gurabi, and N. Ghasabi Kondelaji, “Determination of the sun’s and the moon’s sizes and distances: Revisiting aristarchus’ method,” *American Journal of Physics*, vol. 85, no. 3, pp. 207–215, 2017.

[3] SBIG, “St-4 manual.”

[4] BerryBase, “3,5" display für raspberry pi mit resistivem touchscreen.”

[5] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler, “Comparative study of hough transform methods for circle finding,” *Image and Vision Computing*, vol. 8, no. 1, pp. 71–77, 1990.

[6] W. Rong, Z. Li, W. Zhang, and L. Sun, “An improved canny edge detection algorithm,” in *2014 IEEE International Conference on Mechatronics and Automation*. IEEE, 2014, pp. 577–582.

[7] V. K. Yadav, S. Batham, A. K. Acharya, and R. Paul, “Approach to accurate circle detection: Circular hough transform and local maxima concept,” in *2014 International Conference on Electronics and Communication Systems (ICECS)*. IEEE, 2014, pp. 1–5.

[8] R. A. Garfinkle, Ed., *Luna Cognita*. New York, NY: Springer New York, 2020.

A User Manual

The code was developed to guide a standard commercial telescope mount through a ST-4 port to follow the Moon using a Raspberry Pi (4B), a camera and a relay board. The system should work with any version of Raspberry Pi running Raspberry Pi OS as long as the other components are compatible. Due to computing demand a version 4 or higher is recommended.
The relay card can be any 4 channel relay card compatible with the Raspberry Pi. For this project a opto-isolator relay card was used but other relays will also work. Any camera compatible with the Raspberry Pi will work as long as the video stream can be called with Picam2. To archive a reasonable precision the sensor and lens should provide for a sufficient resolution of the Moon. At least 100 pixels per Moon diameter are recommended. The other components can be chosen freely.

For the original project, as shown in figure 1–1, these components were used:

- Raspberry Pi 4B
- Raspberry Pi High Quality Camera (C-Mount)
- *sertronics* 50mm Lens (C-Mount)
- *sertronics* RELM-4 4 Channel Relay Module
- *waveshare* 3.5 inch RPi LCD (A) Display
- *Innomaker* aluminium case
- Steel Button
- GPIO extension
- Dupont connector cables
- Standard RJ12 cable
- Standard camera screws

Fig. 1–1: Picture of the completed Moon guider assembly

A.1 Installation

On your Raspberry Pi running Raspberry Pi OS, do a update/upgrade and install the latest version of git (if not installed already):

```
sudo apt install git
```

Then get the MoonGuider repo from github:
```
git clone https://github.com/tobiasKurz1/MoonGuider.git
```

Navigate to the download folder and install use the requirements file to install the necessary libraries:
```
cd MoonGuider
pip3 install -r requirements.txt
```

Adjust the configuration parameters by editing the `config.ini`:
```
nano config.ini
```

The program can be run by executing the main file:
```
python3 main.py
```

A.2 Usage

The system is designed to be easy to use and self explanatory. Also the code is modular and comprehensive to allow for changes and adjustments to changing circumstances. Use the `config.ini` to configure the system with different parameters. The configuration file includes settings for the Moon phases full, half and new Moon but custom profiles can be added. The parameters are explained at the top of the file. The most important parameters for first launch include `pin_ra_down`, `pin_ra_up`, `pin_dec_down`, `pin_dec_up`, `pin_button` (depending on the wiring) as well as `image_height` and `image_width` (depending on the camera resolution). The pins may need to be adjusted based on the specific wiring. The numbers can be looked up in the Raspberry Pi documentation. When starting up and closing, a keyboard is required. While the program is running it can be controlled via the button only.
For additional functionality of the code `show_cam_feed`, `do_relay_test` and `export_to_excel` can be toggled on or off.

`show_cam_feed:` Opens a window displaying the current camera output at a high frame rate. This is used to initially align the camera and the mount while searching for the target. Can be terminated by pressing a key or the button.

`do_relay_test:` Can be used to test the correct wiring of the system. When activated a screen will show up, showing the current target detection. If the correct target (Moon) is detected and shown in on the screen, press the button. The system will then send five 2-second pulses to each relay, displaying the change in target position on the x- and y-Axis. This test should be performed with a realistic moving target (ideally the Moon), with the mount set to lunar velocity guiding. For a precise result the Mount should be aligned as precise as possible.

`export_to_excel:` Saves the logged activity of the Moon guider system. This includes target position, radius and deviation, the timing and durations of the steering signals as well as the configuration profile. The excel sheet can will be saved in the program folder under `/Logs`.

Running the Code

When the main file is executed, the `config.ini` is read. The user must then select a profile using the keyboard (if only one profile is present, it will be selected automatically). Next, the clicking of each relay should be heard. If there are fewer than four on and off clicks, the relay board is not working correctly. After that, the above-mentioned functions for camera setup and testing are executed if activated.
Before the guiding begins, the user can set a current Moon radius. When the prompt shows up, press the button to lock in the current moon radius and the press again to confirm. If the process is skipped by pressing a key, the restrictions on possible detected circles are set very loose.
The guider will now steer the mount to follow the moon within a small margin of error. On default the system will try to put the Moon in the center of the frame. If the button is pressed, the desired position will be set as the current position if the Moon. Hold the button for 2 seconds to reset it to the center. Hold the button for 5 seconds to exit the program.
Log files can be visualized using the included `plotter.py`, which outputs an image to the `/Logs` folder.

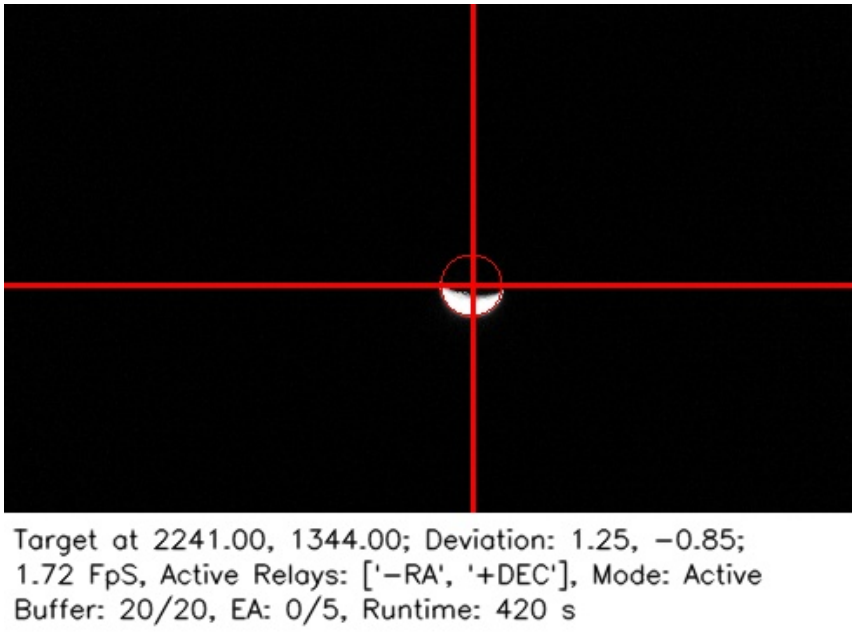


Fig. 1–2: Info screen of the Moon guider during operation

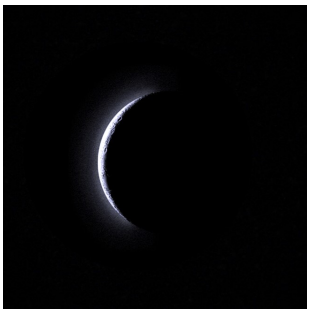
Figure 1–2 shows the display of the Moon guider during operation. The straight red lines show the reference position in the image, while the red circle marks the position of the Moon. Here the overlay is turned on, displaying information about the current

state. In the first line the target position and deviation from the reference point in pixels are displayed. Below the frames per second and the active relays are shown. Next to this is the current guiding mode. This switches between *Active*, *Inactive* and *Repeat*, depending on the `cloud_mode` setting and whether the target is locked or not. In the bottommost line, it is firstly displayed how many valid positions are stored in the buffer, from which the average value for the target is calculated. Here, you can track whether the Moon is detected in each iteration. The next value *EA* stands for `error_accumulator`. This internal variable counts consecutive frames in which the Moon is not detected. When the EA is full (set to 5 here), the guider switches to cloud-mode. Finally, in the bottom right corner, the elapsed time in seconds is indicated.

Preconfigured Moon Profiles

As a reference these parameters are recommendations for various guiding conditions:

Tab. 1–1: Recommended parameters for the Moon guider, depending on the conditions

Parameter	New Moon	Half Moon	Full Moon
Reference picture			
`blur`	6	5	4
`dp`	2	2	2
`param1`	300	300	200
`param2`	10	50	100

A.3 Troubleshooting

Custom Hardware
Hardware other the parts listed above can be used for this system. Here are some configuration changes and considerations which need to be made when running the code with a change in the respective component:

Computing Unit: A lower computing power may lead to very slow response times and low frame rate. To compensate for this, parameters like `image_buffer` and `in_scale` can be set to lower values. It may also help to raise `dp`. However all of there changes may impact guiding precision.

Switching Unit: In the original system the inactive relay position is `GPIO.HIGH`. If this is not the case the `HIGH` and `LOW` may need to be switched in the code inside `relay_handling.py`. In the case of standard three way relays the order of the wiring can also be changed to accomplish the same result without changing the code. Pins used for the connection of the relay board need to be specified in `config.ini`.

Camera: Adjust image width and height in `config.ini`. Change in resolution results in changed computing power requirements. Adjust these parameters to fit for your system (see above). Also `margin` and `pulse_multiplier` may be adjusted since they are dependent on the deviation in pixels. Depending on how the camera is oriented on the telescope Mount, adjust `rotate` in the configurations.

Lens: Different lenses lead to changes in Moon size on the sensor. Therefore `margin` and `pulse_multiplier` may need to be adjusted.

Display: Different screen sizes are usually not a problem as long as they are seamlessly integrated into the OS. Displays which require python libraries to be operated will need to be custom coded in `calc.py`. To improve image quality on higher resolution displays increase `out_scale`.

Mount: For the development of this system the *Skywatcher Star Adventurer GTI* was used as a telescope mount. As different mounts may guide with different speeds the `pulse_multiplier` needs be adjusted accordingly.

Button: The system is designed to include a button, whose pin needs to be specified in `config.ini`. If no button is connected, the system can still be entirely operated by using the keyboard.

B Hardware

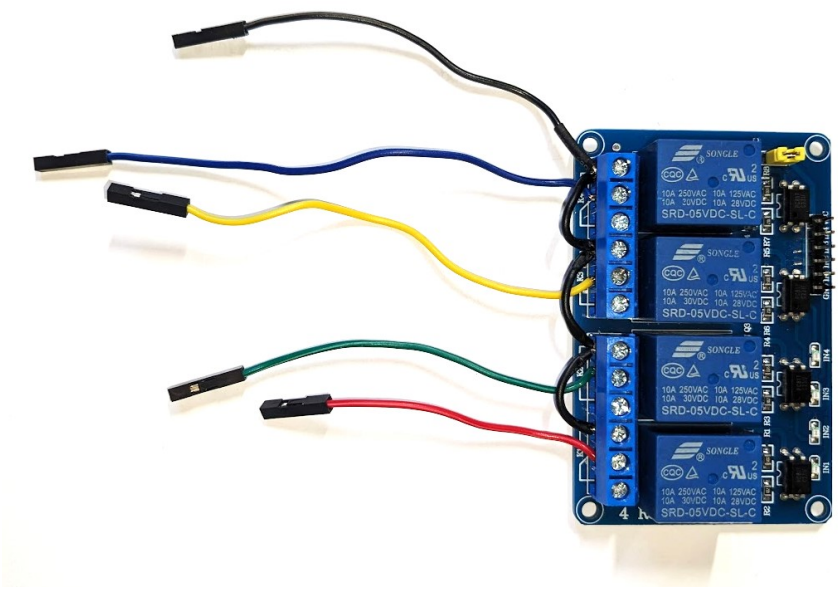


Fig. 2–1: Close-up image of the relay board including the wires for steering the mount. The colors are: black → ground; blue → -RA; yellow → -DEC; green → +DEC; red → +RA

Fig. 2–2: Picture of the assembled bottom plate with covered camera screw and spacer screws attached

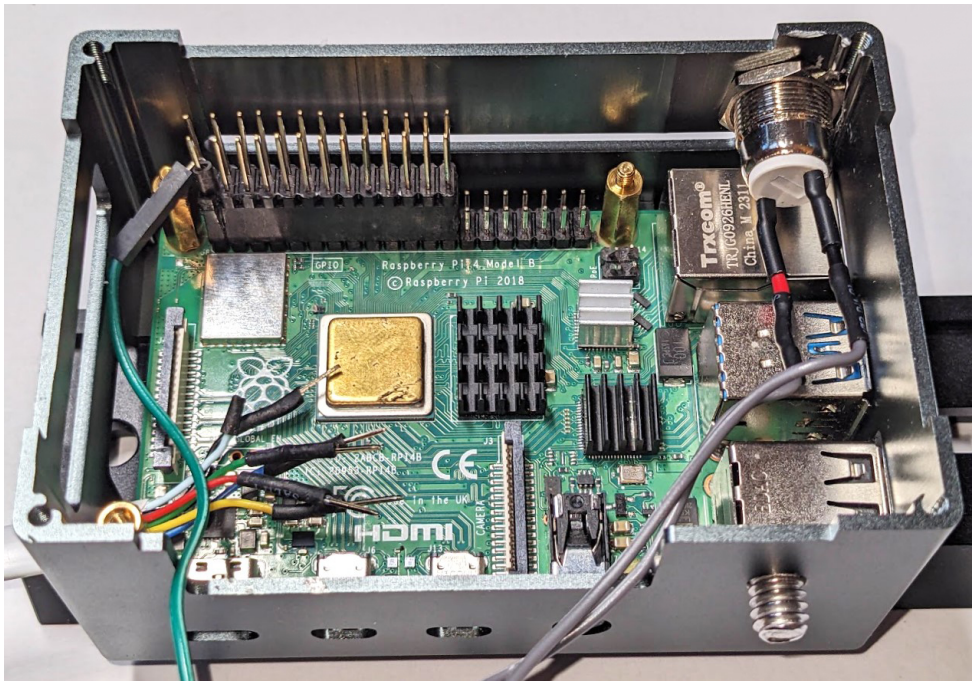


Fig. 2–3: Picture of the assembly after adding the button, GPIO extension, screw and RJ12 cable

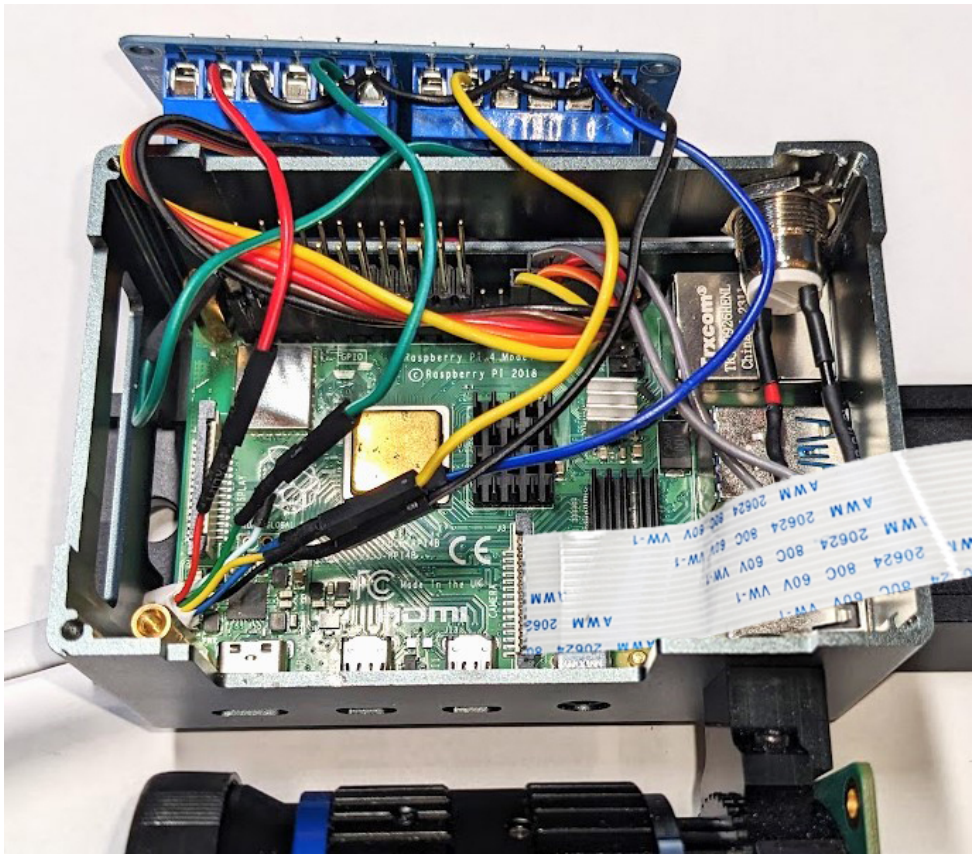


Fig. 2–4: Picture of the assembly after adding the relay board and the wire connections

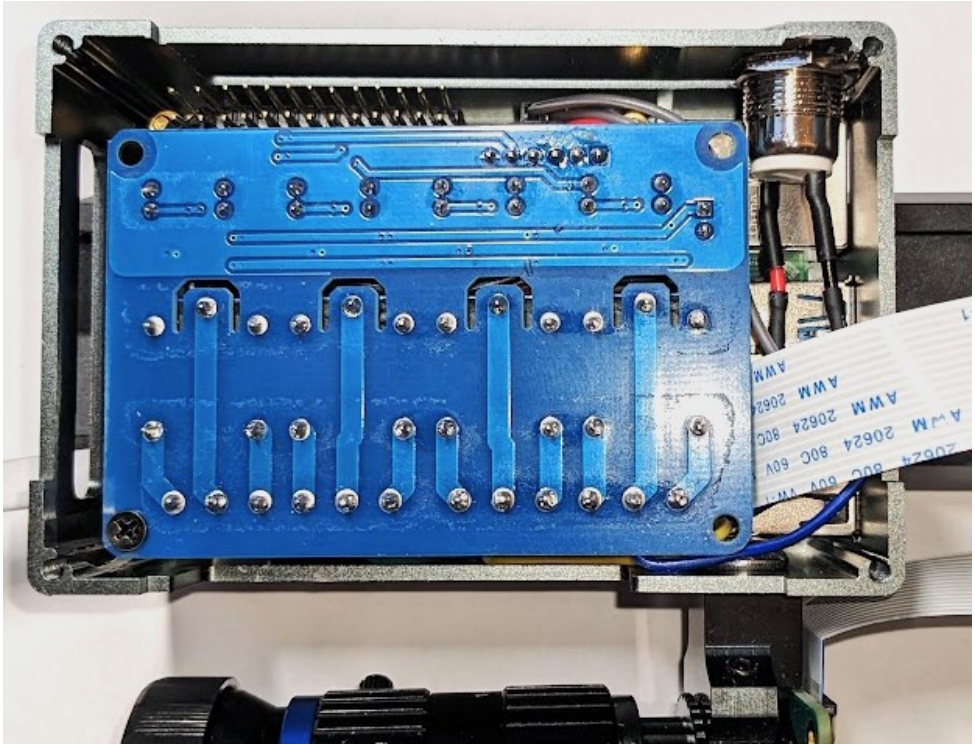


Fig. 2–5: Picture of the Moon guider with the relay board fully assembled

C Software Overview

C.1 Files

Tab. 3–1: Overview of the main files needed to run the Moon Guider with full potential (top) and auxiliary files, which were used for testing and developing (bottom)

File	Description
`main.py`	Main execution file to run the Moon Guider. Incorporates all of the functions directly or indirectly via the other files. Features options for setting up the guider, configuring and testing.
`calc.py`	Carries out the image processing, circle detection and deviation calculation. Handles the visual overlay as well as logging, buffering and exporting of values.
`relay_handling.py`	Initializes and handles the actuation of the relay board during normal operation as well as cloud obstruction, calculates the required signals
`config_loader.py`	Loads the `config.ini` file from memory and carries out the configuration profile handling as well as the distribution of the parameters.
`config.ini`	Main configuration file for the Moon Guider. Takes in multiple configuration profiles, which can be selected during start-up. Incorporates default profile as a fall-back option. Guidance and instruction on parameter selection can be found here.
`plotter.py`	Used to plot Moon Guider behaviour from Excel log-files.
`relay_test.py`	Gives options for relay activation. Is used to check connections and whether relay board is operating correctly.
`capture.py`	Used to capture and save arbitrary amount of high resolution images with the Raspberry Pi camera.
`main_testing.py`	Similar to main file but instead of taking camera images, existing images are used from memory. Was used to test initial main file and functions in predetermined environment.
`movement_test.py`	Activates directions in relay one by one while tracking and logging the target position. Was used to develop steering signal generator. For more details see section 3.6.
`relay_conti.py`	First version of signal output generator. Outputs a continuous signal instead of pulses and features a detection function for sticky relays. Function is discussed in section 3.6. Might be useful for some versions of mounts.
`gridsearch.py`	Performs grid search for `HoughCircles` parameters on folder of input images and outputs Excel file with results.

C.2 Classes

Tab. 3–2: Initializations for classes inside the Moon guider

Nr.	Class Name	File	Input
(1)	`configuration`	`config_loader.py`	`config.ini`
(2)	`calculation`	`calc.py`	`configuration`-class
(3)	`log`	`calc.py`	`configuration`-class
(4)	`buffer`	`calc.py`	`buffer_length`
(5)	`guide`	`relay_handling.py`	`configuration`-class; `log`-class

(1) Reads in the `config.ini` file from memory and checks for configuration profiles. If only one profile is found, it is chosen. Otherwise the user is prompted to choose the profile from a list. Then the contents of the profile are turned into the right data types and initialized as attributes of the `configuration`-class.
(2) Initializes attributes for later image processing from the configuration parameters.
(3) Creates a dictionary to store the data logs and fills in the configuration parameters from the `configuration`-class.
(4) Creates a dictionary to store the buffered values and initializes parameters from the configuration.
(5) Initializes attributes from the configuration parameters and creates locks and threads for the relay activation. If a rotation is set in the configuration, the order of the relay pins will be switched accordingly. Initializes the Raspberry Pi GPIO and then pulses every relay once.

C.3 Functions

Tab. 3–3: Functions for running the Moon guider

Nr.	Name	Input	Output
(1)	`perform_relay_test`	-	-
(2)	`lock_moon_size`	-	-
(3)	`setup`	`Picam2`-instance	-
(4)	`preprocessing`	`calculation`-class; image	image
(5)	`targetmarkers`	`calculation`-class; target position and radius; reference position, deviation, image, hand-over string	image

Continued on next page

Table 3–3 continued from previous page

Nr.	Name	Input	Output
(6)	`moonposition`	`calculation`-class; image	target position and radius
(7)	`get_deviation`	`calculation`-class; target position; reference position	deviation
(8)	`add`	`log`-class; sheet name; data	-
(9)	`export`	`log`-class	excel file
(10)	`errorcheck`	`buffer`-class; key	-
(11)	`get_valid`	`buffer`-class; key	string
(12)	`add`	`buffer`-class; value; key	-
(13)	`average`	`buffer`-class; key	average of stored values
(14)	`clear_all`	`buffer`-class	-
(15)	`to`	`guide`-class; deviation	-
(16)	`cloud_handling`	`guide`-class	deviation
(17)	`record`	`guide`-class	-
(18)	`activate_ra`	`guide`-class	-
(19)	`activate_dec`	`guide`-class	-
(20)	`switch_pin_on`	`guide`-class; direction	-
(21)	`switch_pin_off`	`guide`-class; direction	-
(22)	`button_is_pressed`	`guide`-class	boolean
(23)	`pulse`	`guide`-class; pin; count; up-time; down-time	-
(24)	`showactive`	`guide`-class	active pins and guide mode
(25)	`stop`	`guide`-class	-

(1) Outputs the image of the camera to the display, marking a detected target. The user then needs to confirm that the correct target is found by pressing the button. The function not iterates over the relay pins, activating them in pulses. After each pin the target location is checked again and the deviation is printed to the console. It is useful to identify, weather the wires to the relays are connected correctly. Ideally it should be performed on a stationary target, with the mount's sidereal tracking turned off. When the mount is aligned precisely it can also be performed on the live Moon. However any misalignment may skew the results.
(2) Outputs the image of the camera to the display, marking a detected target. The

user can confirm the detected Moon size by using the button. When it is pressed the user has to confirm the selection within 5 seconds or else the selection will be aborted. The Moon maximum and minimum radii will then be set to the selected size ± 5 pixels.
(3) Current camera images are streamed to the display in high refresh rate mode. This can be helpful for setting up the guider and finding the Moon. Can be terminated by pressing a key or the button.
(4) Applies grey scale filter and blur to the image.
(5) Adds all the informational overlays to the camera images and resizes the output if specified. This includes: target markings, reference point markings, deviation arrow, white information bar overlay. Additionally to the target location and deviation, the information bar outputs any text that is specified in the function input as the `handover_value`. The information bar automatically jumps to the top of the image if it would obstruct the view of the target.
(6) Applies the `HoughCirlces` circle detection to the image. If more that one circle is detected, the occurance is printed to the console and the largest circle is chosen.
(7) Computes the deviation between two points per axis in pixels.
(8) Adds data to the specified sheet in the logs dictionary.
(9) Creates a log file in `.xlsx`-format and writes all the gathered data to it. The user is prompted to name the file or can use the default naming convention by pressing enter. Entering "no" or likewise will skip the saving of the file. After naming the user can enter a note, which will be added to the "Configuration" sheet of the file. The file will be saved in the `/Logs` folder in the main directory.
(10) Checks if a correct buffer name is referenced.
(11) Counts the number of stored values in a given buffer. Returns a string in the format "[number of values] / [size of buffer]".
(12) Adds a value to a given buffer. If the buffer size is exhausted, the oldest item will be removed.
(13) Returns the average of all valid data points in the buffer.
(14) Clears the value buffers.
(15) Main function for guiding the mount, given a deviation. Will check if the target is found and initiate `cloud_handling` if necessary. Starts threads for steering the two axis, if they are not already running. Records valid deviations for repeating in cloud mode.
(16) Checks, which cloud mode is active and returns the corresponding substitute deviation. If the mode is `None`, the deviation will be set to 0, deactivating the relays. On `repeat` mode the function will iterate over the recorded deviations by setting the internal deviation value to the corresponding record. The other functions will then interpret the deviation as normal, making it repeat the exact activation pattern.
(17) Stores the deviations in a list and pops the oldest item if the length of `record_buffer` is reached.
(18) and (19) Main functions for interpreting the deviation as a correction signal for the respective axes. Checks for deviation within margin and calculates signal length based on the deviation and `pulse_multiplier`. The functions for switching the pins on and off will be called. Activity is communicated through the `active` attribute and logged with a time-stamp. Will always be executed as threads.

(20) Activates pins based on a list of booleans. The directions are mapped to the order of the booleans as follows: [Right, Left, Down, Up] or, with the original set-up in the northern hemisphere [-RA, +RA, -DEC, +DEC].
(21) Same as `switch_pin_on` but deactivates pins.
(22) Returns `True` if the button is pressed.
(23) Pulses a specified relay pin for a given amount of times.
(24) Returns a list of strings of the activated relays and the current guiding mode. Is used for the information bar.
(25) Joins the relay activation threads, turns off any remaining relays and releases the Raspberry Pi GPIO.

C.4 Test Results

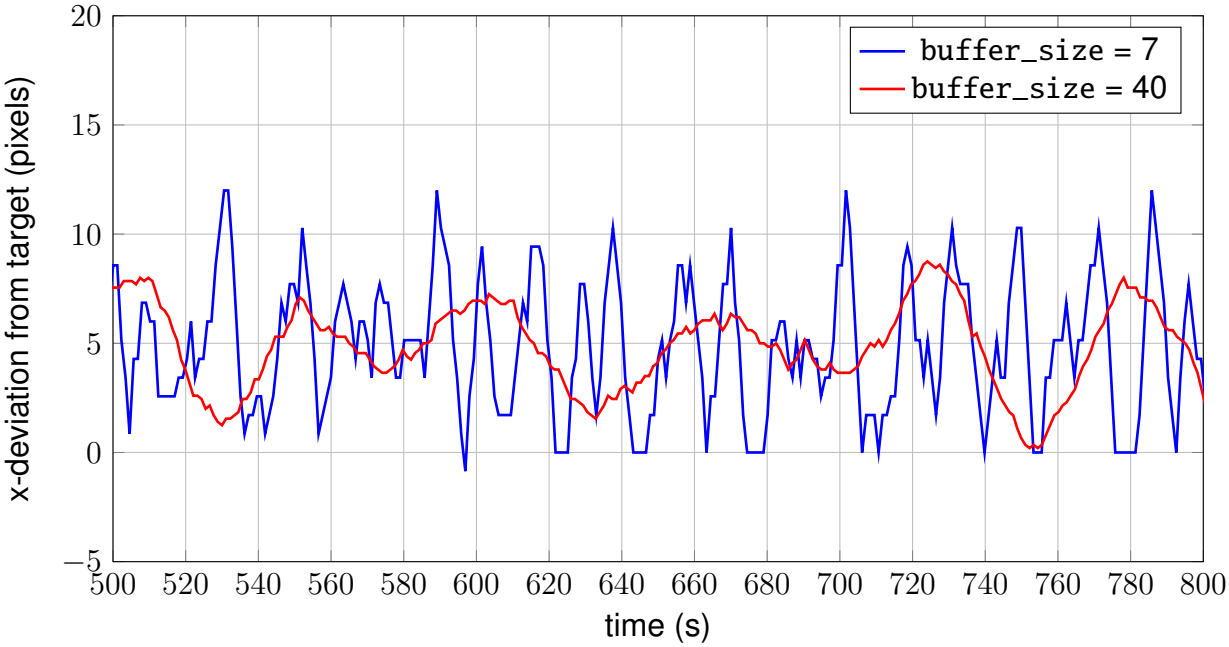


Fig. 3–1: Plot of two tests with different buffer sizes but otherwise the same settings

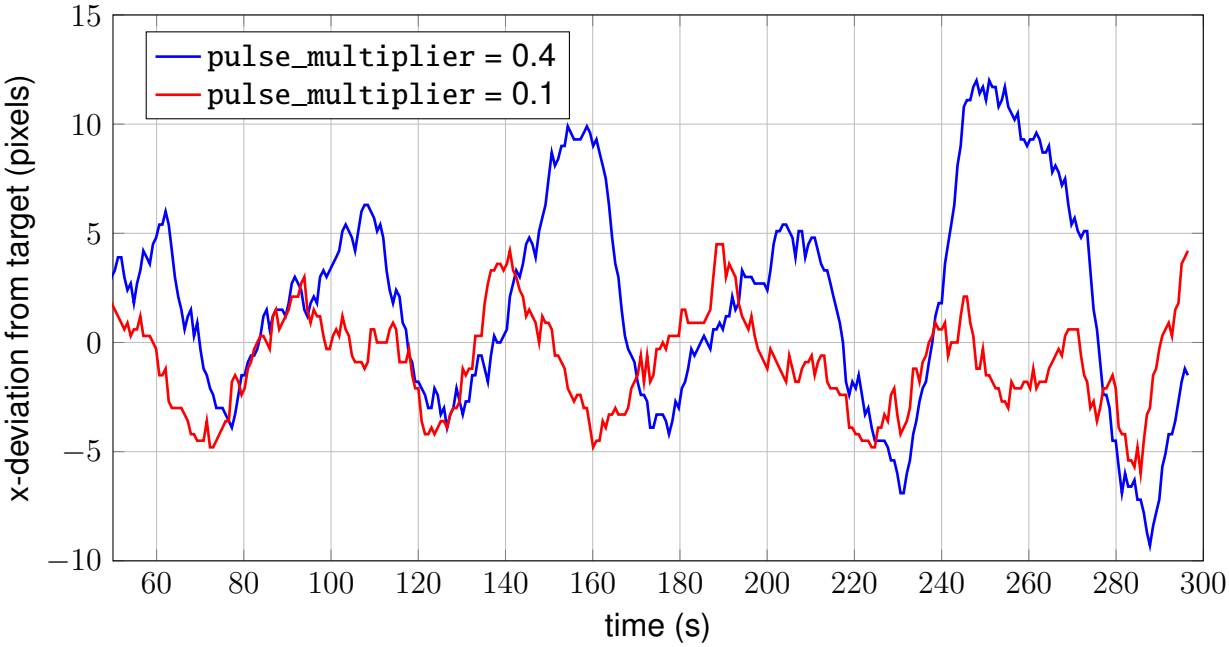


Fig. 3–2: Plot of two tests with different pulse multipliers but otherwise the same settings