Final Report on Visual Attitude Estimation

Group Mike

AA 322 Final Report
2020-12-14

William Pope, Victor Xia, Jinhe Xu, and Alex Zhen
William E. Boeing Department of Aeronautics and Astronautics
University of Washington
Seattle, WA 98195-2400

Abstract: Star sensors are attitude sensors widely used in aerospace and astronautics due to their highly precise measurements. The objective of this project is to build a camera-based system that demonstrates the function of a star tracker at a much lower cost using a Raspberry Pi board. To accomplish this, algorithms for image processing, database construction, star identification and attitude determination were developed. The system was implemented on hardware using a Raspberry Pi camera and a star field pattern fixed to a wall. The device was able to visually estimate its attitude from the pattern, while tolerating optical error. The system was also implemented in simulation to iteratively test a large dataset to derive a relationship between success rate and computational runtime. The hardware and algorithms developed were able to successfully replicate an operational star tracker's function at a simpler and more accessible level.

Nomenclature

FOV	field of view of the camera
IMU	inertial measurement unit
MP	megapixels
nFov	number of stars visible in the current field of view
nMap	number of stars present in the entire star field
SR	success rate of the algorithm

I. Introduction

To succeed in a space mission, it is vital to have a reliable attitude determination system. Spacecraft use an array of sensors to determine attitude, such as magnetometers, accelerometers, sun sensors, and horizon sensors [1]. However, many of these sensors are subject to moderate amounts of bias and instability, and many are not suitable to interplanetary travel. Star trackers avoid these problems, and are generally recognized to be the most accurate sensor available for these applications [2]. However, they are quite expensive and not readily available for use.

Star trackers employ visual attitude estimation, and all operate according to a common general setup [3]. The sensor consists of an imaging device with a known relation to the spacecraft body frame and a computing unit. The camera uses a medium or wide field of view (FOV) to capture images of a portion of the celestial sphere. The images are then processed and compared with a star catalog stored on board to identify which stars are in the FOV. A number of these star identification algorithms are mentioned in [4]. Finally, an attitude determination problem is solved using the coordinates of FOV stars and matched catalog stars. Several solutions to this problem are available in the literature [5][6][7].

The use of star identification for attitude determination was first considered in 1969 [4], and since then the problem has been explored and used frequently. In this report, two commercial star trackers are used as references. Basic performance information on these star trackers is listed in Table 1.

Table 1: Star Tracker Basic Information

Product	FOV	Accuracy	Sensitivity (star magnitude)	Operation frequency (Hz)	Tracking Capacity
ROSAT precision star tracker [8]	5.9°×4.4°	2" - 10"	6.5	1	4
Ball's HAST [9]	8°×8°	0.5"	4.8 - 6.5	40 - 100	4-8

The objective of this junior capstone project was to apply the operating principles of a satellite-based star tracker to a cheap camera setup in order to explore sensing methodologies at an accessible level. An emulation of the sensor was built using a small camera, a Raspberry Pi,

and a dotted pattern on a wall. As with a star tracker, the attitude was determined using only visual input from the sensed environment. The fixed pattern acted as a star field, providing the attitude determination algorithm the reference points necessary to identify locations and orientations. The goal of the system was to determine a given camera orientation based solely on an image captured from that orientation. Over the course of the quarter, we determined that further insights could be gained by simulating our star tracker on vast randomized datasets, and set out to design a virtual experiment alongside our hardware implementation. From these goals, formal system requirements were created and displayed in Table 2 below.

Table 2: Requirements for Visual Attitude Estimation System

#	Requirement
1	The camera and Raspberry Pi shall capture and store a usable photo at a 99% success rate.
2	The image processing algorithm shall produce the correct location of contrasting points on a 2-D plane within a 10 pixel error margin.
3	The visual attitude estimation algorithm shall calculate a camera orientation angle within 2 degrees of the angle measured by the onboard inertial measurement sensor
4	The star identification algorithm shall positively identify target stars based on a reference photo with a 99% success rate at standard operating parameters for a commercial star tracker.

II. Theory

Despite the numerous choices of spacecraft attitude determination methods, star tracking is the most robust, effective, and commonly relied on choice. However, star tracking is not a plain subject nor a simple task to achieve; there are many free variables in the process. Some spacecraft take luminosity or brightness of stars into account, some use the light frequencies other than visible light emitted by the stars (IR, X-ray, etc.), while others take advantage of the parallax effect. The presence of large celestial bodies, other spacecraft, and other known parameters also affect the system design.

For this project particularly, it was decided only to use the stars' positions and a camera with known FOV. This allowed for a design that would be both simple and robust, while

accommodating the course's schedule requirements. It was also assumed that the stars are stationary, and that the FOV has narrow viewing angles. This means that the stars' positions in a spherical background can be approximated with a Cartesian coordinate system. The constraints of the test hardware (a Raspberry Pi board) meant that the system could not give real time attitude estimates due to limited processing power.

Brown and Stubis [10] 's work serves as a reference and inspiration for the development of the main algorithm used in this project. The general idea of this kind of algorithm is to build a high dimensional space with the traits and characteristics of the sample stars and then approach the objective either by regression, local optimization, or stepping down the dimensions. By the nature of the stars (discrete, sparse, and nondifferentiable), it is chosen to use the characteristics of the stars to build a three dimensional space where each star has a relatively unique coordinate, then shrink/step down the dimension of the search space to positively identify each star.

III. Experimental Apparatus

The hardware of this project was a sensor platform mounted atop an adjustable photography tripod head (Fig. 1.). The sensor platform contained a Raspberry Pi Camera module, an Adafruit LSM9DS1 9-DOF inertial measurement unit (IMU) sensor, and a Raspberry Pi 4 (RasPi). The RasPi was connected to the camera and IMU via cables and commanded both concurrently. The RasPi was connected to a monitor, keyboard, and mouse for the user interface.

The electronics were all mounted to a wooden board using M3 screws and standoffs. During testing, this allowed all devices to rotate together and maintain their relative angles. The tripod head has one handle for roll and one handle for pitch and yaw. In this experiment, only the roll handle was used to tilt the sensor platform in the plane parallel to the star field.

The physical star map was used for all hardware testing. It consisted of 42 small black paper dots, stuck to a white poster board in a random pattern. The star field was bounded by a red circle, which was used to calibrate the imaging algorithm as described in the following sections. The star map was 1 meter wide and about 1.25 meters from the camera.

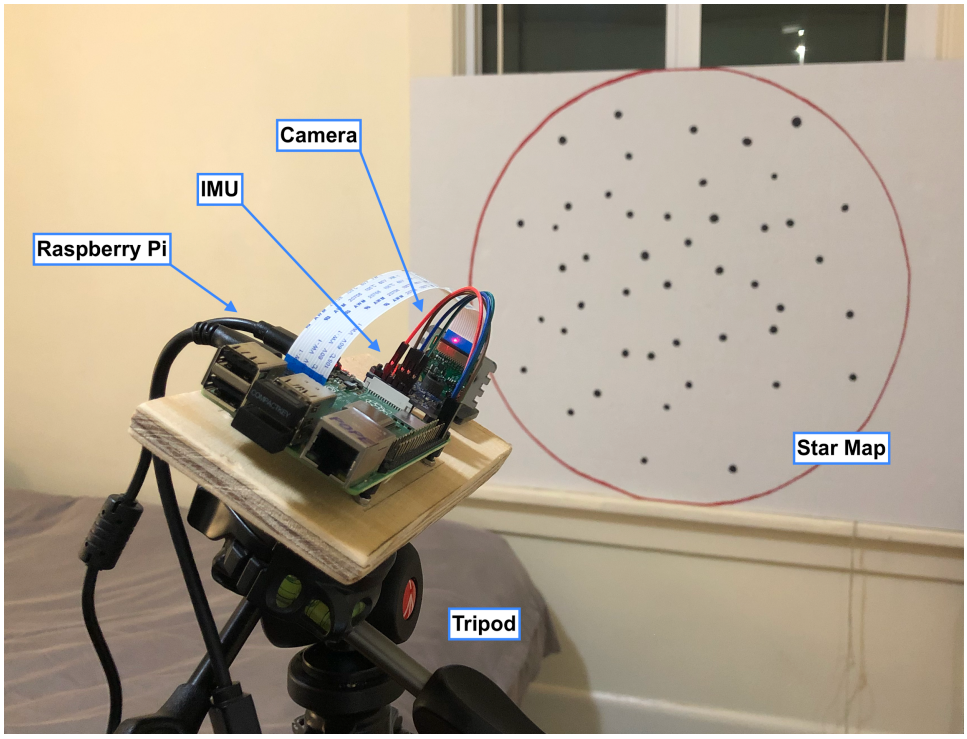


Fig. 1. The camera platform pointed towards the star map at a nonzero roll angle.

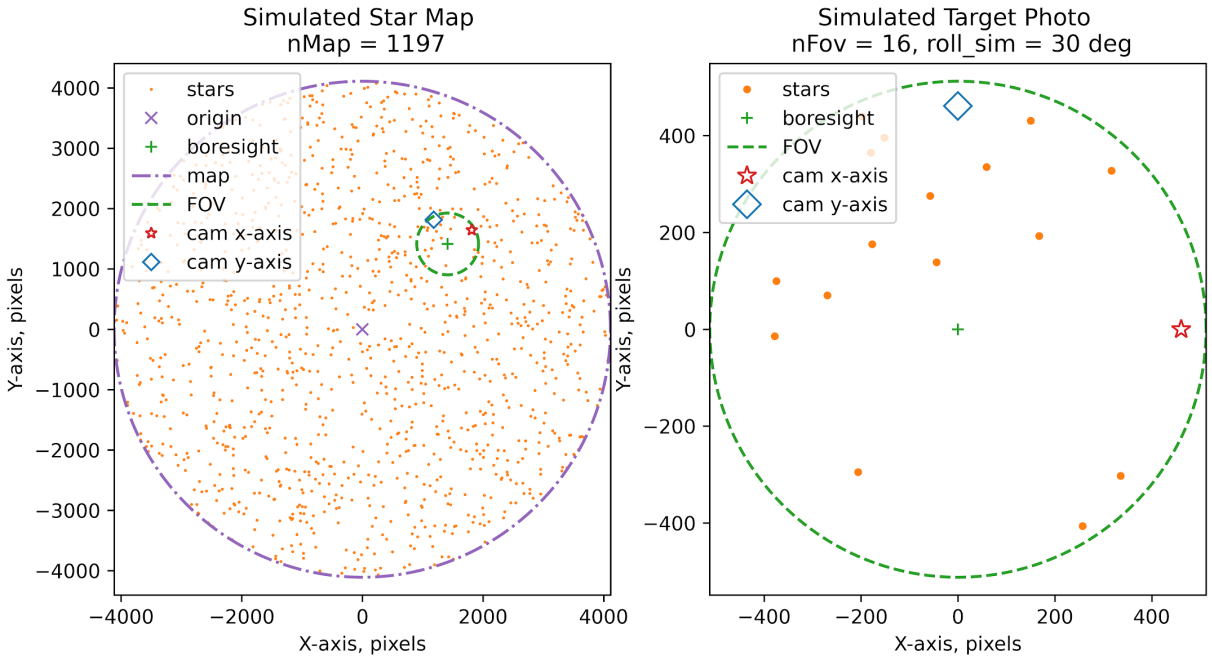
IV. Procedure

The project was split into two subsystems: image processing and star algorithms. Imaging handled all hardware and focused on processing a digital star image into usable coordinates for attitude determination. The star algorithms used inputs from the optical system to complete the primary objective, estimating the attitude. Furthermore, testing was split into hardware testing and simulated testing. Hardware testing allowed for use of image processing and challenged the algorithms with real world error. Simulated testing allowed for much larger iterative test sets and full control of inputs to probe the algorithms' behavior. Both test environments shared the same Python functions and algorithms, with additional infrastructure written to facilitate end-to-end experiments.

In the hardware case, a star map (nMap=42) was created on a poster board and fixed to the wall. The camera was set at zero degree roll (as measured by the IMU), then the user triggered a catalog photo. The catalog construction function then characterized each star in the found, in a process described below. Once cataloged, the user would tilt the camera to an arbitrary angle and trigger a target photo. The same image processing was conducted, but now the star identification

algorithm took over and positively matched stars in the new image to stars in the catalog. Finally the attitude determination algorithm estimated the change in angle between the catalog attitude and the current target attitude. This estimate was compared with the IMU's true attitude to determine success. The hardware testing was intended to demonstrate our baseline objective of attitude determination, and demonstrate that the system was robust to real world error.

The simulated testing was dedicated to test two key performance parameters for our star tracker design; one is the success rate (SR), and the other one is the algorithm runtime. The success rate is the overall probability of our system successfully determining the current attitude when the camera is pointing to a random direction on the celestial background. The algorithm runtime is the time needed for our system to finish one calculation of the attitude. These two parameters are directly related to the number of stars in the catalog (nMap), as more stars in the celestial background create more reference points but more calculations. For simulated testing, the poster and camera were replaced with a random scatter generator in Python. This function generated a randomly distributed star field on a much larger virtual 2-D plane (Fig. 2.a), from which a star catalog was constructed. Another function chose a location and roll angle to create a simulated field of view (Fig. 2.b), then fed those star coordinates to the identification algorithm and calculated an attitude.



(a) **(b)**

Fig. 2. Demonstration of the simulated star map and target photo (FOV). Plot (a) is a demonstration of the distribution of stars in a simulated star map. The purple dash-dot line is the imaginary boundary of the star field, but in reality the star field is a boundless spherical surface. Plot (b) is a simulated target photo in its body frame. The shape of the FOV is chosen to be a circle, which is also a practice for convenience when calculating rotations.

Using this method, the star algorithms could be tested on over a million unique scenarios in a matter of hours. Due to the randomness of each scenario the algorithm encountered, a large number of tests was required to average out any noise in the results. To test the relationship between the SR and nMap, 10 random star fields were generated while nMap was varied from 20 stars to 1600 stars. For each star map, around 1,200 different FOVs were selected for testing. To test the relationship between runtime and nMap, 100 random scenarios (one unique star map with one unique FOV) were generated while nMap was varied, and the average runtime of 100 scenarios was recorded. For every simulated test, the FOV radius was 1/8 of the star map radius (equivalent to 20° circular FOV in a spherical star field), so the FOV contained roughly 1/64 of all generated stars.

Dividing up the project allowed for more flexibility and a modular development approach. The building blocks of the system are described in further detail in the following subsections.

A. Image Processing

The imaging algorithm is needed to capture an image and convert it to usable coordinates for the star identification algorithm. It runs on open source Python libraries and employs a computer vision technique known as the circle Hough transform (CHT), which is able to find the center and radius of circles found within a digital image.

The process begins when the user initiates a new catalog or target photo. A photo is taken of the star map poster at max resolution (3280x2464 pixels, 8.1 MP) using standard Raspberry Pi camera procedures. Because of the different rotation axes in the tripod head, the circular star map is usually not exactly in the center of the camera's view. This must be corrected through software to keep the coordinate frames consistent between photos. To do this, a Hough transform is run on the raw image to locate the large ring surrounding the star map. The minimum search radius is set large so that the extraction will not detect the stars in the star map. Once the center and radius of the outer ring are found, the image is cropped to the exact coordinates of the circle, so that the origin of the star map is always placed at the center of the new image. From there, a second Hough transform is run with a much smaller minimum search radius. This pinpoints the center of each star in the image, and extracts the coordinates into a Python dictionary for further use by the star algorithms.

While the camera is taking a photo, the accelerometer in the IMU measures the acceleration of gravity along each of its three axes. From this measurement, a true tilt angle of the platform can be calculated and later compared to the output of the attitude determination algorithm.

B. Star Catalog Construction

The star catalog is a mass record that characterizes each star in the celestial field. In real applications, this catalog would be generated by telescopes on Earth and loaded onto the spacecraft. However in our case, it was conducted by the camera before testing began, and the process was known as "catalog mode". For every star, a catalog entry would be constructed to record its numeric ID, coordinates, and calculated characteristics. In order to do this, the system must iterate through every possible star and conduct the following process.

First, the area around a given star would be divided into three concentric ring-shaped sub-regions. For each sub-region, a series of vectors would be drawn from the given star at the center to the surrounding stars within the sub-region. Then a value V would be generated according to the following Eq. (1), where n is the number of stars in this sub-region, and $\boldsymbol{v}_{t,i}$ is each vector in this sub-region.

$$V = [0\ 0\ 1]\ * (\sum_{i=1}^{n} v_r \times v_{t,i}) \tag{1}$$

The vector with the smallest magnitude is $\boldsymbol{v}_r$. This value contains both the number and the position information of the stars around the center star.

Ultimately, each star would have three V values corresponding to each of its three rings. These values become the unique fingerprint of each star and are stored alongside the ID and coordinates within the star catalog.

C. Star Identification and Attitude Determination

Once catalog mode had been completed, the system entered flight mode, where it attempted to estimate attitude using said catalog. After reorienting to a new FOV, the same calculations would be run to produce fingerprints of the visible stars, which were currently unidentified. By checking the unidentified fingerprints with the catalog of known fingerprints, stars could be positively identified. Once matched, the translational and rotational difference between the current coordinates and the catalog coordinates of each star was calculated, and presented as an estimate of attitude.

The V values can be regarded as coordinates in a 3D space -- a virtual search space constructed such that every star uses its characteristics as its position. So the system would search for stars in the catalog with positions in the search space that are sufficiently close to one of the coordinates given by the stars in the photo. Any star in the catalog that matches two components with a coordinate would be recorded as a “candidate”, so that it is robust to faulty V values. Any matches with pure trivial V values (all $V=0$, no nearby stars) would be discarded. This process shrunk the range of candidates from the entire 3D search space to a 1D line with far fewer stars.

It is possible that a coordinate would have multiple corresponding candidates. To address this ambiguity, two facts of these candidates would be used. 1) Each subset of candidates clustered together in the 3D search space corresponds to a sample star in the photo. 2) There exists a subset of candidates clustered together in the 2D star map that are able to appear in the same photo as the sample stars are. Candidates can be cross referenced to each other to find the subset appearing in the photo, thus pinpointing the sample stars' IDs. If the determination process fails, the last resort of the system is to wait for the spacecraft attitude to change, in hope of a more friendly patch of stars.

D. Error Tolerance

Because of the resolution limits of the camera and other disturbances, star coordinates are not always exact between photos. This error needs to be addressed in the algorithm to allow for real world use of the system. Small changes in the relative position of stars could push a star into or out of the ring of another. To mitigate this, an error tolerance is applied at the edge of each ring to catch movement.

When calculating the tolerance of value *V* of each star's each ring, the algorithm uses Eq. (2).

$$tolerance = \sum_{i=1}^{n} (|v_r| + |v_{t,i}|) \cdot err \cdot [0\ 0\ 1] \frac{v_r \times v_{t,i}}{|v_r||v_{t,i}|} \quad (2)$$

In this equation, *tolerance* is the tolerance of the sample star's value *V* when it is compared with another *V* from the catalog. The variable *err* is the imaging error. When calculating the tolerance when comparing angular distance, the tolerance would double err, because the error of the distance between two stars is simply adding two errors together. In the hardware test, *err* was set up to 3 pixels and for the simulated test, *err* was set up to 0, since there was no error introduced to the system.

V. Results and Discussion

A. Hardware Test Results

With the software fully developed, hardware testing was executed as described in the 'Procedure' section. The first objective of the imaging system was to simply output a photo, as

shown in Fig. 3. The camera produced high resolution images and was able to handle undesirable lighting conditions in the test area.

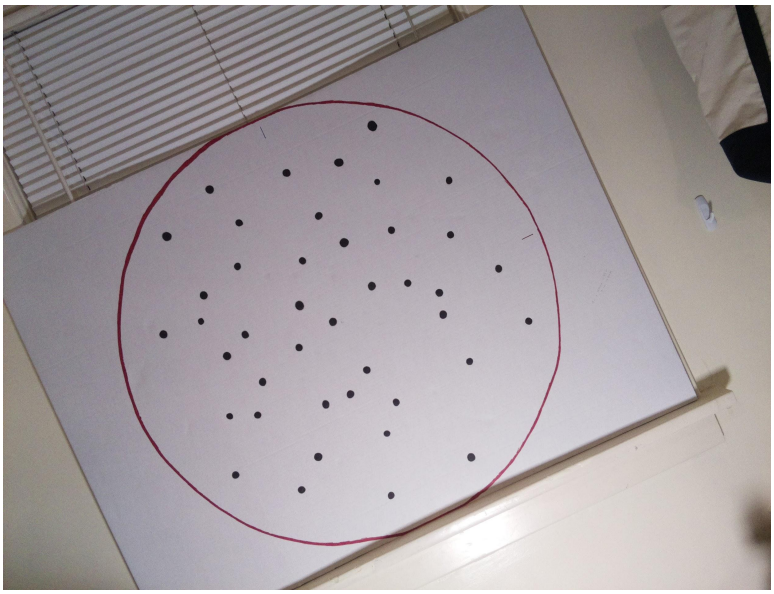


Fig. 3. Raw target image taken by RasPi camera, fed to image processing algorithm.

With usable photos, they could be fed into image processing and star algorithms to fulfill the main objective of the project, visual attitude estimation. With photos like Fig. 3, the algorithms created catalogs and matched stars to a fairly high degree of accuracy. With a fully integrated hardware/software system running, the attitude output appeared as in Fig. 4.

Figure 4 demonstrates the estimation process from start to finish. In the raw photo, the star map is arbitrarily located towards the left side of the photo, while surrounding objects are also visible in the frame. The first step of image processing is to automatically locate the large red ring and lock the frame to it. This is seen in the output image, as the frame is the exact size of the ring's diameter. Then, the next Hough transform attempts to locate every star in the image. This is not an exact process, as shown by the false positive star located above the poster. It is common for the transform to pick up 1-2 extra stars on the peripheral, but the star algorithms are not affected by this error. With the stars located, the identification and attitude calculations take place for this given photo. In this case, it has estimated the attitude to be -25.12 degrees from the catalog attitude (zero degrees). The IMU takes its own attitude measurement, which is slightly different from the one calculated visually. The three frames (camera, estimated inertial, true

inertial) are represented by horizon lines superimposed on the star field. The two small lines drawn at the edge of the star field are not visible to the algorithm, but provide a final verification that the estimation was successful.

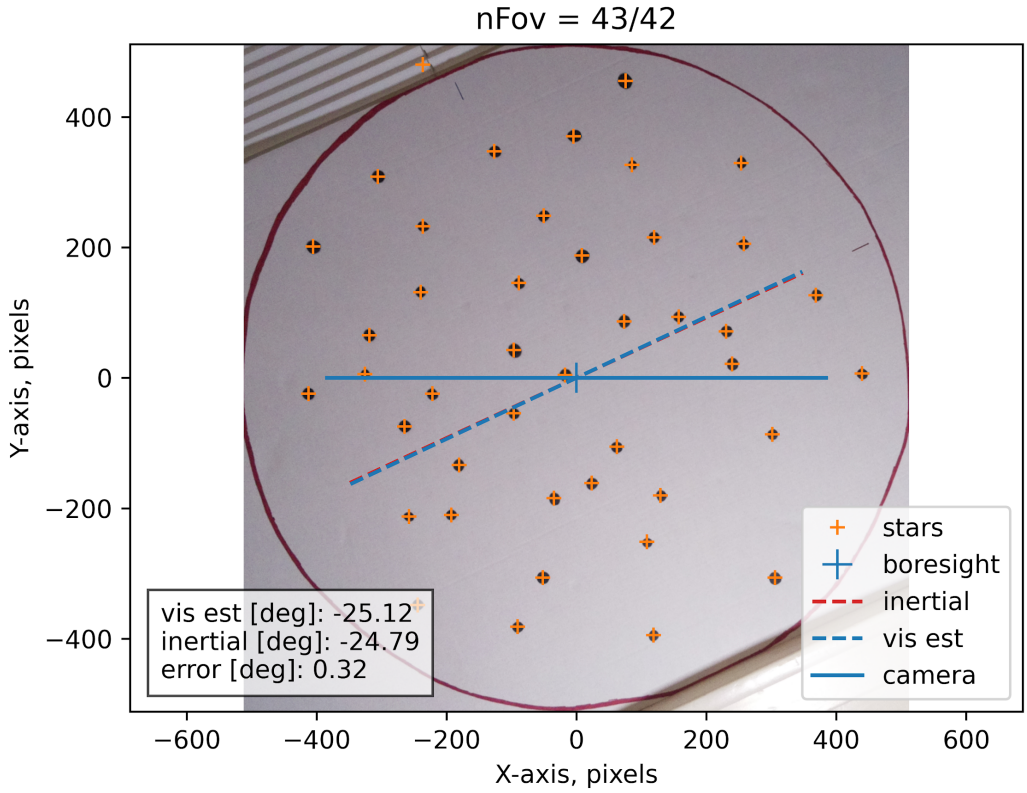


Fig. 4. Demonstration of visual attitude estimation implemented on Raspberry Pi hardware. Note the blue lines denoting the camera frame and the visually estimated true frame. The dashed red true inertial frame (measured by IMU) is barely visible because the angle error is small.

Under the given test conditions, the hardware implementation was fairly successful with a rate of 38 positive matches for 40 estimation attempts. The tripod head only rotated between negative 45 and positive 90 degrees, which limited the usable test range. A sample of successful runs over this range is shown in Table 3.

When similar tests are performed in simulation, the error is perfectly zero for all successful matches. This indicates that any error is produced by the hardware system, and not inherent to the estimation algorithm. The angle error represents the angle difference between the IMU angle and the estimated angle, while the position error represents the pixel difference between the true

origin (0,0) and where the algorithm thinks the FOV is centered. Based on the design of the system, it appears that these errors come from two different places.

Table 3. Angle and position estimates from testing through a full range of motion of tripod head. The 'Center Norm' is the distance from the star map's origin to the camera's boresight.

True Angle [deg]	Est Angle [deg]	Angle Error [deg]	True Center Norm [px]	Est Center Norm [px]	Position Error [px]
-44.44	-45.18	0.74	0	6.750	6.750
-31.11	-31.61	0.50	0	6.246	6.246
-14.39	-14.50	0.11	0	9.574	9.574
0	-0.03	0.03	0	2.287	2.287
18.75	18.70	0.05	0	4.297	4.297
31.34	31.21	0.13	0	6.436	6.436
45.09	44.10	0.99	0	7.235	7.235
63.59	63.10	0.49	0	11.779	11.779
75.99	74.62	1.37	0	12.217	12.217
91.06	89.09	1.97	0	17.412	17.412
	Average:	0.638 deg		Average:	8.423 px

The angle is determined by matching stars to the relative positions of stars in the catalog. When a pattern of stars is identified, there is only one possible angle that will match it back to the catalog orientation. Therefore, it can be assumed that when a positive match is found, the angle is almost exactly accurate to the orientation of the camera in the 2-D star frame. This angle is compared with a measurement from the IMU, which is actually more likely to fluctuate out of agreement with the visual estimate. This is because the IMU reading is subject to +/- 0.5 degree while stationary, and the platform rotation is not solely in the 2-D plane parallel to the poster. Even with this uncertainty, the angle reading is within the desired range. More accurate

measurement tools would be needed to determine the true inaccuracy of the visual estimation scheme.

The position of the FOV boresight is determined by finding the center of the large ring on the poster. This is what the location of each star is measured off of, since the star field's position within the raw image frame can vary wildly with each reposition. With the boresight determined, the identification algorithm estimates the translational coordinate of each star from the catalog boresight, as if the camera had intentionally been pointed to a different spot. However, the boresight is supposed to remain at (0, 0), since it is using the same outer ring every time. Any change in this position, which averages to 8.423 pixels in Table XX, can be attributed to the inaccuracy of finding the exact center of the ring through a Hough transform circle extraction. This error is also attributable to experiment design, and less to the design of the actual star tracking system.

On first attempt, the hardware implementation failed every time because it had no margin for error built in. Once it was apparent that the coordinate readings would not be precise in every photo, a set pixel margin was added in to allow the algorithm to still create positive matches, with just a small amount of error in the result. In some cases, the coordinate error was severe enough that the system did not recognize the stars it viewed, resulting in the two failed attempts. While the error present here was a result of our DIY test setup, these types of errors are just as prevalent in the use of actual star trackers. No system operates in a perfect environment, so processes must always be adaptable and robust to an imprecise world.

B. Simulated Test Results

Simulated testing was intended to allow for larger and more varied datasets for the algorithm to test on. There are two important performance indicators: success rate (SR) and the algorithm runtime. Both of these are directly related to the number of stars in the entire celestial map (nMap). The relationship between nMap and the performance indicators was produced empirically by iterating through over a million random test sets, as described in 'Procedure'. The results quantify how effective the system is, and what it needs to be successful.

In the modeling of the relationship of nMap and the algorithm runtime, we assumed it to be a quadratic polynomial because the ring method has a $O(n^2)$ time complexity. The ring method iterates through every star in the FOV; then, the method compares every star in the catalog to

identify the star from the FOV. Thus, two nested loops together make a $O(n^2)$ time complexity. The coefficients of this model are obtained from a linear regression, which are

$$f(x) = x^2 + \beta_1 x + \beta_0 \tag{3}$$

$$\beta_2 = 1.0696\times10^{-6},\ \beta_1 = 1.1215\times10^{-5},\ \beta_0 = -0.0020$$

Figure 5 is the plot of this linear regression. The coefficient of determination, R^2,for this regression is equal to 0.999, which indicates excellent curve fitting. As expected, the relationship exhibits quadratic behavior.

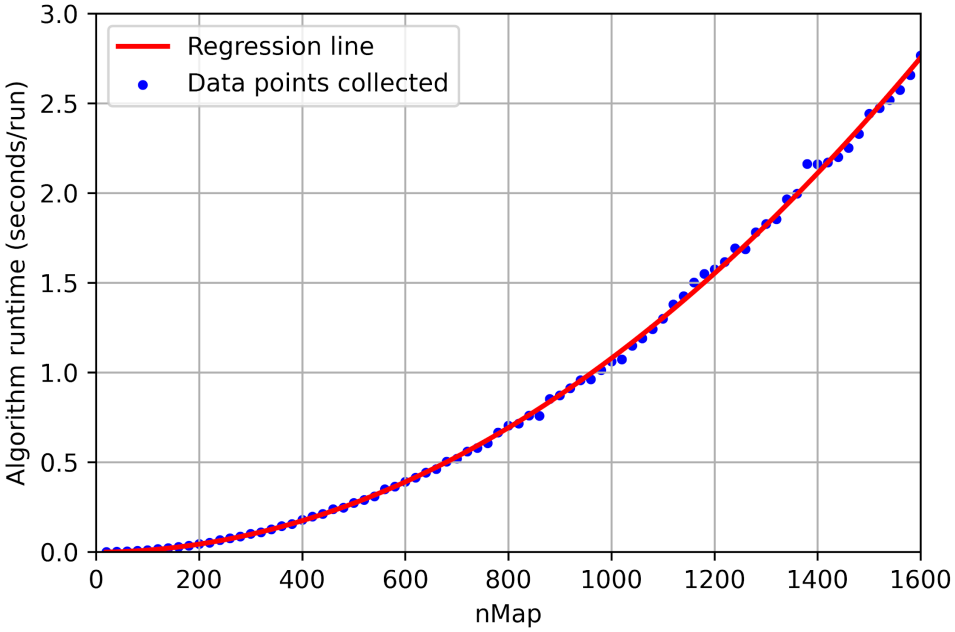


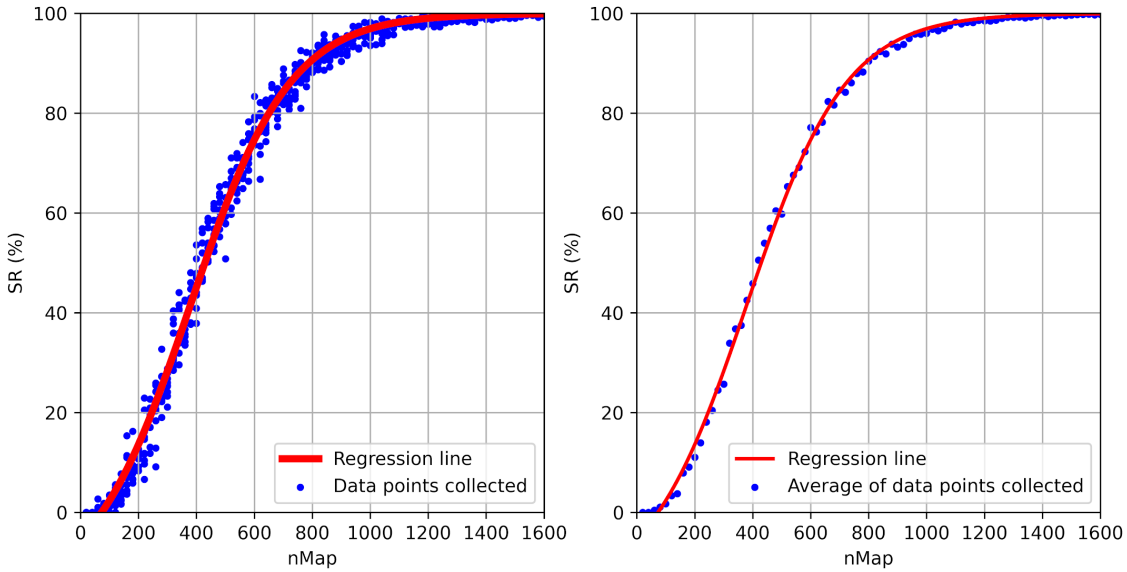
Fig. 5. Relationship between algorithm running time and nMap. The red line is the linear regression line, and the blue dots are the data points.

For the relationship between the nMap and the algorithm's success rate, we modeled it to be a logistic variance function. One can make three logical predictions of the system's behavior. One is that when nMap is extremely low, i.e., zero stars, SR is zero, as there is no information to identify stars. Next is that the SR will increase as nMap increases. Last is that SR will asymptotically approach unity as nMap approaches infinity. The more stars in the catalog, the

more information can be utilized when identifying stars. Still, because the stars are randomly distributed in the celestial background, there is always a possibility that the richness of stars' information is not enough for the algorithm in some areas. Thus, we came up with this mathematical model (Eq. 4) of the relationship between nMap and the algorithm's SR.

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}} - (L-1) \tag{4}$$

The coefficients in the non-linear model were calculated empirically from a curve fitting (non-linear regression) of the simulated data. The values of the coefficients are calculated to be $L = 1.1683$, $k = 5.8179\times10^{-3}$, and $x_0 = 379.39$. Figure 6.a is the plot of both the simulation data and the curve obtained from Python curve fitting. As there is no standard method to evaluate the success of a non-linear regression, we can look at the closeness between the regression line and the mean of all data points. In Fig. 6b, the regression line overlaps the mean of the data points very well. Thus, the validity of this model is demonstrated.



(a) **(b)**

Fig. 6. Relationship between nMap and algorithm SR.

With the mathematical models of SR and runtime, a relationship between these two parameters can be established, as seen in Fig. 7. This relationship is useful for designing the star tracker's optical sensitivity, which determines how many stars will be fed to the algorithm. The trade-off between SR and runtime is highly mission dependent. For example, suppose the star tracker on a mission only serves the purpose of an infrequent attitude calibration. In that case, a high SR may be desired, so the slowest possible runtime is used. On the contrary, another mission may need the star tracker to update the attitude state estimation in real time, so a fast sample time is required, allowing some failures in the name of speed. Some commercial star trackers have different operating modes to be able to adapt to different situations like this.

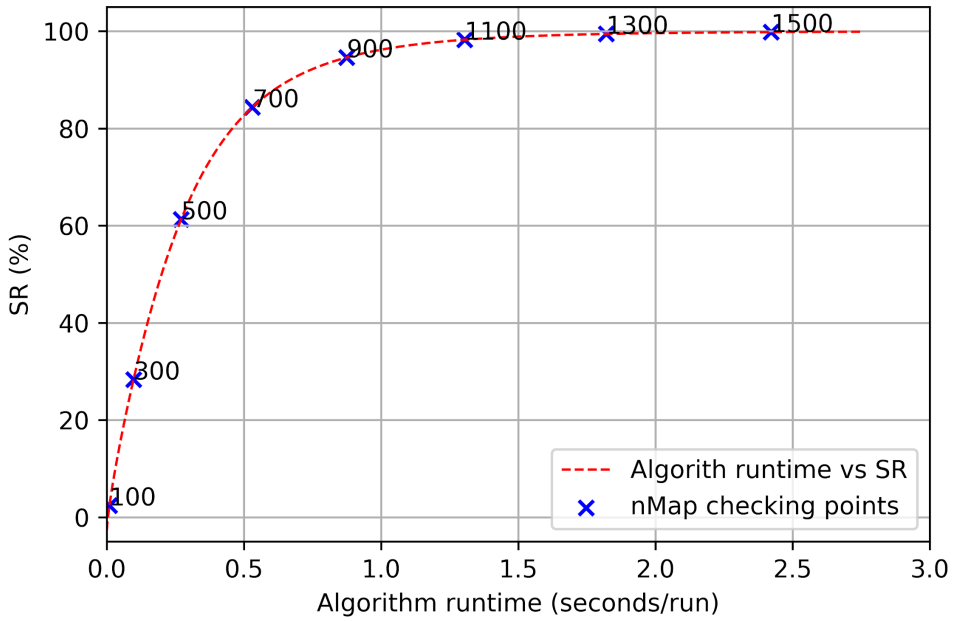


Fig. 7. Relationship between the algorithm running time and SR. The red dashed line is the estimated relationship obtained from setting up Eq. 3 and Eq. 4 equal. The blue cross points are some checking points to see the values of nMap at those points.

If we want a SR higher than 99% (achieved by most commercial star trackers), the number of catalog stars needed is calculated to be 1197, and the algorithm runtime is limited to 1.5490 seconds. A typical star tracker will equip a catalog with 2405 to 5000 stars, based upon different hardware and algorithms, so our design outperformed the general requirement of a star tracker, albeit in an error free environment [11]. The runtime speed clocked by our system can

not compete with commercial star trackers, who are able to estimate an attitude in less than a second [9]. Even in the 1980s, star trackers were able to run at a 1 second period, including imaging time which was excluded from our simulated test [8]. However, algorithm runtime will vary for different computing units.

Based on the nMap required, the optical sensor sensitivity can be derived. With 1197 stars distributed across a whole sphere, the star density is equal to 0.0028 stars per square degree. From Table 4, we can estimate the absolute magnitude threshold for our design to be 5. All star tracker sensors have a sensitivity limit, so star magnitudes lower than a certain threshold will not be captured by the sensor. Both Ball's HAST and ROSAT star tracker are able to detect apparent magnitudes lower than 6.5 (the higher, the dimmer). Our star tracker algorithm is thus proven to operate under this reference hardware limitation.

Table 4. Target star distribution [8]

This table shows the density distribution of stars at different magnitudes.

Apparent Magnitude, *m*	Stars Per Square Degree Brighter than *m*
8	1.0
7	0.34
6	0.12
5	0.04
4	0.013
3	0.0044

VI. Conclusions

This project demonstrated a practical implementation of end-to-end star tracker operation, conducted at a course appropriate level. Using cheap Raspberry Pi hardware and accessible Python code, algorithms for image processing, database construction, star identification, and attitude determination were all developed and tested. All project requirements were completed successfully within the constraints of the quarter.

While simplifying assumptions were made and experimental conditions weren't perfect, our system was able to replicate and explore both the optical and computational elements of a star tracker. The relationships derived between optical sensitivity, success rate, and calculation time have very real implications for use within spacecraft control systems. While we demonstrated these relationships in this project, the next step for a future project would be to figure out how to improve the relationships and optimize performance. A system like ours can serve as a testbed for new algorithms and implementation schemes, allowing students to continue to explore methods of visual attitude estimation.

Acknowledgments

We wish to thank our faculty advisor for this project, Professor Mehran Mesbahi, as well as course instructors Professor James Hermanson, Professor Erik Hurlen, and TA Andrew Jensen. Additionally, we'd like to thank the W.E. Boeing Department of Aeronautics & Astronautics for their financial support of the project.

References

[1] R. Kandiyil, Attitude Determination Software for a Star Sensor, Lulea University of Technology, Germany, 2009

[2] C. Liebe, "Accuracy performance of star trackers—A tutorial", IEEE Transactions on Aerospace and Electronic Systems, vol. 38, no. 2, pp. 587-599, 2002.

[3] E. Silani, M. Lovera, Star identification algorithms: novel approach & comparison study, IEEE Trans. on Aerospace and Electronics Syst. 42 (2006) 1275–1287.

[4] M. Na, P. Jia, A survey of all-sky autonomous star identification algorithms, 2006 1st International Symposium on Systems and Control in Aerospace and Astronautics, Harbin, China IEEE, Los Alamitos, 2006, pp. 896–901.

[5] M. Shuster and S. Oh, "Three-axis attitude determination from vector observations", Journal of Guidance Control and Dynamics, vol. 4, no. 1, pp. 70-77, 1981.

[6] F. Markley, "Attitude determination using vector observations: A fast optimal matrix algorithm", The Journal of the Astronautical Sciences, vol. 41, no. 2, pp. 261-280, 1993.

[7] D. Mortari, "Euler-q algorithm for attitude determination from vector observations", Journal of Guidance Control and Dynamics, vol. 21, no. 2, pp. 328-334, 1998.

[8] Marcel J.S, Spacecraft Dynamics & Control A practical Engineering Approach, Cambridge University Press, 1997, Appendix B.4
[9] “Star Trackers,” Aerospace, Available: https://www.ball.com/aerospace/markets-capabilities/capabilities/technologies-components/star-trackers.
[10] Brown, J., & Stubis, K., “TETRA: Star Identification with Hash Tables”, 31st Annual AIAA/USU Conference on Small Satellites, SSC17-VIII-3.
[11] J. Li, G. Wang and X. Wei, "Generation of Guide Star Catalog for Star Trackers," in IEEE Sensors Journal, vol. 18, no. 11, pp. 4592-4601, 1 June1, 2018