AAS 20-028

DESIGN AND PERFORMANCE OF AN OPEN-SOURCE STAR TRACKER ALGORITHM ON COMMERCIAL OFF-THE-SHELF CAMERAS AND COMPUTERS

Samuel Pedrotty,[*] Ronney Lovelace,[†] John Christian,[‡] Devin Renshaw,[§] Grace Quintero[**]

Recent frustration in finding low size, weight, power (SWaP), cost, and lead time star trackers has driven an internal research and development effort at Johnson Space Center (JSC) in partnership with Rensselaer Polytechnic Institute (RPI) to develop and demonstrate a commercial off-the-shelf (COTS) camera and COTS computer-based star tracker system. A set of open-source algorithms has been developed and their function demonstrated on multiple low-cost COTS single board computers (SBCs) across a variety of operating systems and COTS cameras. The goal of this effort is to release the software and setup guide to the community in order to reduce spacecraft development costs while increasing their capability (perhaps most of interest to low-cost missions like CubeSats). This material will show the high level architecture of the system, detail the algorithm, various tested configurations, and results. Forward work and applications will also be discussed.

INTRODUCTION

Star trackers are ubiquitous spacecraft sensors that typically provide high-accuracy attitude information to the spacecraft's guidance, navigation, and control (GNC) system. These sensors typically use some sort of optical detector to collect light from a portion of the sky and then use an integrated computational capability to turn the detected light into a series of centroids that are matched against a catalog of known stars to produce an estimate of the detector's attitude in inertial space.[1]

While the cost, form factor, and lead time of star trackers has remained fairly constant over the last decade, that of optical detectors (in the form of COTS cameras) and computational capability (in the form of COTS computers) have significantly improved. Advances have certainly been made in star trackers and certain units have significantly improved on their cost, form factor, and lead time, but they still are not close to the improvements seen in COTS cameras and COTS computers. Others have seen the opportunity to implement algorithms on COTS systems in order to improve cost, lead time, and form factor over purpose-built space-grade star trackers.[2,3] The downside of most of these implementations is that they are tailored to a particular set of hardware, which usually

[*] Aerospace Engineer, EG/Aeroscience and Flight Mechanics Division, NASA Johnson Space Center, Houston, TX 77058
[†] Aerospace Engineer, EG/Aeroscience and Flight Mechanics Division, NASA Johnson Space Center, Houston, TX 77058
[‡] Assistant Professor, Dept. of Mechanical, Aerospace, and Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180
[§] Student, Dept. of Mechanical, Aerospace, and Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180
[**] Student, Dept. of Mechanical, Aerospace, and Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180

either quickly leaves production or is far behind the performance of newer market entries. A flexible alternative is needed that supports multiple platforms and is readily transferable and upgradeable.

The goal of this effort is to develop and demonstrate a set of software that is able to capture imagery from a variety of COTS cameras and compute a resulting attitude accurate to <0.5 degrees (with no additional information) all within a few seconds on a variety of COTS computers. This software must be easy to develop and maintain, open-source, and easily transferable between computers and their associated operating systems. While this capability will likely be of greatest use to small, low-budget satellites, larger spacecraft may also find use for this as a backup attitude determination method. Such spacecraft are more likely to already have cameras and spare computational capability.

CURRENT STAR TRACKER MARKET

Most star trackers available commercially are comprised of robust high performance sensors with a high technology readiness level (TRL). The cheapest of these sensors range in the tens of thousands of dollars and many require lead times of many months. While this performance may be welcome for some satellite missions, their costs are prohibitive to many CubeSat missions that emphasize reduced development cost.

Modern commercial star trackers, like those in Table 1, operate with both a lost-in-space mode and with a tracking mode. While in lost-in-space mode, these star trackers image a star field and then compare the unique distribution of stars in the star tracker's field-of-view to an onboard star catalog to determine attitude. Once a reasonable attitude has been found, the star tracker will then switch to the tracking of stars to determine attitude at a faster rate.

The SWaP and lead time of these star trackers is a driving factor in sensor selection for most CubeSat missions. The size and weight of star trackers currently on the market is largely dependent on whether or not the star tracker has an incorporated baffle to mitigate stray light. Many CubeSat star trackers forgo incorporating a baffle to save on size, and instead use operational planning to mitigate stray light. Other star trackers, like the Blue Canyon NST, incorporate a bullpup-like baffle design. Small form-factor star trackers without a baffle typically fit a 50x50x50 mm^3 volume, while an incorporated baffle will usually double the length of a star tracker. For other satellites, a star tracker can be much larger, like the Jena-Optronik Astro APS which fits a volume three times larger than a CubeSat star tracker.

Table 1: Selection of Available Star Trackers

Product	**Manufacturer**	**Accuracy** (cross/about)	**Size** mm^3	**Mass** grams	**Power** Watts	**Cost**
Space Sextant	Adcole Maryland Aerospace	4" / 27"	50x50x50	170	1.5	$32,500*
KULST	KU Leuven	2" / 10"	45x50x95	250	< 1	$33,400†
NST-3	TY-Space	5" / 70"	50x50x50	165	n/a	$33,400‡
NST	Blue Canyon	6" / 40"	100x55x50	350	1.5	n/a
ST400	Hyperion Technologies	10" / n/a	48x57x89	280	0.7	n/a
Astro APS	Jena-Optronik	1" / 8"	154x154x237	2000	12	n/a

SYSTEM DESIGN AND INITIAL IMPLEMENTATION

The system goals of ease of development, open-source, and portability between computers and operating systems drove the selection of Python as the primary language of the software. Python is an open-source, interpreted, object-oriented, high-level programming language, which makes it very easy to use.[4] Its common usage has resulted in a wealth of support across hardware platforms and reusable algorithms, known as "modules." As this preliminary version of the system is viewed as a proof of concept, optimization is not a concern (e.g. no need to implement portions of the software in C for speed improvements).

For candidate COTS SBCs, units were selected that could support Unix operating systems, were ready to ship, were smaller than 10x10x10 cm, had USB and Ethernet ports, were cheaper than $500, and were fairly dissimilar. While a large number of computers fit this criteria, time constraints limited development efforts to a Raspberry Pi 3B+ and an Odroid XU4Q. A Windows laptop was also used to simplify development and demonstration. The specifications of these platforms are detailed in Table 2.

Table 2: Tested Platform Specifications

Computer	**Operating System**	**CPU**	**RAM**	**Disk**	**Python version**	**OpenCV version**
Odroid XU4Q	Ubuntu 18.04	Samsung Exynos5422 (Cortex-A15 and Cortex-A7)	2GB LPDDR3	64GB SanDisk Extreme U3 card	2.7.17	3.2.0
Raspberry Pi 3B+	Raspbian Stretch	Broadcom BCM2837B0, Cortex-A53	1GB LPDDR2	64GB SanDisk Extreme U3 card	2.7.13	2.4.9.1
2018 Dell Precision 7720	Windows 10	Xeon E3-1535M	32 GB DDR4	512 GB NVMe PCIe SSD	3.6.4	4.0.0

* https://www.cubesatshop.com/wp-content/uploads/2016/06/MAI-SS-Specification-10-11-17.pdf, 02/2020
† https://www.cubesatshop.com/product/kul-star-tracker/, 02/2020
‡ https://www.cubesatshop.com/product/nst-3-nano-star-tracker/, 02/2020

Initially, it was desired to be completely camera agnostic. Experimentation with various generic camera modules with basic USB Video Class drivers proved that this was not feasible as these units lacked gain control and therefore could not discern stars from space. This led to the pursuit of low-cost COTS cameras that provided explicit driver support and Python APIs. Just like the SBCs, there were many cameras that fit this criteria, but time and resources drove the selection of a Ximea MQ013MG-E2 and an IDS UI-3180CP-M-GL R2. The specifications of these cameras are detailed in Table 3.

Table 3: Tested Camera Specifications

Camera	Resolution	Sensor Size	Body Size	Pixel Size
MQ013MG-E2	1280x1024	6.9x5.5 mm	26x26x26 mm	5.3 µm
UI-3180CP-M-GL R2	2592x2048	12.44x9.83 mm	29x29x29 mm	4.80 µm

For the purpose of this project, our lens selection allowed for two different fields-of-view to give some insight to the star tracker algorithm's behavior. Lens selection for a star tracker sensor is mainly a compromise between the desired accuracy of the star tracker's attitude determination and the ability for the star tracker's algorithm to match stars. The specifications of these lenses are detailed in Table 4.

Lens field-of-view can impact solve times and success. If a star tracker's field-of-view is too narrow, the star tracker may not have enough suitable stars in view to match to its catalog. Conversely, if the star tracker's field-of-view is too large, the star tracker algorithm may take longer to converge to a solution. This is due to the larger number of stars for the algorithm to match and the added difficulty of distinguishing smaller apparent stars from noise.

Table 4: Tested Lens Specifications

Lens	Angular FoV	Focal Length	Iris Range
HF35HA-1B	14°20′ × 10°46′	35 mm	F1.6 - F22
HF12.5HA-1B	38°47′ × 29°35′	12.5 mm	F1.4 - F16

ALGORITHM DESIGN

The algorithmic approach used to process star field images and return a sensor attitude follows the conventional paradigm for star trackers.[5] This paradigm works essentially as follows: (1) raw images undergo photometric and geometric calibration, (2) image processing algorithms find candidate stars in the calibrated image, (3) image pixel coordinates for each star centroid is converted to a bearing direction, (4) star identification (ID) algorithms match candidate star bearings to a catalog of known star bearings, and (5) pairs of corresponding measured/catalog star bearings are used to compute attitude via a solution to Wahba’s problem. These five tasks are all

performed in real-time and onboard the sensor's flight computer and we refer to them as the "online workflow."

These online tasks rely on specialized pre-computed data (e.g., calibration, curated star catalog) that is computed infrequently (e.g., once preflight, once during a post-launch checkout). We refer to these infrequent tasks as the "offline workflow."

OFFLINE WORKFLOW

Catalog Creation

Modern star catalogs contain a great deal of astrometric data for a large number of stars. The star ID and attitude determination algorithms used in the online workflow rely only on star geometry, so we primarily care about the accurate modeling of star bearings. While a variety of star catalogs exist, we choose to use the Hipparcos catalog which encodes star bearing information using a five-parameter model: [6] [7]

$$\boldsymbol{v}_i = \langle \boldsymbol{\ell}_i + \left(t - t_{ep}\right)\left(\mu_{\alpha*_i}\boldsymbol{p}_i + \mu_{\delta_i}\boldsymbol{q}_i\right) + \varpi \boldsymbol{r}_i / A_u \rangle$$

$$\boldsymbol{\ell}_i = \begin{bmatrix} \cos(\delta_i)\cos(\alpha_i) \\ \cos(\delta_i)\sin(\alpha_i) \\ \sin(\delta_i) \end{bmatrix} \quad \boldsymbol{p}_i = \begin{bmatrix} -\sin(\alpha_i) \\ \cos(\alpha_i) \\ 0 \end{bmatrix} \quad \boldsymbol{q}_i = \begin{bmatrix} -\sin(\delta_i)\cos(\alpha_i) \\ -\sin(\delta_i)\sin(\alpha_i) \\ \cos(\delta_i) \end{bmatrix}$$

where the five astrometric parameters are: α_i is right ascension, δ_i is declination, $\mu_{\alpha*_i}$ is proper motion in the right ascension direction, μ_{δ_i} is the proper motion in the declination direction, and ϖ is the annual parallax. The star catalog directions are stored at a reference epoch time t_{ep} and the observations occur at time t. The position of the sensor relative to the solar system barycenter (SSB) is given by $\boldsymbol{r}_i$, and A_u is one astronomical unit (given in the same units as $\boldsymbol{r}_i$). The triangle brackets $\langle\cdot\rangle$ indicate vector length normalization. Star directions in Hipparcos are given relative to the International Celestial Reference Frame (ICRF), so this naturally becomes the "inertial" frame used for computing sensor attitude.

A star tracker like the one considered in this study cannot generally see very dim stars, thus there is no use in retaining such stars in the onboard catalog. Therefore, we remove stars dimmer than a specified threshold (specified as a maximum apparent magnitude). For all stars brighter than the threshold, we compute the inter-star angle for every unique star pair,

$$\theta_{ij} = \cos^{-1}(\boldsymbol{v}_i^T \boldsymbol{v}_j)$$

We can further reduce future computations by excluding star pairs whose inter-star angle exceeds the camera's field-of-view, since such a pair could never be observed by the sensor. This significantly reduces the number of star pairs to search later in the pipeline.

We now wish to save this catalog of inter-star angles in a way that can be efficiently searched. As has become a popular practice, we do this using the *k*-vector construct developed by Mortari and Neta.[8]

Dark Frame Computation

Most CCD/CMOS detectors experience some form of fixed-pattern noise (FPN) that can frustrate the easy detection of dim point sources (e.g., stars). This is generally handled by computing a dark frame (an image where no photons strike the focal plane), and then removing this known dark

frame from each image. Without a mechanical shutter on our camera, we are unable to block external light from entering the aperture and striking the focal plane. Instead, we approximate the dark frame by computing the median intensity value amongst an ensemble of random star field images. Supposing that any given pixel will contain a star in fewer than half of the images, this provides a good approximation of the dark frame. The same approach is used for the Orion OPNAV camera.[9]

Distortion Map Computation

Since we are working with real cameras with real lenses, we must also account for the distortion of light that occurs before it strikes the sensor. We choose to model this distortion using the Brown model, which is the same mathematical model used for geometric calibration of the Orion OPNAV camera.[10] [11] Use of the Brown model is convenient, as both OpenCV and MATLAB have built-in toolboxes that will undistort an image given the Brown distortion parameters.[12] Regardless, we have developed our own custom Python scripts to implement these well-documented algorithms for the present application.

ONLINE WORKFLOW

Image Acquisition and Processing

Each image that the camera captures must undergo some basic image processing to remove detector FPM and lens distortions. The FPN is removed by subtracting the dark frame, which was precomputed as part of the offline workflow. Note that the dark frame and the image must be collected by the same camera, with the same settings (e.g., exposure time), and under similar operational conditions (e.g., detector temperature). Substantial differences in settings or operating conditions may invalidate existing dark frames and necessitate the construction of a new dark frame.

Once the dark frame is removed, the image is undistorted using the undistortion map constructed as part of the offline workflow. This approach was selected to leverage heritage from the authors' existing codebase. It is also possible to simply undistort the specific star centroids, but this was not the approach taken here.

Finding and Centroiding Candidate Stars

We search the calibrated image for groups of pixels that resemble star patterns. Like most star trackers, a small amount of defocus blurs the stars producing a Gaussian-like point spread function (PSF) on the camera sensors. Since not all such bright spots are stars, we refer to these as *candidate stars* until they are successfully identified. Following the approach of Lang et al., we find candidate stars by (1) applying a median filter to the image, (2) subtracting the original image the filtered image, and (3) searching for contiguous spots of a specified brightness and size.[13] Once these spots are found, we compute the centroid by using the center of intensity method, which is known to be robust and is computationally inexpensive.[14] The result is the 2D subpixel coordinate of each candidate star.

Computing Bearing to Candidate Stars

Since the image was undistorted in the first step of the pipeline, 2D image coordinates are related to bearings by the pinhole camera model.[15] This may be done by converting pixel $[u_i, v_i]$ coordinates into image plane $[X_i, Y_i]$ coordinates.[16]

$$\begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{d_x} & \frac{\alpha}{d_x d_y} & \frac{d_x \alpha - d_y u_p}{d_x d_y} \\ 0 & \frac{1}{d_y} & -\frac{v_p}{d_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

where α is the skew parameter of the camera, u_p and v_p are the principal axes, u_i and v_i denote the candidate pixel location, and d_x and d_y are given below

$$d_x = {}^{f}/_{\mu_x}, \quad d_y = {}^{f}/_{\mu_y}$$

where f is the focal length and μ_x and μ_y are the pixel pitch in the x and y directions. The bearing unit vector we seek ($\boldsymbol{u}_i$) is then found by normalizing the resulting vector on the left-hand-side,

$$\boldsymbol{u}_i = \left(X_i^2 + Y_i^2 + 1\right)^{-1/2} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix}$$

Star Identification and Attitude Estimation

We pass the measured bearing vectors to a triangle star search algorithm that processes sets of three stars at each iteration of a loop. To avoid reusing false positive stars repeatedly for numerous successive iterations, we employ an algorithm to search through the catalog with as few repeated stars as possible; this algorithm reduces the number of successive cycles without reuse of recent stars.[17] Our algorithm is a modification of a reflector identification (REFLID) algorithm, which itself is a modification of the Pyramid star ID approach.[18] [19]

For each triad of candidates, we compute the inter-star angle of each of the three pairs formed by those three stars (i.e., stars A and B, B and C, and A and C); we refer to AB as the angle between candidates A and B, and similarly to the other two pairs as BC and AC. Each of the inter-star angles is then used as the search parameter in the *k*-vector search algorithm. This search will return the indices of all catalog star pairs that fall within a specified range.

For each of the star indices in one of the results (we choose candidates of A), we find corresponding matches of A in AC, as well as matches for B in BC. If the number of matches of A in AC and the number of matches of B in BC are each exactly one, and the corresponding C stars match, then we proceed to the next sequence, or repeat this loop until we find a set that succeeds, or we exhaust our list and exit.

Once we find a set of three stars that meet our matching requirements above, we then compute the attitude of the potential match. This is done using singular value decomposition (SVD) solution of the Wahba problem, which directly produces the 3x3 attitude matrix.[20] [21]

Using this attitude matrix, we transform the candidate star bearings into the potential inertial frame. For each of the candidate star bearings, we compute the angle between the transformed measurement and every star from the catalog. If only one star falls within a specified angle tolerance, we increment the number of found matches by one and try the subsequent candidate star. If the number of candidates that we match to the catalog exceeds the minimum number of stars specified by the system designer, the algorithm returns the number of matches found, the attitude matrix that achieved the matches, and the corresponding match indices from the catalog.

TERRESTRIAL PERFORMANCE CHARACTERIZATION

Imagery, corresponding truth data, and camera calibration file was made available from the Houston Orion Test Hardware (HOTH) rig optical navigation test bed. The imagery was taken by a pixelink camera of a screen with rendered star fields in the HOTH. The imagery and associated data was copied to the Pi3, the Odroid, and the Windows machine which were all at the same state of the star tracker software (but different Python and Python module states as shown in Table 2). A new catalog was created to match the camera parameters including only stars brighter than apparent magnitude 5.0. The star tracker code was then run and the resulting accuracy and time to compute a solution was recorded. The Pi3 and the Odroid were also running the htop command line utility to gauge resource usage and had a couple file explorer windows open. The Windows machine had many other applications open, but more than 50% of all computational resources were still available prior to testing. Each run was repeated three times to gain additional insight into runtime variability.

Accuracy and solve time results for a single run are shown below in Figure 1 and Figure 2. Even though they have different Python modules, the Odroid and Pi3 recorded nearly identical attitude solutions with 16 images within 0.02 and 0.049 degrees of the truth. The Windows machine accuracy was slightly higher with 18 images within 0.02 and 0.049 degrees of the truth and the error spread shifted closer towards 0. Solve times (the time it takes to process the image and compute an attitude estimate) were fairly consistent across runs for the Pi3 and the Windows machine. The Odroid, however, had two fairly consistent runs, but the third run had solve times that were nearly double the others.

Algorithm-related system resource usage was approximated by observing htop during the runs. Both Unix systems had a single core running at 100% and approximately 100 MB RAM dedicated to the process. The Windows machine had a single core at about 75% and approximately 100 MB RAM dedicated to the process (determined using the Task Manager).

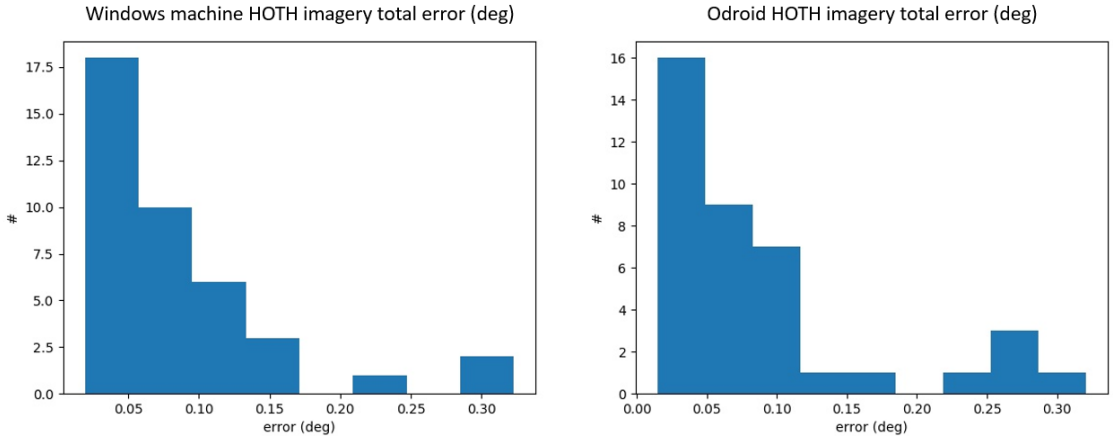

Figure 1: Total error histograms for the HOTH data with the Windows machine on the left and the Odroid on the right

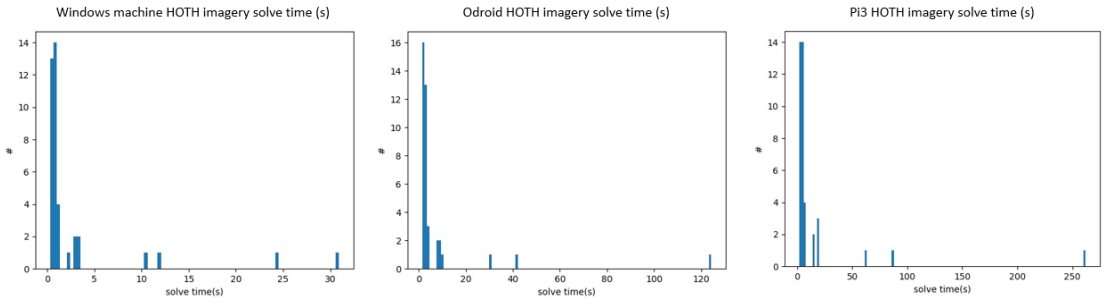

Figure 2: Solve time histograms for the HOTH data with the Windows machine on the left, Odroid in the middle, and Pi3 on the right

Imagery, corresponding truth data, and camera calibration file was made available from the Orion star tracker International Space Station Detailed Test Objective (ISS DTO). The imagery was taken by a pixelink camera onboard ISS looking out cupola window 2. It should be noted that the corresponding "truth" data supplied with this image set has an error of approximately -0.235 degrees in the inertial X axis. As above, the imagery and associated data was copied to the Pi3, the Odroid, and the Windows machine (with the same software configuration as above). A new catalog was created to match the camera parameters including only stars brighter than apparent magnitude 5.0. Again, the star tracker code was then run, recording accuracy and solve time, and htop and Task Manager were used to assess the system's resource usage. Each run was repeated three times to gain additional insight into runtime variability.

Accuracy and solve time results for a single run are shown below in Figure 3 and Figure 4. The aforementioned -0.235 degree bias is evident in the per-axis breakdown below in Figure 5 and is the major contributor to the total error. With the bias, it is hard to determine which platform performed best. The Windows machine had the fastest solve times at around 0.25 s, then the Odroid with 1.5 s, and finally the Pi3 at 2.0 s. As before, the Windows machine and Pi3 solve times across the runs were fairly consistent while the Odroid had two consistent runs and a third which was approximately 50% slower. Resource usage was approximately the same as the HOTH imagery.

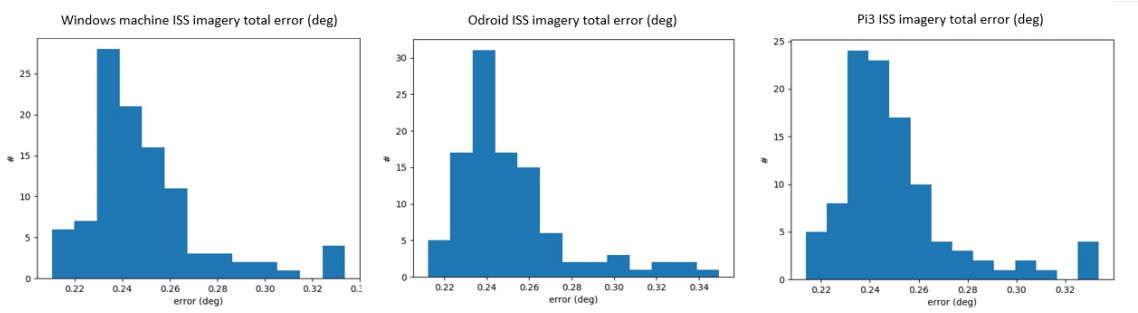

Figure 3: Total error histograms for the ISS data with the Windows machine on the left, Odroid in the middle, and Pi3 on the right

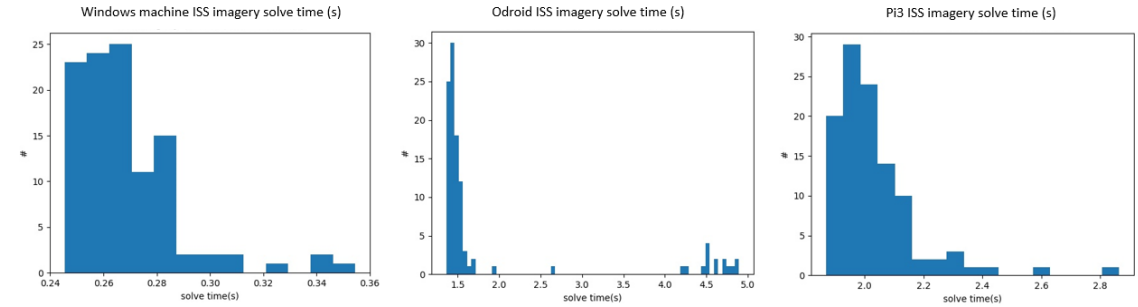

Figure 4: Solve time histograms for the ISS data with the Windows machine on the left, Odroid in the middle, and Pi3 on the right

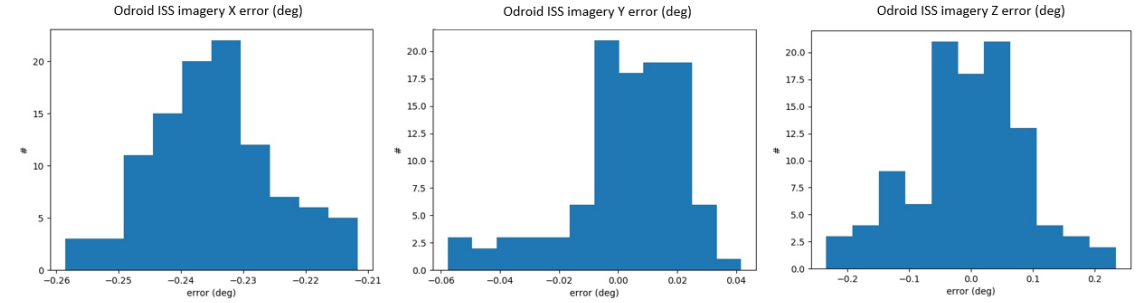

Figure 5: Inertial X, Y, and Z axis errors (degrees) on the ISS imagery (left, middle, and right, respectively)

The cameras and SBCs were also demonstrated in an integrated fashion with a live sky where the SBCs automatically configured the cameras, streamed imagery, and generated attitude solutions. Both SBCs returned solutions for the Ximea camera at approximately 2 Hz and for the IDS camera at approximately 0.3 Hz. No corresponding truth data has been generated for this imagery, so the error is unknown.

CHALLENGES

The process of developing the algorithms, configuring the software and hardware platforms, and integrating them presented many challenges. The first challenge encountered in this process (aside from the usual challenges associated with developing and implementing algorithms) was the deployment of the algorithms across very dissimilar computer systems. Some had very limited RAM, which required changes to the software installation process. The next challenge was rectifying the differences between critical packages on the systems (like NumPy and OpenCV). A significant difference between two computer's ability to appropriately match centroids to the catalog was traced to a difference in packages where one computer had a March 2017 release and the other had a September 2017 release.

Perhaps the largest challenge was extracting stars from an image. It became apparent early in the development and testing of the system that cameras with generic UVC drivers provided very little control over camera parameters critical to low-light imagery (e.g. gain). This was a much larger impact than the pixel or detector size. COTS cameras are also fairly noisy, which requires dark frame subtraction and careful selection of thresholding parameters.

Once stars are being centroided by the algorithms, another challenge is then adjusting various input parameters to facilitate a quick and accurate match to stars in the catalog. This goes all the

way back to parameters in catalog creation and also includes steps like acquiring a reasonable camera calibration. Any failure in those steps or other poor inputs such as the interstar angle tolerance or the minimum number of matches will cause the algorithm to match the wrong stars, take many minutes to create a match, or completely fail to make a match.

FORWARD WORK

The current state of the software and the efforts described by this paper are just the first steps towards an easy to configure, robust capability for spacecraft. Near-term efforts include comparing the performance to the MIT TETRA code and the University at Buffalo Nanosatellite Laboratory OpenStarTracker while also adding additional transformations to enable comparison to Astrometry.net's output.[22] [23] [24] [*] [†] [‡] Such comparisons and improvements would allow for more informed development of the core algorithms in order to further improve accuracy and reduce the time to compute a solution. Additional areas of focus include automating the process of tuning camera, algorithm, and catalog parameters and building a database of low-cost cameras, computers, and the algorithm's resulting performance on them.

In order to advance the TRL of this system, flight opportunities will also be sought. This technology is likely suitable for either a suborbital flight demonstration (e.g. sounding rocket or balloon) or an orbital demonstration (e.g. ISS experiment), depending on the opportunity.

The end goal of this effort is to create a useful capability for the space community (large and small spacecraft alike). In order to maximize the community's access to the software, an open-source release is currently being pursued. It is hoped that in the near future the software will be available on a common software repository platform like GitHub.

CONCLUSION

This effort has successfully achieved its goals of providing an attitude estimate better than 0.5 degrees on a variety of low-cost COTS SBCs and gathering star images with a variety of COTS cameras. This is a significant step towards creating an easy-to-implement capability that is compatible with a variety of low-cost COTS SBCs and cameras, but much work remains. It is hoped that those interested in this capability will contact the authors and that an improved version will soon be available as open-source software.

ACKNOWLEDGEMENTS

The authors would especially like to thank our families for their understanding and patience as we covered the furniture in SBCs and cameras and stared at the sky long into the night. In addition, the authors would like to thank the JSC Technology Working Group for their support of this project and seed funding and Steve Lockhart, Dave Saley, Greg Holt, John McGregor, Malak Samaan for their support.

[*] http://openstartracker.org/, 02/2020
[†] https://github.com/brownj4/Tetra, 02/2020
[‡] https://github.com/UBNanosatLab/openstartracker/wiki, 02/2020

REFERENCES

[1] C. C. Liebe, "Star trackers for attitude determination," IEEE Aerospace and Electronic Systems Magazine, vol. 10, pp. 10-16, June 1995.

[2] NASA Ames Research Center, "Low Cost Star Tracker Software," https://www.nasa.gov/sites/default/files/at-oms/files/arc-16289-1_low_cost_star_tracker_software.pdf, 2015.

[3] S.T. Gutierrez et al., "Open Hardware and Software Star Tracker: An Opportunity for Collaboration in the Emerging Cubesat Community," Preprints 2017.

[4] Python Software Foundation, "What is Python? Executive Summary," https://www.python.org/doc/essays/blurb/, 2020

[5] C. C. Liebe, "Accuracy performance of star trackers - a tutorial," IEEE Transactions on Aerospace and Electronic Systems, vol. 38, pp. 587-599, April 2002.

[6] European Space Agency, "The Hipparcos and Tycho Catalogues," ESA SP--1200, 1997.

[7] J. A. Christian, "StarNAV: Autonomous Optical Navigation of a Spacecraft by the Relativistic Perturbation of Star-light," Sensors, vol. 19, p. 4064, 2019.

[8] D. Mortari and B. Neta, " k-Vector Range Searching Techniques," Advances in the Astronautical Sciences, vol. 105, pp. 449-464, 2000.

[9] J. Christian, L. Benhacine, J. Hikes and C. D'Souza, "Geometric Calibration of the Orion Optical Navigation Camera using Star Field Images," The Journal of the Astronautical Sciences, vol. 63, pp. 335-353, 2016.

[10] A. E. Conrady, "Decentered Lens-Systems," Royal Astronomical Society, vol. 79, pp. 384-390, 1919.

[11] D. C. Brown, "Decentering Distortion of Lenses," Photogrammetric Engineering, vol. 32, pp. 444-462, 1966.

[12] G. B. a. A. Kaehler, Learning OpenCV, O'Reilly, 2008.

[13] D. Lang, D. Hogg, K. Mierle, M. Blanton and S. Rowels, "Astrometry.net: Blind astrometric calibration of arbitrary astronomical images," Instrumentation and Methods for Astrophysics, vol. 139, no. 5, pp. 1782-1800, 2010.

[14] L. Benhacine, "Optical Aberrations and their Effect of the Centroid Location of Unresolved Objects," West Virginia University, 2017.

[15] L. Ma, Y. Chen and K. and Moore, "A New Analytical Radial Distortion Model for Camera Calibration," 2003.

[16] J. Christian, "Accurate Planetary Limb Localization for Image-Based Spacecraft Navigation," Journal of Spacecraft and Rockets, vol. 54, no. 3, pp. 708-730, 2017.

[17] D. Arnas, M. Fialho and D. Mortari, "Fast and robust kernel generators for star trackers," Acta Astronautica, vol. 134, pp. 291-302, 2017.

[18] C. Ertl and J. Christian, "Identification of Partially Resolved Objects in Space Imagery with Convolutional Neural Networks," The Journal of the Astronautical Sciences, 2019.

[19] D. Mortari, M. Samaan, C. Bruccoleri and J. Junkins, "The Pyramid Star Identification Technique," Navigation, vol. 51, 2004.

[20] F. L. Markley, "Attitude Determination Using Vector Observations and Singular Value Decomposition," Journal of the Astronautical Sciences, vol. 36, no. 3, pp. 245-254, 1988.

[21] G. Wahba, "A Least Squares Estimate of Spacecraft Attitude," SIAM Review, vol. 7, no. 3, p. 409, 1965.

[22] A. Tennenbaum, "Automatic Star-tracker Optimization Framework," Small Satellite Conference, 2017.

[23] J. Brown and K. Stubis, "TETRA: Star Identification with Hash Tables," Small Satellite Conference, 2017.

[24] D. Lang et al., "Astrometry.net: Blind Astrometric Calibration of Arbitrary Astronomical Images," The Astronomical Journal, 2009.