Instrument for Obtaining Positional Information and Telescope Calibration

Thesis by
Kayleigh A. Childress

In Partial Completion of the Requirements for the
BS Astronomy-Physics Major

Under the Supervision of
Professor Bradley Johnson

UNIVERSITY OF VIRGINIA
Charlottesville, Virginia

2024

ABSTRACT

This instrument uses an Inertial Measurement Unit (IMU) and a GPS to collect information on the user's or telescope's physical position and velocity. It then captures an image of the sky and uses plate solving techniques to perform astrometry, thus providing the user with sky position. Combined, these key data points allow the instrument to be used in determining the orientation or alignment of a telescope regardless of whether or not the telescope has an optical component.

Keywords: telescope instrument, inertial measurement unit, astrometry

ACKNOWLEDGEMENTS

I would like to thank Professor Brad Johnson for his continuous support throughout this project, Senior Machinist Peter Dow for his help in the construction of the instrument, and the members of the Johnson Research Group for their insights. I would also like to thank my family, friends, and Trolli Sour Gummy Worms for helping me through many late nights.

Chapter 1

MOTIVATION

This instrument was developed to benefit research in observational astronomy at the University of Virginia. The purpose of this instrument is to provide the user with real-time positional information that can be used to calibrate or monitor an attached telescope. It was designed with no specific telescope in mind, and therefore can be utilized alongside most telescopes.By combining preexisting hardware and software packages, a low cost and compact alternative to industry standard instruments was able to be created.

1.1 Applications

Affectionately referred to as the SkyPi device, this instrument has multiple applications within Astronomy and at the University of Virginia. Serving as the instrument's computer, a Raspberry Pi provides a graphical interface for the user to interact with and real-time data output. The on-board Inertial Measurement Unit (IMU) and GPS collect the location and position of the instrument, while the attached planetary camera captures an image of the sky in FITS format. Astrometry is performed on the image, providing the celestial coordinates at which the instrument is pointed. This data is then output to the user via the terminal and also stored within the FITS header for later reference.

As stated above, the instrument was intentionally designed without considering a specific telescope or mount. It can be used as a hand held or free standing planetary camera and plate solver. Another application is mounted onto an optical telescope, serving as a pointing and calibration device similar to the Celestron StarSense camera. Lastly, this instrument can serve as an optical component for non-optical telescopes. This application may be of most use as the instrument can provide information to radio telescope users about the position of the telescope. Such information can be used to determine whether or not the telescope is pointed at the appropriate portion of the sky and allow users to make positional adjustments as needed.

Chapter 2

METHOD

This section outlines the materials and steps required to construct and operate the instrument.

2.1 Materials

Software

Table 2.1 contains a list of the software packages used in the operation of this instrument. The list contains the name of the package, doubling as a link to its documentation, and a brief description of its use. In addition to the software packages listed, numerous Python libraries were used in scripting for the instrument and data analysis.

Table 2.1: Software List

Software Package	Description
Ubuntu 22.04	Operating System
gpsd	GPS Service Daemon
Astrometry.net	Astrometry and Plate Solving
KStars	Digital Planetarium and Astrophotography
Ozzmaker	IMU Python Package

Hardware

Table 2.2 contains a list of hardware utilized in the construction of this instrument. General construction materials do not include a part number, but do contain relevant information in their descriptions. Specific materials that do not have a part number have links to where the part can be purchased instead as part numbers were not obtainable.

Table 2.2: Hardware List

Description	Part Number	Quantity
12" x 4"x 4" 1/4" Rectangluar Aluminum Tube	N/A	1
8" x 12" 1/16" Aluminum Sheet	N/A	1
4" x 4" 1/8" Aluminum Sheet	N/A	3
3" x 3" Aluminum L-Bracket	N/A	1
ZWO Planetary Camera	ZWO ASI120MC-S	1
Fujinon 75mm TV Lens	HF75HA-1B	1
Raspberry Pi 4B	SC0195(9)	1
Argon Fan HAT	N/A	1
SunFounder 5V Power Supply	CN0409D	1
BerryGPS-IMU V4	N/A	1
External GPS Antenna	N/A	1
UFL to SMA Adapter	N/A	1
50mm Gorilla Glass Window	22-855	1
50mm Diameter O-Ring	N/A	1
90 Degree USB A-B Adapter	N/A	1
Micro HDMI-HDMI Adapter	1358	1
Panel Mount HDMI Cable	978	1
Panel Mount Ethernet Cable	909	1
Panel Mount Female USB Cable	908	2
Panel Mount USB-C Cable	4218	1
M2.5x11+6mm Spacer	N/A	4
M2.5x5m+5mm Spacer	N/A	8
M2.5x5mm Screw	N/A	8
1/4-20 Screw	N/A	1
10-24 Screw	N/A	4
4-40 Screw	N/A	15

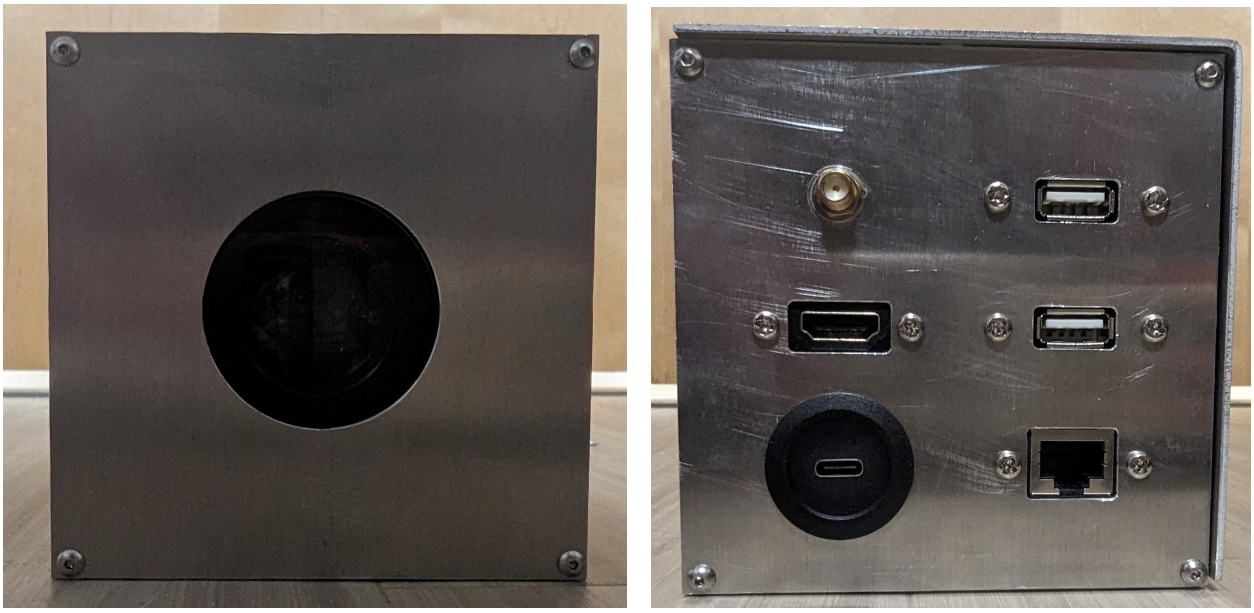
2.2 Instrument Construction

The enclosure was designed using Solidworks 3D CAD Modeling Software. In collaboration with Senior Machinist Peter Dow, the design was carefully refined to meet machining and material constraints. Once a 3D model was finalized, Peter Dow was able to machine the components and assemble the enclosure.

Figure 2.1: Enclosure with Dimensions

The enclosure for this device consists of a rectangular aluminum tube fitted with a removable lid and end caps. The approximate dimensions of the instrument are 12.375 inches in length, 4.0625 inches in width, and 4.0625 inches in height (See Fig. 2.1). For ease of access, a 10" x 3" x 3" section was removed from the center of the tubing. The lid was created from a 12" x 8" sheet of 1/16" thick aluminum bent at a 90 degree angle and is fastened to the device with 4-40 screws. The end caps were crafted from 4" x 4" sheets of 1/8" aluminum and made to fasten with 4-40 screws. The rear end cap has been machined to fit the panel mount connectors listed above in the parts list. The front has two end pieces that have been machined to encapsulate the glass window, held by the rubber o-ring (See Fig 2.3).

A custom aluminum L-bracket, designed to fit the ZWO Planetary Camera, is mounted on the inside of the enclosure with 10-24 screws. The camera, and attached 75mm lens, mount to the L-bracket using the standard 1/4"-20 screw used on most tripods and camera mounts. Mounting holes for the Raspberry Pi and BerryIMU-GPS were tapped through the bottom of the enclosure to ensure that the internal components are unable to move freely.



(a) Front with Window (b) Rear with Connectors

Figure 2.2: Enclosure End Caps

Using the standard size spacers and screws (listed above) for the Raspberry Pi, the Raspberry Pi, Argon Fan HAT, and BerryIMU-GPS can be mounted inside of the enclosure. For best practice, the Fan HAT is mounted directly on top of the Raspberry Pi using the GPIO pins and the IMU is mounted directly to the enclosure instead of the Raspberry Pi.This allows for the IMU to get the most accurate measurements of the device's position.The IMU is then connected to the Raspberry Pi GPIO pins using jumper wires.Additionally, there is space within the enclosure to include a battery pack to act as the 5V power supply. It is not necessary to include the battery pack, but it allows for the instrument to be used without an external power supply. The battery pack can also serve as a backup power source if needed.

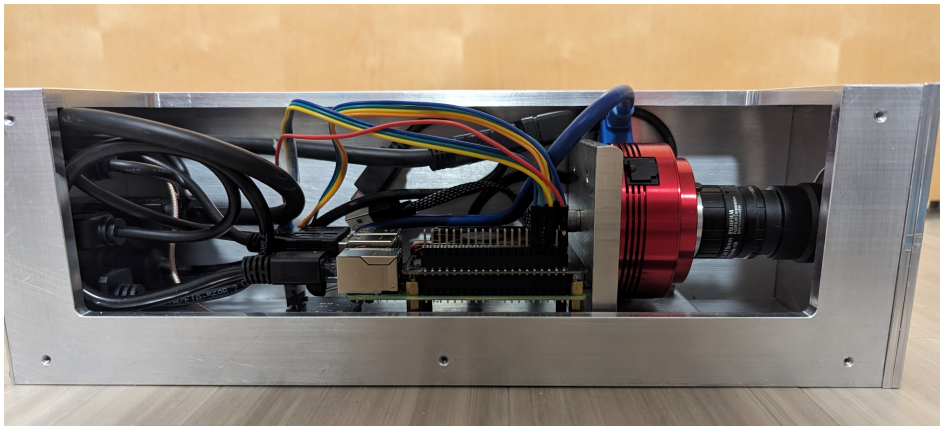


Figure 2.3: Internal Setup (Side View)

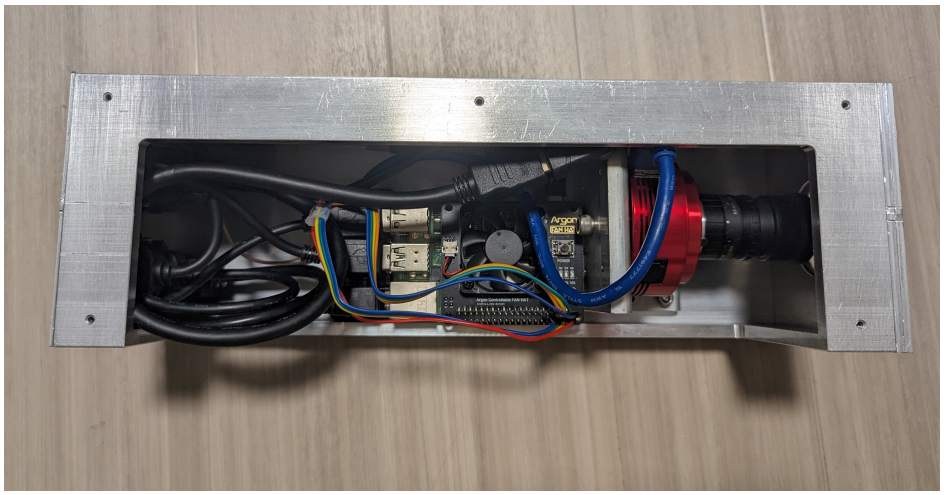


Figure 2.4: Internal Setup (Top View)

The construction of this instrument can be adjusted to fit a wide variety of applications. Some suggestions for adjustment include weather-proofing the instrument and adding mounting hardware. By using replacing the panel mount connectors with weatherproof alternatives and fitting the enclosure with rubber gaskets, the instrument may be left outdoors for an extended time period. Adding a 1/4" threaded hole to the bottom of the enclosure would allow for the instrument to be mounted to a standard tripod. Other types of mounting hardware may be attached as long as it does not interfere with the placement of the IMU or Raspberry Pi. It is likely that a weatherproof and tripod mountable variant of this instrument will be made in the future.

2.3 Instructions

These instructions are based on the assumption that the user has constructed the enclosure, but has not yet installed any of the software. Instructions on how to use the completed instrument fall under the "Instrument Use" section.

Instrument Setup

The first step is to install the operating system onto the Raspberry Pi. Ubuntu 22.04 [11] was used for this instrument and is most easily installed by flashing the operating system onto an SD card using the Raspberry Pi Imager. A more thorough instruction manual can be found here: https://ubuntu.com/tutorials/how-to-install-ubuntu-desktop-on-raspberry-pi-4#1-overview. Once completed, the Rasberry Pi can be booted up. Like with any new computer, a username, password, and other preferences will need to be selected. After the initial setup process is complete, the rest of the necessary software can be installed.

In order to access the GPS information, serial port communication must first be enabled on

the Raspberry Pi[10]. By default, the console uses communicates over the same serial port the GPS uses. To disable this, open the file 'cmdline.txt' and remove "console = tty1" and "console = ttyAMA0".Also, check that "cmdline=cmdline.txt" is present near the top of the file. Next, open the file 'config.txt' and add "enable_uart=1" at the bottom. With 'config.txt' open, check to see that "dtparam=i2c_arm=on" and "dtparam=spi=on" are listed[7]. These parameters allow for data from the IMU to be read as well. These files can be found under /boot/firmware and edited either within a text editor or using the nano command within the terminal.

Begin by updating the Raspberry Pi using "sudo apt-get install update". After the update the GPS daemon can be installed using "sudo apt-get install gpsd-clients gpsd -y". After installation, use "sudo nano /etc/default/gpsd" and edit the file so that it says "DEVICES = "/dev/serial0" so that gpsd reads the data from serial port 0[10, 7]. Similarly, KStars[5] can be installed using the command "sudo apt-add-repository ppa:mutlaqja/ppa" followed by "sudo apt-get install indi-full kstars-bleeding"[5]. To make kstars compatible with the ZWO camera, the proper driver needs to be installed using "sudo add-apt-repository ppa:mutlaqja/ppa" and then "sudo apt-get install indi-asi"[9]. When KStars is ran for the first time, create a profile and choose "ZWO CCD" and "Telescope Simulator" as the camera and telescope respectively[5]. The Astrometry.net[8] software can be installed using:

"sudo apt install build-essential curl git file pkg-config swig libcairo2-dev libnetpbm10-dev netpbm libpng-dev libjpeg-dev zlib1g-dev libbz2-dev libcfitsio-dev wcslib-dev python3 python3-pip python3-dev python3-numpy python3-scipy python3-pil"

After installation, the index files need to be downloaded into "/usr/local/astrometry/data". The index files can be found here: https://astrometry.net/doc/readme.html#getting-index-files[8]. Finally, the Ozzmaker Pyhton package can be downloaded from the github repository using "git clone http://github.com/ozzmaker/BerryIMU.git"[10]. Note: not all of these files are needed for the instrument to operate, but they provide useful examples that may be used to improve how the instrument operates in the future. In addition to the software packages listed, Python3 and the following packages need to be installed: numpy[4], matplotlib[6], scipy[12], and astropy[2, 1, 3]. Complete guides on software installation can be found using the URLs in the bibliography.

The Python script used to collect and record data is attached at the end of this document. Place the script as a .txt or .py file into a chosen directory on the Raspberry Pi. In the same directory, place the following scripts from the Ozzmaker repository: berryIMU.py, bmp388.py, CalibrateBerryIMU.py, IMU.py, LSM6DSL.py, and LIS3MDL.py.

Instrument Use

Once the software has been fully installed, the instrument is ready to be used. The first step is to enable the GPS daemon. This step is only needed if the GPS daemon does not start at boot. In the terminal, use these commands to enable and start the GPS daemon:

sudo systemctl enable gpsd.socket

sudo systemctl start gpsd.socket

sudo gpsd /dev/serial0 -F /var/run/gpsd.sock[10]

Next, open the KStars GUI and under "tools" select "ekos". Using the profile the coordinates with the ZWO camera described above, start the indi server and connect to the devices[5]. Place the instrument in the desired position and run the provided Python script. In the command line enter "python3 <filename> <number of captures> <exposure time>", replacing the place holder values with the name of the file, number of images to be captured, and the exposure time for each image. For each image captured, the data collected will be printed in the terminal and also written to the image's header file. Images will be captured every 5 seconds until all of the images have been captured. This time can be adjusted within the script if needed. A reminder of how to run the file is included within the script if needed.

After the images have been collected, the data can be read from the FITS header of each image. If desired, the script can be altered to record the information within a text file instead of or in addition to the FITS header.

Chapter 3

TESTING

This section describes how the instrument was tested and the results.

3.1 Procedure

Camera Testing

In order to test the capabilities of the camera, it was tested separately from the rest of the device. Since the camera would be used to identify portions of the sky, it needed to be able to focus "at infinity" and resolve an image. This, and the need to save images in the FITS format are largely why the ZWO Planetary Camera and Fujinon 75mm lens were chosen. On 3/20/24 at approximately 10:00pm EDT, the camera with the 75mm lens were mounted on a standard tripod and placed outside in the McCormick Road Residence Hall area of the University of Virginia. The lens aperture was set to f/2.8 with the focus fully extended. From this point on, references to the camera include the 75mm lens. The camera was then attached to a laptop computer via USB-A to USB-B cable and controlled using ZWO's ASIStudio Planetary Camera software. The camera was first pointed at the Moon to test its resolving capabilities and to gain familiarity with the camera software. After successfully resolving multiple images, the camera was then pointed at a random area of sky. This was done to prevent any bias in the results that could come from selecting a well known object or region in the sky. Appropriate exposure times were chosen by taking several test images at various exposure lengths until multiple objects could be resolved. Over the course of observing, approximately 20 images were taken. 10 exposure time test images, 5 images of the Moon, and 5 quality images of the random sky area. Due to some software complications and limited laptop battery life, more images could not be taken on this date. The 5 quality images of random sky were uploaded to the astrometry software to be plate solved. The identities of the plate solved objects were then confirmed using a digital sky map.

Complete Instrument Testing

After the complete instrument was assembled, it was tested on 4/18/24 at approximately 11:00pm EDT in the Night Lab area at the University of Virginia, located near the Astronomy building. The instrument was placed on one of the telescope observing piers where it was connected to an external power supply, display, and GPS antenna. The antenna was placed within 1ft of the instrument to allow for the most accurate GPS position to be measured. Additionally, to make the accelerometer and gyroscope readings valuable, the instrument was placed at an angle between 60

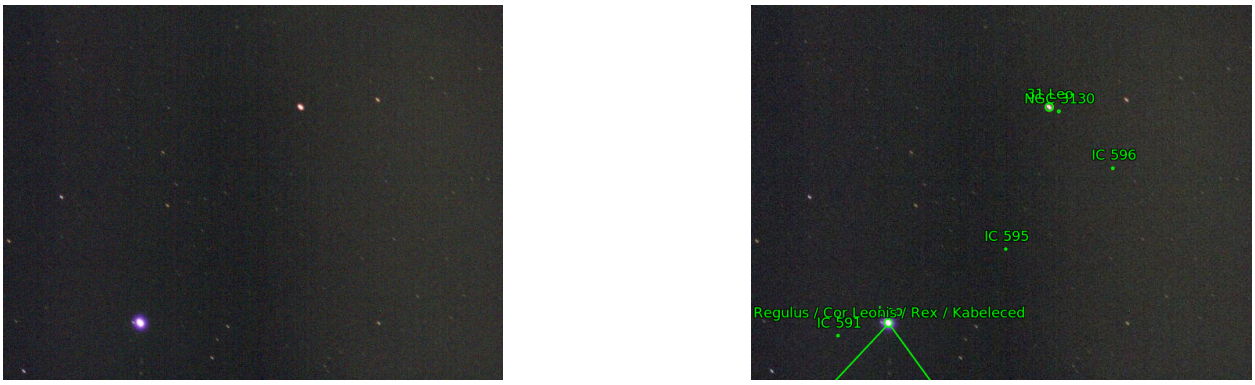
and 90 degrees atop the pier. After initializing the instrument as described in the setup process above, the main Python script for the instrument was ran. Due to some complications with the KStars software package, images were unable to be taken at this time. However, the instrument was still able to collect and record positional data from the IMU and GPS using a slightly modified version of the script below. Data was recorded once per second for 2 minutes and also once per minute for 10 minutes. A longer observing period was planned, but much of the observing time had been spent managing the software issues and modifying the Pyhton script.

Since the initial tests, the issues pertaining to the KStars software have been rectified, but a full scale test has not occurred.

3.2 Results and Analysis

Camera

The initial camera and plate solving test revealed that the instrument can successfully identify regions of the sky based on a captured image. The randomly selected region of the sky was identified to contain the star Regulus, binary system 31 Leonis, and galaxies IC 591, IC 595, IC 596, and NGC 3130 (Figure 3.1). These objects were confirmed through the digital sky map. The center of the image was estimated to have a Right Ascension (RA) of 152.113 degrees and a Declination (Dec) of 10.702 degrees.



(a) 5s exposure of random sky

(b) Annotated solved image

Figure 3.1: Annotated results of plate solving a captured image.

Using the pixel scale of the camera, 10.3 arcsec/pixel, and the known coordinates of IC 595, the estimated central image coordinates can be confirmed. Knowing that the central coordinates of each image are estimated to a reasonable accuracy, this information can be used to track or confirm the region of the sky at which a telescope is pointed. An example of this can be seen in Figure 3.2 below.

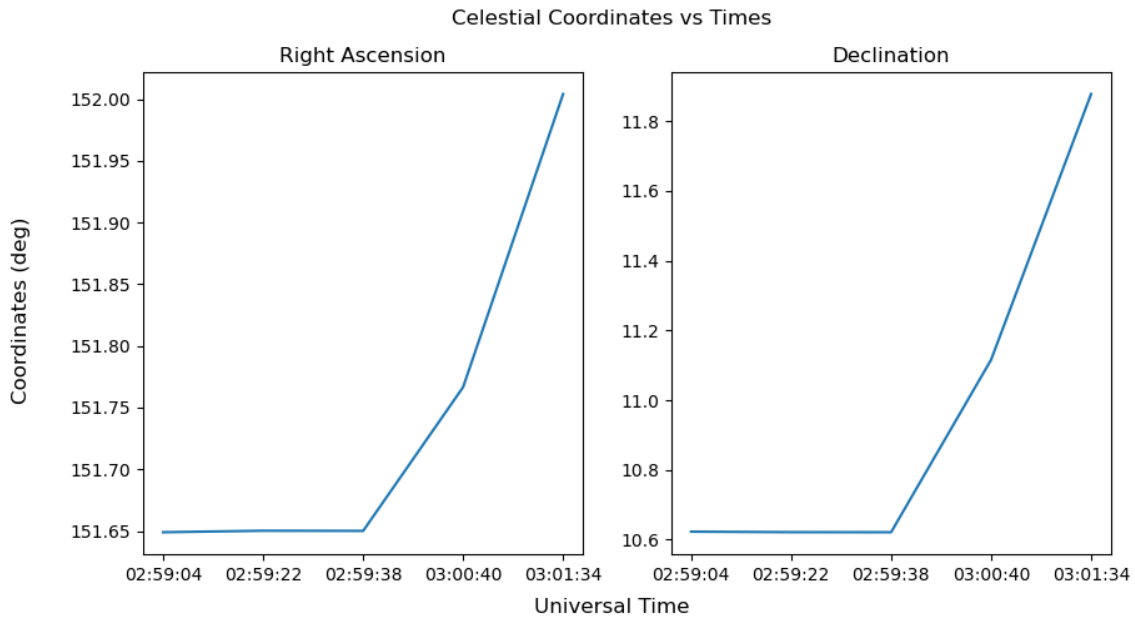
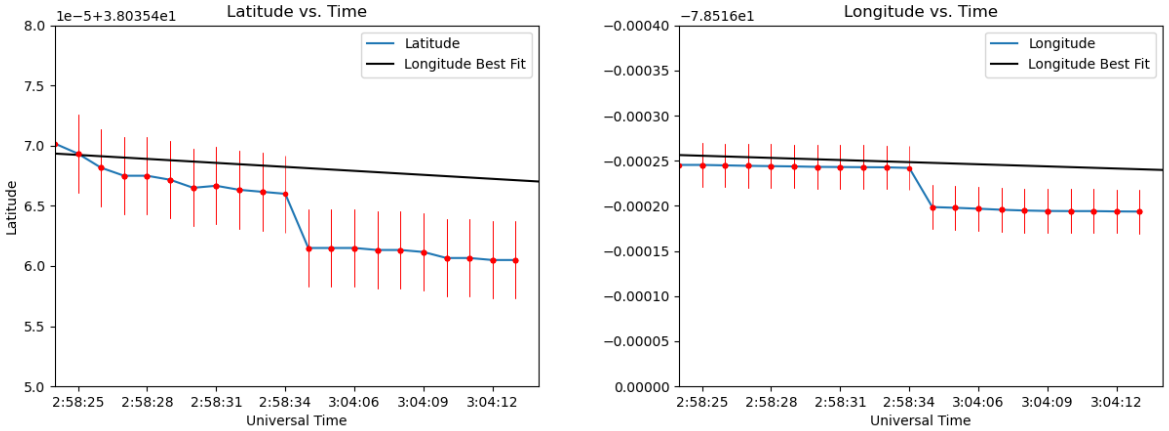


Figure 3.2: Central coordinates of plate solved images over the course of time.

Instrument

Positional data collected consists of the instrument's latitude and longitude, tilt in the X and Y directions, and velocity in the X, Y, and Z directions.Though the instrument is also capable of determining temperature, barometric pressure, and altitude. Of the 120 data points collected, 95 were considered to be usable. Any data point including a zero or NaN value was excluded as this indicates a failure to read information from the sensors. This test revealed that the instrument can record valuable information about its location and orientation over a period of time. Since the device was stationary during testing, the recorded values should not change significantly over time. Table 3.1 contains the full list of significant values obtained by the instrument.

Using Google Maps, which is assumed to be reasonably, the actual coordinates of where the instrument was placed were found to be 38.03548 degrees in latitude and -78.51619 degrees in longitude. The instrument found an average latitude and longitude of (38.03546, -78.51622) with a standard deviation of (3.2E-6, 2.4E-5). Though this places the measured latitude and longitude slightly outside of the standard deviation, it equates to a percent error of approximately 5E-5% in latitude and 4E-5% in longitude. A linear regression performed on the data revealed that the line of best fit for latitude to be y = -1E-7x + 38.03547 and longitude to be y = 8E-7x - 78.51626 (Fig. 3.3). The best fit lines are each nearly linear which aligns with expectations.This shows that the instrument can collect GPS data accurate enough to be utilized in telescope calibration. It is most likely that the discrepancy between the actual coordinates and measured coordinates has to do with the accuracy of the Google Maps coordinates.



(a) Latitude

(b) Longitude

Figure 3.3: Global Coordinates vs. Time

The X and Y directional tilt was measured by the accelerometer attached to the IMU. The X direction corresponds to the angle at which the instrument is tilted about its center, while the Y direction corresponds to the angle at which the instrument tilted relative to the ground. In the X direction, an average angle of 87.36439 degrees with a standard deviation of 0.15833 degrees was found. An angle of 90 degrees was expected as the x-axis of the accelerometer is rotated 90 degrees from the x-axis of the instrument. It is likely that the discrepancy between the measured and expected value is due to the instrument being placed on an uneven surface, causing a slight tilt. Linear regression revealed the best fit to be y = -2E-3x + 87.47045 (Fig. 3.5), which suits the semi-linear expectation.In the Y direction, an average angle of 82.33436 degrees with a standard deviation of 0.45658 degrees was found, which is within the estimated range of the instrument's position.The line of best fit was found to be y = -6E-3x + 82.62859 (Fig. 3.5). The linearity of the best fit lines and small standard deviations of the sample, show that the instrument can reliably record directional tilt of a telescope to within half a degree.

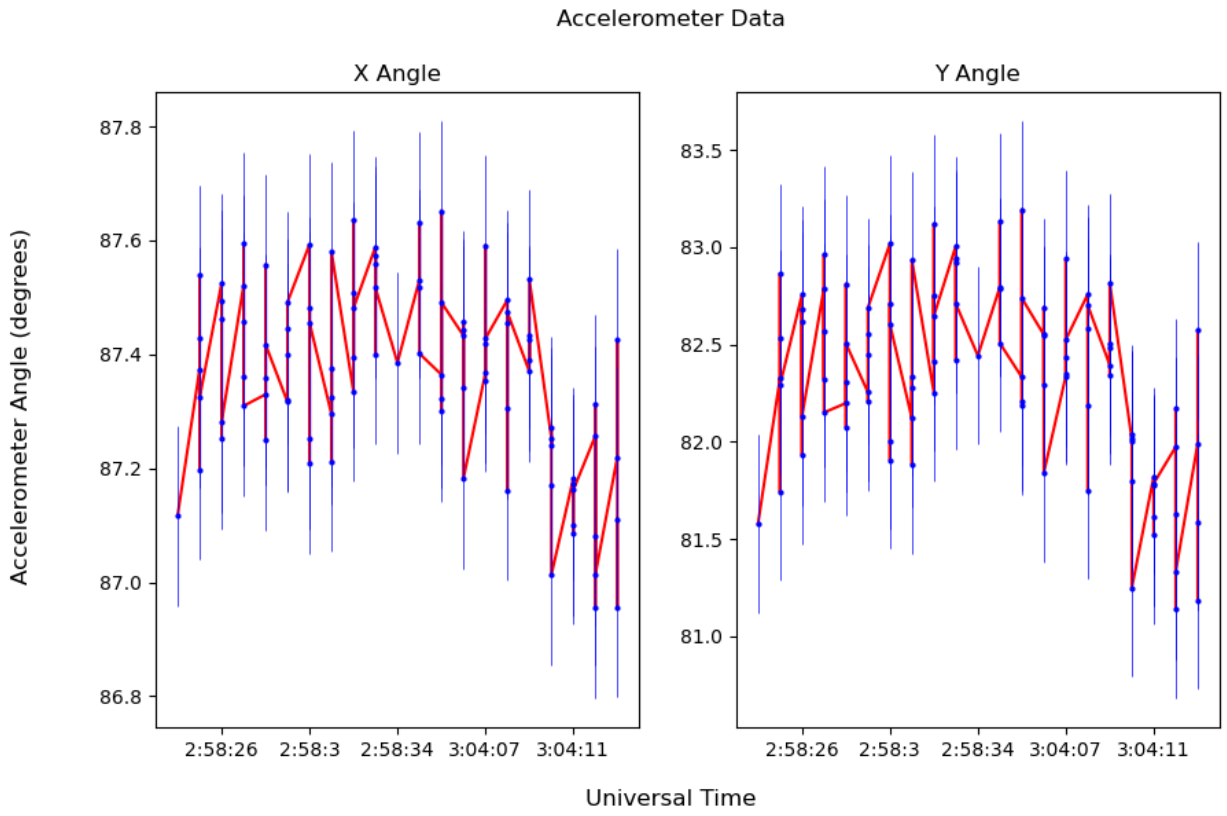


Figure 3.4: Instrument tilt including error measured by the accelerometer

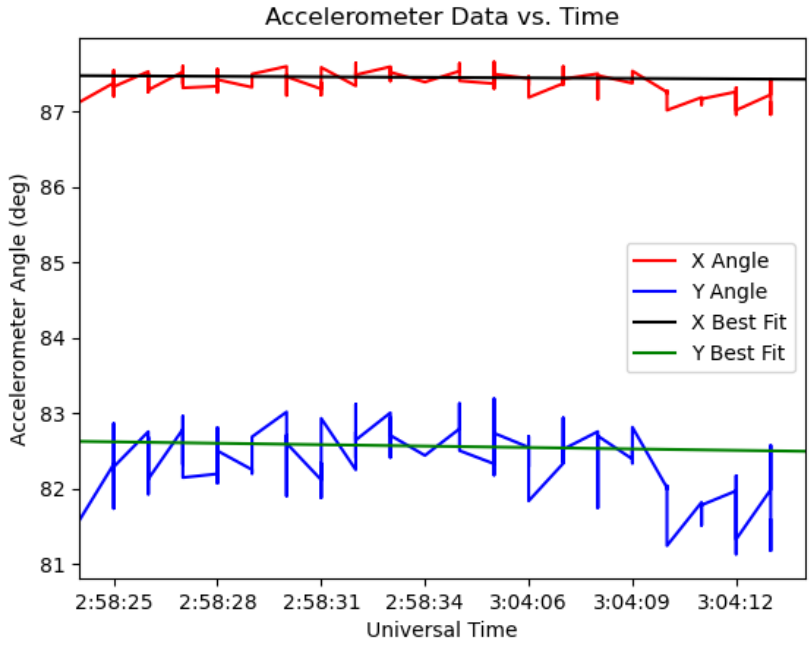


Figure 3.5: Instrument tilt including best fit measured by the accelerometer

The change in position of the instrument, measured in degrees per second, can be used to determine the rate at which a telescope is moving or if it is drifting away from its intended location. Since the instrument was stationary, a velocity of 0 degrees per second is to be expected in all directions. For the gyroscope, the X and Y directions correspond with those described above while the Z direction corresponds to a rotation about the center. In the X direction, the gyroscope recorded an average velocity of 4.34E-6 degrees per second with a standard deviation of 3E-6 degrees per second. The line of best fit is y = 6.68E-8x + 1.2E-6.In the Y direction, the gyroscope recorded an average velocity of 3.29E-5 degrees per second with a standard deviation of 1.7E-4 degrees per second. The line of best fit is y = 1.03E-7x + 2.8E-5. In the Z direction, the gyroscope recorded an average velocity of -1.93E-5 degrees per second with a standard deviation of 1.2E-4 degrees per second. The line of best fit is y = -9.2E-8x + 1.5E-5. Even though the gyroscope did not record a velocity of zero, the values recorded are negligible when considering that zero is within one to two standard deviations in each direction. Additionally, as seen in Figure 3.6, the line of best fit in each direction has a minimal slope and lies close to zero. This serves as evidence that the instrument can accurately measure a telescope's change in position over time.

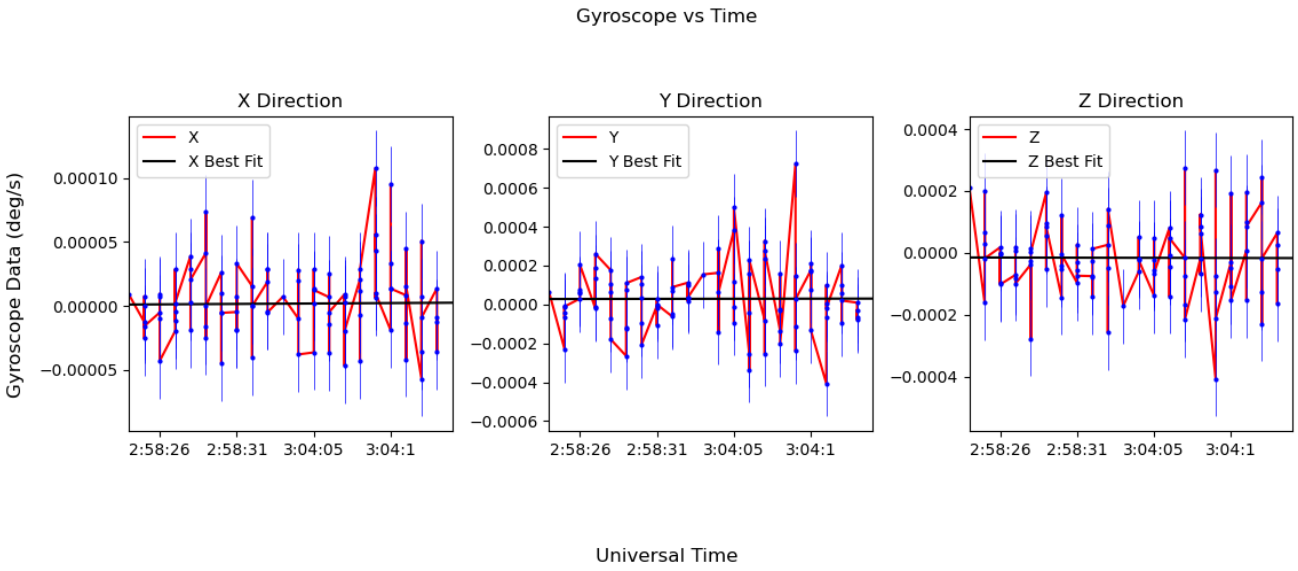


Figure 3.6: Instrument change in position recorded by the Gyroscope

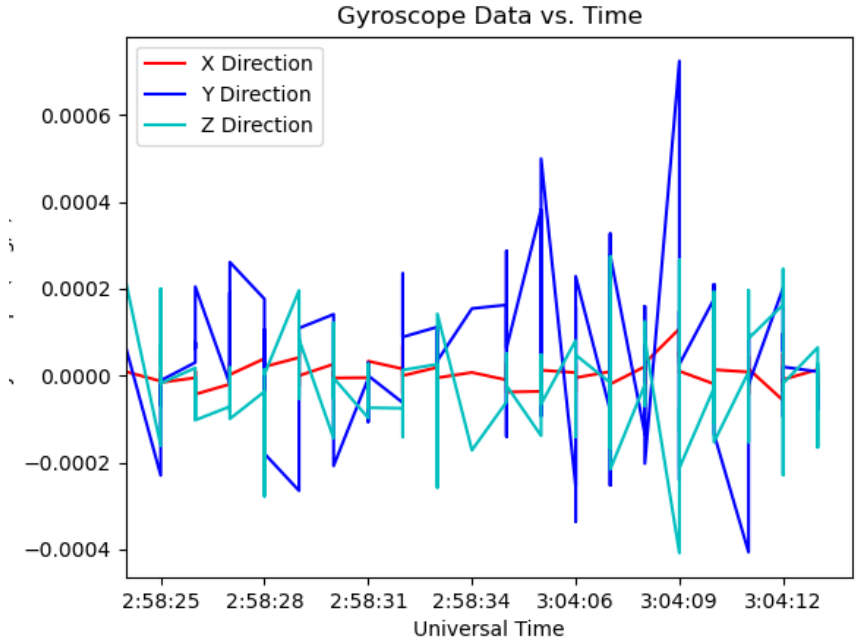


Figure 3.7: Deg/s vs. Time

Table 3.1: The Average, Minimum, Maximum, Standard Deviation, and Best Fit Parameters

Value	Average	Minimum	Maximum	Standard Deviation	Intercept	Slope
Latitude	38.03546	38.03546	38.03547	3.21050e-06	38.0354693	-1.1048e-07
Longitude	-78.516219	-78.51624	-78.51619	2.422e-05	-78.51625	7.8640e-07
X Tilt Angle	87.36349	86.954594	87.65155	0.15833	87.47044	-0.0022756
Y Tilt Angle	82.334364	81.13654	83.19285	0.45658	82.62859	-0.00626
X Velocity	4.3407e-06	-5.698e-05	0.000118	2.9362e-05	1.1989e-06	6.6848e-08
Y Velocity	3.2855e-05	-0.000406	0.0007238	0.000170	2.80164e-05	1.02949e-07
Z Velocity	-1.926e-05	-0.000408	0.000274	0.00012	1.4928e-05	-9.2167e-08

Chapter 4

CONCLUSION

Based on the results of the tests described, it can be concluded that the instrument successfully collects celestial coordinates, global coordinates, tilt angle, and velocity within a reasonable margin of error, such that it can be applied in observational astronomy research. This device is able to provide the user with specific physical orientation data and perform astrometry. Thus, can be used in the calibration of telescopes or monitoring of telescope position. This is particularly useful in radio astronomy where telescopes do not have optical components to assist with positioning, allowing the user to identify where the telescope is pointed and determine if its position needs to be modified. In the future, some minor adjustments will be required to prepare the instrument for full time use. Some areas of improvement include real-time plotting through a graphical interface, full instrument weatherproofing, and adding mounting hardware to the enclosure. However, it serves as proof that readily available materials can be used to construct low cost alternatives to industry standard instruments.

BIBLIOGRAPHY

[1] Astropy Collaboration, A. M. Price-Whelan, B. M. Sipőcz, H. M. Günther, P. L. Lim, S. M. Crawford, S. Conseil, D. L. Shupe, M. W. Craig, N. Dencheva, A. Ginsburg, J. T. Vand erPlas, L. D. Bradley, D. Pérez-Suárez, M. de Val-Borro, T. L. Aldcroft, K. L. Cruz, T. P. Robitaille, E. J. Tollerud, C. Ardelean, T. Babej, Y. P. Bach, M. Bachetti, A. V. Bakanov, S. P. Bamford, G. Barentsen, P. Barmby, A. Baumbach, K. L. Berry, F. Biscani, M. Boquien, K. A. Bostroem, L. G. Bouma, G. B. Brammer, E. M. Bray, H. Breytenbach, H. Buddelmeijer, D. J. Burke, G. Calderone, J. L. Cano Rodríguez, M. Cara, J. V. M. Cardoso, S. Cheedella, Y. Copin, L. Corrales, D. Crichton, D. D'Avella, C. Deil, É. Depagne, J. P. Dietrich, A. Donath, M. Droettboom, N. Earl, T. Erben, S. Fabbro, L. A. Ferreira, T. Finethy, R. T. Fox, L. H. Garrison, S. L. J. Gibbons, D. A. Goldstein, R. Gommers, J. P. Greco, P. Greenfield, A. M. Groener, F. Grollier, A. Hagen, P. Hirst, D. Homeier, A. J. Horton, G. Hosseinzadeh, L. Hu, J. S. Hunkeler, Ž. Ivezić, A. Jain, T. Jenness, G. Kanarek, S. Kendrew, N. S. Kern, W. E. Kerzendorf, A. Khvalko, J. King, D. Kirkby, A. M. Kulkarni, A. Kumar, A. Lee, D. Lenz, S. P. Littlefair, Z. Ma, D. M. Macleod, M. Mastropietro, C. McCully, S. Montagnac, B. M. Morris, M. Mueller, S. J. Mumford, D. Muna, N. A. Murphy, S. Nelson, G. H. Nguyen, J. P. Ninan, M. Nöthe, S. Ogaz, S. Oh, J. K. Parejko, N. Parley, S. Pascual, R. Patil, A. A. Patil, A. L. Plunkett, J. X. Prochaska, T. Rastogi, V. Reddy Janga, J. Sabater, P. Sakurikar, M. Seifert, L. E. Sherbert, H. Sherwood-Taylor, A. Y. Shih, J. Sick, M. T. Silbiger, S. Singanamalla, L. P. Singer, P. H. Sladen, K. A. Sooley, S. Sornarajah, O. Streicher, P. Teuben, S. W. Thomas, G. R. Tremblay, J. E. H. Turner, V. Terrón, M. H. van Kerkwijk, A. de la Vega, L. L. Watkins, B. A. Weaver, J. B. Whitmore, J. Woillez, V. Zabalza, and Astropy Contributors. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. , 156(3):123, September 2018.

[2] Astropy Collaboration, Adrian M. Price-Whelan, Pey Lian Lim, Nicholas Earl, Nathaniel Starkman, Larry Bradley, David L. Shupe, Aarya A. Patil, Lia Corrales, C. E. Brasseur, Maximilian N"othe, Axel Donath, Erik Tollerud, Brett M. Morris, Adam Ginsburg, Eero Vaher, Benjamin A. Weaver, James Tocknell, William Jamieson, Marten H. van Kerkwijk, Thomas P. Robitaille, Bruce Merry, Matteo Bachetti, H. Moritz G"unther, Thomas L. Aldcroft, Jaime A. Alvarado-Montes, Anne M. Archibald, Attila B'odi, Shreyas Bapat, Geert Barentsen, Juanjo Baz'an, Manish Biswas, M'ed'eric Boquien, D. J. Burke, Daria Cara, Mihai Cara, Kyle E. Conroy, Simon Conseil, Matthew W. Craig, Robert M. Cross, Kelle L. Cruz, Francesco D'Eugenio, Nadia Dencheva, Hadrien A. R. Devillepoix, J"org P. Dietrich, Arthur Davis Eigenbrot, Thomas Erben, Leonardo Ferreira, Daniel Foreman-Mackey, Ryan

Fox, Nabil Freij, Suyog Garg, Robel Geda, Lauren Glattly, Yash Gondhalekar, Karl D. Gordon, David Grant, Perry Greenfield, Austen M. Groener, Steve Guest, Sebastian Gurovich, Rasmus Handberg, Akeem Hart, Zac Hatfield-Dodds, Derek Homeier, Griffin Hosseinzadeh, Tim Jenness, Craig K. Jones, Prajwel Joseph, J. Bryce Kalmbach, Emir Karamehmetoglu, Mikolaj Kaluszy'nski, Michael S. P. Kelley, Nicholas Kern, Wolfgang E. Kerzendorf, Eric W. Koch, Shankar Kulumani, Antony Lee, Chun Ly, Zhiyuan Ma, Conor MacBride, Jakob M. Maljaars, Demitri Muna, N. A. Murphy, Henrik Norman, Richard O'Steen, Kyle A. Oman, Camilla Pacifici, Sergio Pascual, J. Pascual-Granado, Rohit R. Patil, Gabriel I. Perren, Timothy E. Pickering, Tanuj Rastogi, Benjamin R. Roulston, Daniel F. Ryan, Eli S. Rykoff, Jose Sabater, Parikshit Sakurikar, Jes'us Salgado, Aniket Sanghi, Nicholas Saunders, Volodymyr Savchenko, Ludwig Schwardt, Michael Seifert-Eckert, Albert Y. Shih, Anany Shrey Jain, Gyanendra Shukla, Jonathan Sick, Chris Simpson, Sudheesh Singanamalla, Leo P. Singer, Jaladh Singhal, Manodeep Sinha, Brigitta M. SipHocz, Lee R. Spitler, David Stansby, Ole Streicher, Jani Sumak, John D. Swinbank, Dan S. Taranu, Nikita Tewary, Grant R. Tremblay, Miguel de Val-Borro, Samuel J. Van Kooten, Zlatan Vasovi'c, Shresth Verma, Jos'e Vin'icius de Miranda Cardoso, Peter K. G. Williams, Tom J. Wilson, Benjamin Winkel, W. M. Wood-Vasey, Rui Xue, Peter Yoachim, Chen Zhang, Andrea Zonca, and Astropy Project Contributors. The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. , 935(2):167, August 2022.

[3] Astropy Collaboration, T. P. Robitaille, E. J. Tollerud, P. Greenfield, M. Droettboom, E. Bray, T. Aldcroft, M. Davis, A. Ginsburg, A. M. Price-Whelan, W. E. Kerzendorf, A. Conley, N. Crighton, K. Barbary, D. Muna, H. Ferguson, F. Grollier, M. M. Parikh, P. H. Nair, H. M. Unther, C. Deil, J. Woillez, S. Conseil, R. Kramer, J. E. H. Turner, L. Singer, R. Fox, B. A. Weaver, V. Zabalza, Z. I. Edwards, K. Azalee Bostroem, D. J. Burke, A. R. Casey, S. M. Crawford, N. Dencheva, J. Ely, T. Jenness, K. Labrie, P. L. Lim, F. Pierfederici, A. Pontzen, A. Ptak, B. Refsdal, M. Servillat, and O. Streicher. Astropy: A community Python package for astronomy. , 558:A33, October 2013.

[4] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[5] Jason Harris and the KStars Team. Download kstars. *https://kstars.kde.org/download/*, 2024.

[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[7] Shashank Kulkarni. A beginner's guide to using gpsd (gps devices) in linux. *https://kickstartembedded.com/2022/07/23/a-beginners-guide-to-using-gpsd-in-linux/*, 2022.

[8] Mierle K. Blanton M. Roweis S. Lang D., Hogg D. W. Blind astrometric calibration of arbitrary astronomical images. *Astronomical Journal*, 137, 2010.

[9] Jasem Mutlaq. Zwo asi cameras. *https://www.indilib.org/individuals/devices/cameras/zwo-optics-asi-cameras.html*, 2022.

[10] Ozzmaker. Berrygps setup guide for raspberry pi. *https://ozzmaker.com/berrygps-setup-guide-raspberry-pi/*.

[11] Cannonical Ubuntu. How to install ubuntu desktop on raspberry pi 4 | ubuntu. *https://ubuntu.com/tutorials/how-to-install-ubuntu-desktop-on-raspberry-pi-4*, 2024.

[12] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

```
#----README----------------------------------
#To run: Open Kstars GUI and then open Ekos, select skypi profile, start indi server
and connect devices
#then in the terminal: python3 skytestall3.2.txt <number of images> <exposure time>
#Created by: Kayleigh Childress
#Some code provided by OzzMaker, modified by K. Childress
#Some code found on the indilib forums from user dolguldur, modified by K. Childress
#It is recommended to run calibrateBerryIMU.py and input values into calibration
section below before use
#good luck
#Imports-----------------------------------------

import os
import numpy as np
import matplotlib.pyplot as plt
import sys
import time
import math
import IMU
import bmp388
import datetime
from gps import *
from astropy.io import fits
from astropy.wcs import WCS
from astropy.coordinates import SkyCoord
from astropy import units as u
from gi.repository import GObject
from gi.repository import GLib



#Global Variables--------------------------------
RAD_TO_DEG = 57.29578
M_PI = 3.14159265358979323846
G_GAIN = 0.070  # [deg/s/LSB]  If you change the dps for gyro, you need to update
this value accordingly
AA =  0.40      # Complementary filter constant
#timestamp  = datetime.datetime.now().strftime('%m_%d_%Y_%H_%M_%S') #use for
timestamped file names
gpsd = gps(mode=WATCH_ENABLE|WATCH_NEWSTYLE)
hdu = fits.PrimaryHDU()
#-------------------------------------------------
#Variable Lists-----------------------------------
RiAs=[]
Decl=[]
CnRiAs = []
CnDecl = []
UTime = []
GyX = []
GyY = []
```

```
GyZ = []
AcX = []
AcY = []
Glon = []
Glat = []
ALt = []
#-----------------------------------------------------

caps = sys.argv[1]
expo = int(sys.argv[2])
args = len(sys.argv)


#----command arguments check-----------
if args != 3:
        print("Please input number of images and exposure time \n")
        print("Format: python3 skytestall.py <number of images> <exposure time>")
        sys.exit()
if args == 3:

        print("Capturing " + sys.argv[1] + " Images.\n Do Not Move SkyPi Device.")
        print("Unless you are running tests that involve moving the device")

####################KStars DBUS Section#################
#Make sure kstars & Ekos are running
# Create a session bus.
from pydbus import SessionBus
bus = SessionBus()
# Create an object that will proxy for a particular remote object.
remote_object = bus.get("org.kde.kstars", # Connection name
                        "/KStars/INDI" # Object's path
                       )

# Get INDI interface
myDevices = [ "indi_simulator_telescope", "indi_asi_ccd" ]

# Start INDI devices
remote_object.start(7624, myDevices)

print("Waiting for INDI devices...")

# Create array for received devices
devices = []

while True:
    devices = remote_object.getDevices()
    if (len(devices) < len(myDevices)):
        time.sleep(1)
    else:
        break;
```

```
print("We received the following devices:")
for device in devices:
    print(device)

print("Establishing connection to Telescope and CCD...")
# Set connect switch to ON to connect the devices
remote_object.setSwitch("Telescope Simulator", "CONNECTION", "CONNECT", "On")
# Send the switch to INDI server so that it gets processed by the driver
remote_object.sendProperty("Telescope Simulator", "CONNECTION")
# Same thing for CCD
remote_object.setSwitch("ZWO CCD ASI120MC-S", "CONNECTION", "CONNECT", "On")
remote_object.sendProperty("ZWO CCD ASI120MC-S", "CONNECTION")

telescopeState = "Busy"
ccdState       = "Busy"

# Wait until devices are connected
while True:
    telescopeState = remote_object.getPropertyState("Telescope Simulator",
"CONNECTION")
    ccdState       = remote_object.getPropertyState("ZWO CCD ASI120MC-S",
"CONNECTION")
    if (telescopeState != "Ok" or ccdState != "Ok"):
        time.sleep(1)
    else:
        break

print("Connected to Telescope and CCD is established.")



#########################################################################
#IMU DATA READ AND USE###################################################
################## Compass Calibration values ############
# Use calibrateBerryIMU.py to get calibration values
# Calibrating the compass isnt mandatory, however a calibrated
# compass will result in a more accurate heading value.
# last calibrated on 2/23/2024
magXmin =  -3712
magYmin =  -504
magZmin =  -1769
magXmax =  -237
magYmax =  2180
magZmax =  -361

############### END Calibration offsets #################
def imudat():
        IMU.detectIMU()     #Detect if BerryIMU is connected.
        if(IMU.BerryIMUversion == 99):
```

```
                print(" No BerryIMU found... exiting ")
                sys.exit()
        IMU.initIMU()       #Initialise the accelerometer, gyroscope and compass

        gyroXangle = 0.0
        gyroYangle = 0.0
        gyroZangle = 0.0
        CFangleX = 0.0
        CFangleY = 0.0
        kalmanX = 0.0
        kalmanY = 0.0
#read accelerometer, gyroscope, and magnetometer values
        ACCx = IMU.readACCx()
        ACCy = IMU.readACCy()
        ACCz = IMU.readACCz()
        GYRx = IMU.readGYRx()
        GYRy = IMU.readGYRy()
        GYRz = IMU.readGYRz()
        MAGx = IMU.readMAGx()
        MAGy = IMU.readMAGy()
        MAGz = IMU.readMAGz()

        a = datetime.datetime.now() #time values were read

#Apply compass calibration
        MAGx -= (magXmin + magXmax) /2
        MAGy -= (magYmin + magYmax) /2
        MAGz -= (magZmin + magZmax) /2

#Convert Gyro raw to degrees per second
        rate_gyr_x =  GYRx * G_GAIN
        rate_gyr_y =  GYRy * G_GAIN
        rate_gyr_z =  GYRz * G_GAIN

#Calculate loop Period(LP). How long between Gyro Reads
        b = datetime.datetime.now() - a
        a = datetime.datetime.now()
        LP = b.microseconds/(1000000*1.0)
        outputString = "Loop Time %5.2f " % ( LP )

#Calculate the angles from the gyro.
        gyroXangle+=rate_gyr_x*LP
        gyroYangle+=rate_gyr_y*LP
        gyroZangle+=rate_gyr_z*LP

 #Convert Accelerometer values to degrees
        AccXangle =  (math.atan2(ACCy,ACCz)*RAD_TO_DEG)
        AccYangle =  (math.atan2(ACCz,ACCx)+M_PI)*RAD_TO_DEG

        #convert the values to -180 and +180
```

```
        if AccYangle > 90:
                AccYangle -= 270.0
        else:
                AccYangle += 90.0
#Complementary filter used to combine the accelerometer and gyro values.
        CFangleX=AA*(CFangleX+rate_gyr_x*LP) +(1 - AA) * AccXangle
        CFangleY=AA*(CFangleY+rate_gyr_y*LP) +(1 - AA) * AccYangle



    #Calculate heading
        heading = 180 * math.atan2(MAGy,MAGx)/M_PI

    #Only have our heading between 0 and 360
        #if heading < 0:
                #heading += 360


        return [gyroXangle, gyroYangle, gyroZangle, AccXangle, AccYangle, heading]

####GPS DATA#####
def GPSinfo():
        report = gpsd.next() #
        latitude = getattr(report,'lat',0.0)
        longitude = getattr(report,'lon',0.0)
        utctime = getattr(report,'time','')
        gpsAlt = getattr(report, 'alt', 'nan')

        return [longitude, latitude, utctime, gpsAlt]

######################################################################
######################################################################

def main():
        for i in range(0, int(caps)):
                fileform = 'image_*{}'.format(i)
                #this group takes the images

                print("Taking {} second exposure").format(expo)

                # Take exposure
                remote_object.setNumber("ZWO CCD ASI120MC-S", "CCD_EXPOSURE",
"CCD_EXPOSURE_VALUE", expo)
                remote_object.sendProperty("ZWO CCD ASI120MC-S", "CCD_EXPOSURE")


                # Wait until exposure is done
                ccdState       = "Busy"
                while True:
                        ccdState = remote_object.getPropertyState("ZWO CCD
```

```
ASI120MC-S", "CCD_EXPOSURE")
                        if (ccdState != "Ok"):
                                time.sleep(1)
                        else:
                                break

                print("Exposure complete")
                time.sleep(0.1)


                solver = "solve-field --crpix-center {}.fits
%s.solved".format(fileform)


                os.system(solver)

                gpsinfo = GPSinfo()
                IMUdata = imudat()
                barodat = bmp.388.get_temperature_and_pressure_and_altitude()

                longitude = gpsinfo[0]
                latitude = gpsinfo[1]
                utctime = gpsinfo[2]
                GPSalt = gpsinfo[3]

                GyroX = IMUdata[0]
                GyroY = IMUdata[1]
                GyroZ = IMUdata[2]
                AccX = IMUdata[3]
                AccY = IMUdata[4]
                head = IMUdata[5]


                tem = barodat[0]
                pres = barodat[1]
                alti = barodat[2]

                newfits = 'image_*{}.new'.format(i) #identifies solved fits image
                hdul = fits.open(newfits, mode = 'update') #opens solved image
                hdr = hdul[0].header #primary header

                RA = hdr['RA'] #reads and defines RA and DEC based on header
                DEC = hdr['DEC']
                CnRA = hdr['CRVAL1'] #reads and defines RA and DEC of image center
                CnDEC = hdr['CRVAL2']

                coord = SkyCoord(RA, Dec, unit=(u.hourangle, u.deg)) #ra/dec of
solved image in degrees
                print(coord)
                ra = coord.ra.degree
```

```
                dec = coord.dec.degree
                print(ra, dec)

                center = SkyCoord(CnRA, CnDEC, units = (deg, deg)) #ra/dec of solved
image center in degrees
                print(center)

#adding all the stuff to the fits header (yay)

                hdr['GRYOX'] = (GyroX, 'gryoscope x angle') #adds gryro data to
header
                hdr['GRYOY'] = (GyroY, 'gryoscope y angle')
                hdr['GYROZ'] = (Gyroz, 'gyroscope z angle')
                hdr['ACCELX'] = (AccX, 'accelerometer x value') #adds accelerometer
data to header
                hdr['ACCELY'] = (AccY, "accelerometer y value")
                hdr['GPSLONG'] = (longitude, 'GPS Longitude')#adds GPS data to FITS
file
                hdr['GPSLAT'] = (latitude, 'GPS Lattitude')
                hdr['GPSTIME'] = (utctime, "GPS Universal_Time")
                hdr['ALT'] = (alti, "Altitude")



#appends all the stuff to an array so i can plot it
                RiAs.append(ra)
                Decl.append(dec)
                CnRiAs.append(CnRA)
                CnDecl.append(CnDEC)
                UTime.append(utctime)
                GyX.append(GyroX)
                GyY.append(GyroY)
                GyZ.append(GyroZ)
                AcX.append(AccX)
                AcY.append(AccY)
                Glon.append(longitude)
                Glat.appned(latitude)
                ALt.append(alti)


                print("image_{} is done".format(i))
                print("Time: {}, Coordinates: {} , {}, SkyCoordinates:{} , {},
Altitude: {}").format(utctime,latitude, longitude, CnRA, CnDEC, alti)
                hdul.close(newfits)
                time.sleep(5) #waits 5 seconds before next loop through

        return print("Finished with captures")


#########################################################################
```

```
#run the main code
main()
#############################################################################
```