ABSTRACT

Title of Document:	VISION ALGORITHM FOR THE SOLAR ASPECT SYSTEM OF THE HIGH ENERGY REPLICATED OPTICS TO EXPLORE THE SUN MISSION
	Alexander Krishnan Cramer, M.S., 2014
Directed By:	Professor Ramalingam Chellappa, Department of Electrical and Computer Engineering

This work covers the design and test of a machine vision algorithm for generating high-accuracy pitch and yaw pointing solutions relative to the sun on a high altitude balloon. It describes how images were constructed by focusing an image of the sun onto a plate printed with a pattern of small cross-shaped fiducial markers. Images of this plate taken with an off-the-shelf camera were processed to determine relative position of the balloon payload to the sun. The algorithm is broken into four problems: circle detection, fiducial detection, fiducial identification, and image registration. Circle detection is handled by an "Average Intersection" method, fiducial detection by a matched filter approach, and identification with an ad-hoc method based on the spacing between fiducials. Performance is verified on real test data where possible, but otherwise uses artificially generated data. Pointing knowledge is ultimately verified to meet the 20 arcsecond requirement.

VISION ALGORITHM FOR THE SOLAR ASPECT SYSTEM OF THE HIGH ENERGY REPLICATED OPTICS TO EXPLORE THE SUN MISSION

By

Alexander Krishnan Cramer

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2014

Advisory Committee:
Professor Ramalingam Chellappa, Chair
Professor Christopher Davis
Professor Larry Davis

Acknowledgements

I want to thank the HEROES team. My contribution was a small part of very a big project, and I am immensely proud of what we accomplished together. Working on a mission from the design stages all the way through to launch was a unique experience, and seeing the payload we'd worked on so hard for so long floating miles above us is a sight I don't think I'll ever forget.

I'd especially like to thank Dr. Steven Christe and Dr. Albert Shih, HEROES Co-PI and Solar Project Scientist respectively. Developing this algorithm was a collaboration, and my first serious foray into programming. They shared the load when generating the rest of the software for the SAS proved to be far too much for me, and provided invaluable input and assistance both testing and refining the algorithm itself.

Finally I'd like to thank my supervisor at NASA, Dr. Umesh Patel, for guiding me through this entire process. He has been an excellent advisor and has kept me motivated, especially through the writing process. Without his help and feedback I doubt I would have ever finished this thesis, and never have found the confidence I needed to defend it.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This thesis examines the problem of generating solar pointing solutions for the high-altitude balloon payload mission, High Energy Replicated Optics to Explore the Sun (HEROES). Specifically, this work covers the generation of pitch and yaw attitude solutions generated by processing images of a carefully constructed scene onboard the balloon. The algorithm design was influenced by the details of the scene and by requirements of the mission. Mission requirements include those explicitly levied on the pointing system and those which are common to most flight software. Pointing system requirements measured in arcseconds translated to sub-pixel requirements on object detection and location mapping portions of the algorithm. The computer had limited resources and was already devoting computation time to command and telemetry functions, which combined with a strict cadence requirement severely limited the computation time available for processing each image. Bandwidth limits on the telemetry stream during the balloon flight meant that processing had to be able to run reliably with minimal supervision. This work describes how the requirements on accuracy, computation time, and robustness led to a specific approach to analyzing the scene.

The problem of generating solar aspect is common to many missions. Knowledge of aspect relative to the sun is especially important for solar observatories like HEROES, but it is also commonly used by other spacecraft as part of a suite of sensors for determining attitude. Additionally, any system employing solar panels can also make use of this knowledge to optimize harvested power. The approach taken for the PYAS is different from that taken for other similar systems because it achieved fine solar pointing

using relatively inexpensive electronics. In contrast, many similar systems require the use of carefully tuned photo-detectors, FPGAs and other purpose-built electronic hardware. In the case of the HEROES sun sensor, a generic computer was used to run purpose-written software.

Work began on the HEROES mission in 2012 as part of the Hands-On Project Experience (HOPE) program at NASA Goddard Space Flight Center and Marshall Space Flight Center. The goal of HEROES was to adapt an existing balloon payload, known as HERO, for solar observation. The High Energy Replicated Optics (HERO) payload was a NASA high-altitude balloon project aimed at making observations of astrophysical targets in the hard x-ray spectrum. The HERO payload had flown several times since its first flight in 2001 meaning its existing systems had a long heritage [1]. HERO used a combination of differential GPS and gyroscopes for coarse position sensing, combined with an on-axis star camera for fine sensing and drift-correction in the gyros. This suite of sensors combined with a closed-loop controller allowed the HERO payload to achieve pointing stability of 1 arcminute.

While the HERO x-ray optics and detectors could handle solar observation, the on-axis star camera was unable to acquire star fields within about 1° of the sun. In fact, pointing directly at the sun actually had potential to damage the star camera sensor. The HEROES project encompassed many upgrades to the existing HERO payload, one of which was an additional fine-pointing sensor specifically for solar pointing, dubbed the Solar Aspect System (SAS). The SAS itself was further divided into two systems: the Roll Aspect System (RAS) and the Pitch-Yaw Aspect System (PYAS). Since roll data was not actionable for the gondola, RAS was responsible for simply storing images that

could be processed on the ground for roll knowledge. The PYAS, on the other hand, would fill the role of the star camera during solar observations. This meant that the PYAS had to process frames in real-time in order to provide pitch and yaw knowledge to the gondola pointing system during the flight, in addition to storing frames for refinement on the ground post-flight. This thesis will cover the design and test of this in-flight processing.

The remainder of this thesis is organized as follows. Chapter 2 provides background on the HEROES mission, details of the PYAS optical and mechanical elements, the scene observed by the PYAS, and assumptions made about that scene. Chapter 3 covers existing work. It addresses both existing sun-trackers, and existing algorithms that could be applied to this specific computer vision problem. Chapter 4 outlines the details of the algorithm used by the PYAS. Chapter 5 covers performance of the algorithm. This includes performance on synthetic test data and pre-flight test data. This chapter also provides a brief summary of the results from the 2013 flight of the HEROES payload. Finally, Chapter 6 provides a summary of algorithm performance, some lessons learned, and recommended changes for any future PYAS system based on this same design.

2 Background

2.1 Introduction

This chapter has covers the background of the HEROES mission and the place of the PYAS in that mission. The relevant details of the PYAS optical, mechanical and electrical hardware were given, as well as a discussion of how these come together to form the PYAS scene. The algorithm described in this thesis is heavily influenced by the details of the images formed and observed by the PYAS hardware. Finally there will be a list of simplifying assumptions made in the algorithm design. These assumptions are justified based on the PYAS hardware, and are important for keeping the PYAS processing problem tractable.

2.2 PYAS System Requirements

To replace the star camera during solar pointing, the PYAS had to meet similar requirements. The PYAS was expected to provide pitch and yaw offsets from a target anywhere on the sun with an accuracy of better than 20 arcseconds to the gondola control system (CTL), and to do so with a cadence of 1 Hz. Each frame also needed to be saved for post-processing on the ground. These solutions had to be generated any time the sun was within the field of view (FOV) of the PYAS. The HERO payload was capable of pointing with an accuracy of ±1° in pitch and yaw on coarse sensing alone, at which point it depended on fine sensing to stabilize on a target. For solar pointing, the target could lie anywhere on the sun. Because the sun is roughly .5° across, ensuring the sun was always fully on the screen after coarse pointing, regardless of where the target was placed on the sun, would require at least a 3° FOV. The PYAS was only able to achieve a 2.8° FOV, so

it was decided that it would only be required to provide a coarse solution for a partially-visible sun. Because of the narrow FOV of the x-ray telescope, accuracy at the edges of the FOV was less important. Degraded solutions were permissible there, as they only had to be accurate enough to tell the CTL roughly where to slew to bring the sun fully onto the screen. A summary of the relevant requirements on the SAS are shown in Table 1. The entire system is a closely coupled pair of both hardware and software, but the focus here will be on software. Specifically, this will be a detailed description of the algorithm used for converting raw PYAS images into pitch and yaw offsets.

Requirement	**Minimum**	**Target**
Cadence (storage)	1 Hz	4 Hz
Cadence (processing)	1 Hz	4 Hz
Accuracy (flight)	< 20 arcseconds	< 10 arcseconds
Accuracy (post-flight)	< 15 arcseconds	< 15 arcseconds
FOV	3°	2.8° (fine) >3.3° (coarse)

Table 1: List of relevant requirements for the HEROES PYAS

2.3 PYAS System Overview

The design of the PYAS hardware went through several iterations, but in its final iteration the hardware was laid out as in Figure 1. A CCD camera indirectly observes the sun by imaging a plate onto which the sun is projected. The optics for handling this projection consisted of three optical elements. First there was an IR filter to block heat, followed by a band-pass filter to make the incoming light monochromatic. This reduced the effect of chromatic aberration when passing through the last element, a plano-convex lens with 3m focal length. This lens was focused on the fiducial plate 3m away. This plate itself was covered in a printed pattern of fiducial markers as shown in Figure 2. The markers were spaced in a pattern designed to allow registration of the focused solar

image, even when only a fraction of the screen was illuminated. The entire assembly was then surrounded with baffles to cut down on stray light, ensuring the focused solar image was the only source of illumination on the fiducial plate. Taking advantage of precise knowledge of the locations of each fiducial marker relative to the HEROES scientific payload, the hope was to determine the payload's orientation relative to the sun by locating the projected image of the sun relative to the fiducials.

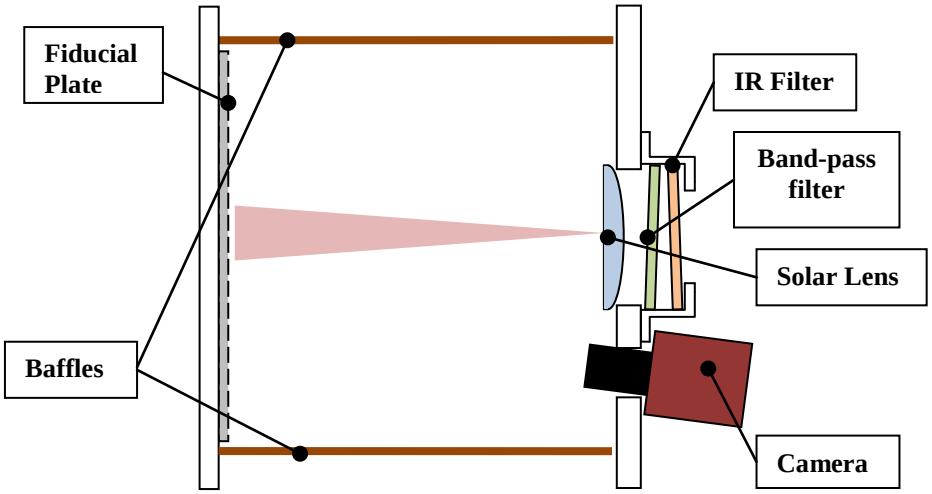

Figure 1 Diagram of the PYAS optics for a single PYAS system

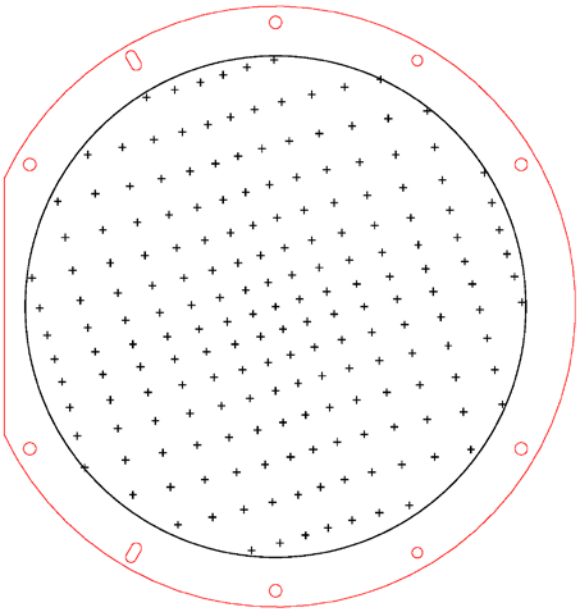

Figure 2 Fiducial Pattern. Diagram shows outline of the fiducial plate in red, with the pattern itself marked in black.

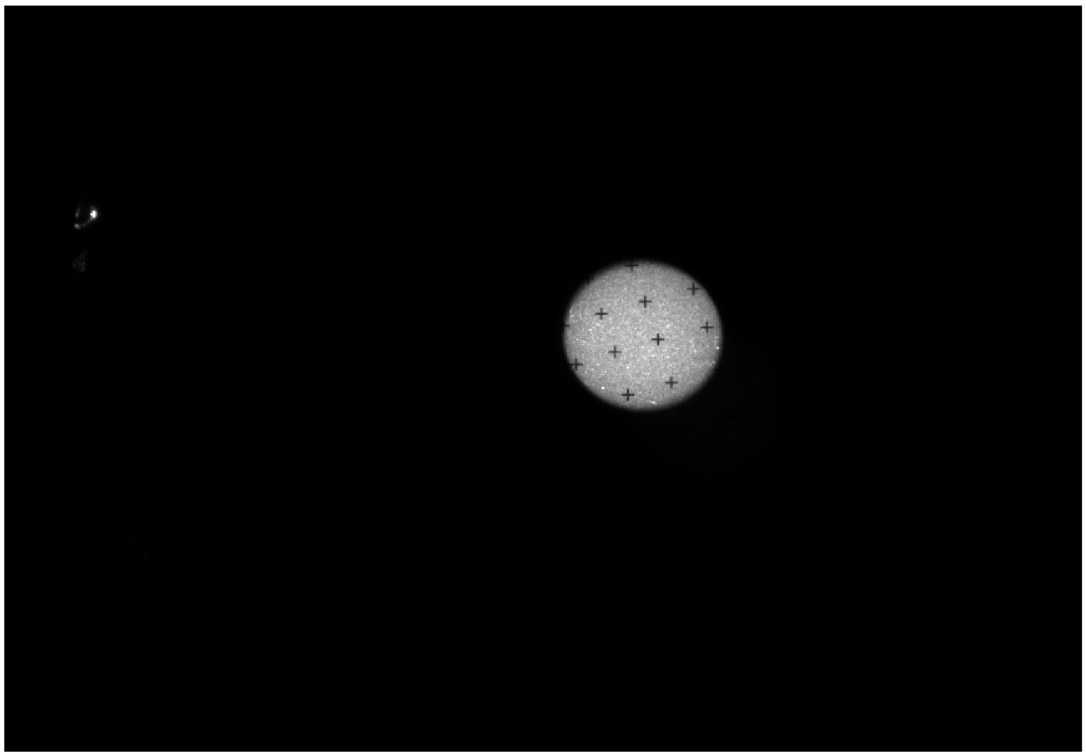

Figure 3 Example of a typical PYAS frame

2.4 PYAS Images

An example of the images seen by the PYAS is shown in Figure 3. Images are rectangular with a small circular region illuminated by the sun. Thanks to the band-pass filter the projected solar image is monochrome. Matching this, a monochrome camera was selected for the PYAS, and frames were treated as intensity images. The solar projection falls into a larger circular area defined by the fiducial screen. A diagram of the entire screen is shown in Figure 2. The screen is sized to span nearly the entire image along the short axis, but the location of the center of the screen along the long axis can vary depending on camera installation. Angle of the camera relative to the screen could be adjusted in this axis to get the screen to fall completely in the camera's FOV.

Within the solar image it is possible to see several cross-shaped fiducial markers. Since only a fraction of the fiducial screen is illuminated, the bulk of the fiducial markers in the pattern are unusable for registration. Ideally then, each fiducial would be completely self-identifying, so that locating and identifying a single fiducial would generate a correspondence pair between the fiducial plane and the image plane. With enough of these points a mapping from image to fiducial planes could be generated. Location and orientation of the fiducial plate was measured relative to the x-ray telescope, and this calibration data bridged the gap from knowledge of the sun's location on the fiducial screen to knowledge of the HEROES telescope's offset from target.

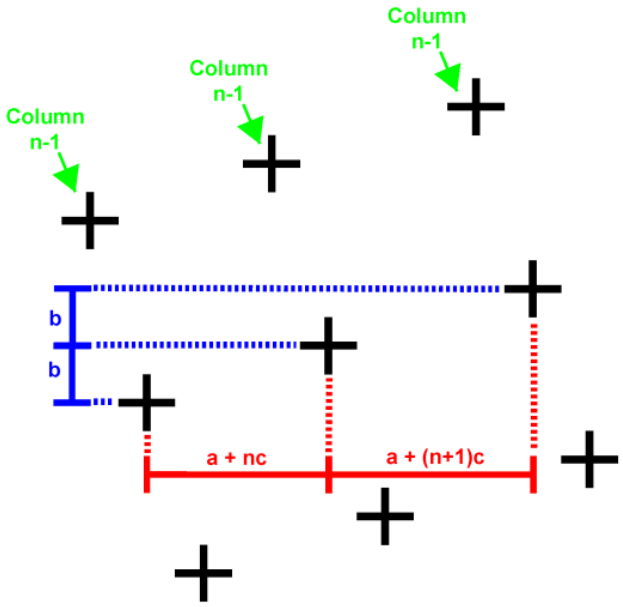

Figure 4 Fiducial Spacing. Fiducial spacing along a row is shown here. Subsequent entries along a row are spaced vertically by a fixed distance b. Horizontal spacing starts at distance a near the center of the screen and increases by c for each subsequent column

The fiducial screen was a matte black and white pattern screen printed on an aluminum plate. The pattern is laid out in such a way that markers can be identified completely if three or more adjacent non-collinear markers are visible. Marker identity was encoded in inter-marker distances, as illustrated in Figure 4. Starting at the center of the screen, fiducials in the same “row” are placed with increasing spacing "a + nc" in the x-axis, but fixed spacing "b" in the y-axis. The opposite is done for “columns.” Fiducials are often referred to by their “ID,” starting with (0,0) at the center of the screen and increasing or decreasing as they extend away from the center. Identifying fiducials depends on finding pairs which are spaced by the fixed spacing "b" in one axis, and measuring their spacing in the varied axis. This provides either row or column ID for both fiducials in the pair. To identify a fiducial it must be a member of both a row and a column pair.

In addition to the solar image and fiducial markers, a variety of undesirable features might be present in a PYAS image. In Figure 3, some of these features are visible. For example, light bleeding through a joint in the baffle is visible at the edge of the frame as a small bright spot. The baffles were comprised of several sections of painted cardboard tube, with adjacent sections taped together. Under the right conditions it was possible for light to leak through the tape and illuminate part of the fiducial plate. There are also bright spots on the solar image which correspond to scratches on the fiducial plate. Deep scratches would remove the paint and expose reflective bare metal, which would show up as much brighter than the rest of the screen. Examples of both baffle leaks and scratches are visible in Figure 3, with cropped versions below in Figure 5.

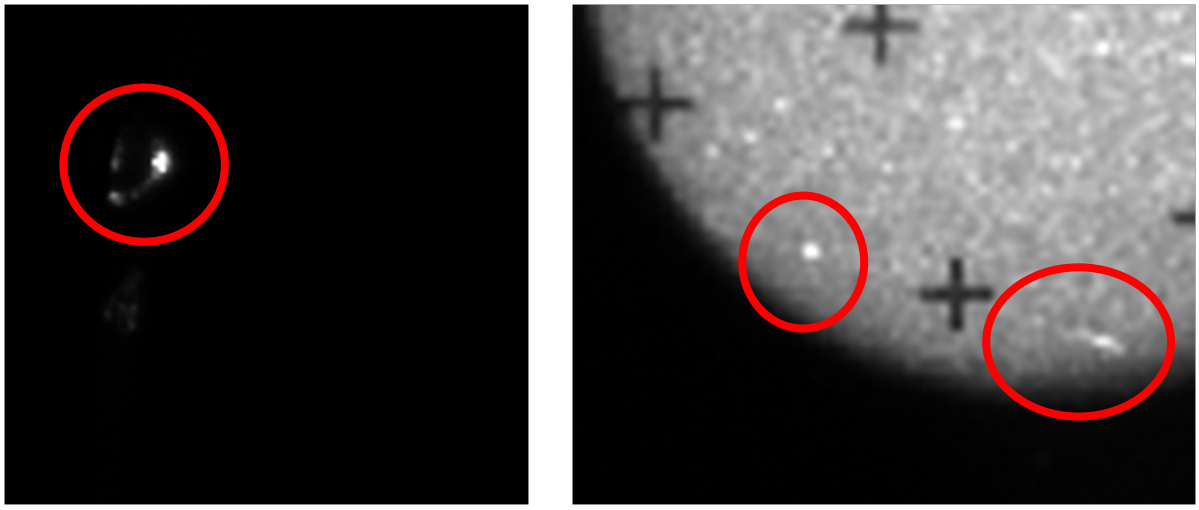

Figure 5 Examples of disturbances in PYAS image. (Left) Bright spot at the edge of the fiducial plate, probably due to light leaking through the baffle. (Right) Bright spots which are likely scratches on the fiducial plate.

2.5 PYAS Imager

The HEROES electronics of immediate interest to the PYAS would be the SAS computer stack and the camera. The camera was an Imperx IGV-B1310, which

communicated with the SAS computer through a gigabit ethernet connection. A summary of the camera specs is given in Table 2. The max frame rate was more than high enough to meet the PYAS cadence requirements, even when returning the full image rather than a smaller region of interest (ROI). The detector size of 966 pixels in the short axis, combined with 2.8° FOV, gave each image pixel an approximate width of 10.6 arcseconds. Given the minimum requirement for PYAS resolution of 20 arcseconds, the PYAS algorithm would need to refine position estimates to roughly 2 pixels. A target accuracy of 10 arcseconds would require accuracy to better than a pixel and would leave a sizeable margin against the requirement.

Camera Parameters	
Camera	Imperx IGV-B1310
Lens	MVL75L (75mm lens)
Image size	1296x966 pixels
Bit Depth/Color	12 bit monochrome (8 used)
Max Frame rate	26 FPS

Table 2: PYAS Camera parameters

The SAS computer stack performed all processing of PYAS images, storage of PYAS and RAS image, and all command and telemetry functions. The computer stack was assembled from off-the-shelf components conforming to the PC/104+ standard. There were five boards in the stack, only the single-board computer and solid-state storage were directly relevant to the PYAS algorithm. The specs for both are listed below.

SBC	
Model	Cool RoadRunner-945GSE
Processor	Intel Atom N270 Single-core 1.6 GHz
Ram	2 GB DDR2 533 MHz
Storage	
Type	Solid-State Drive
Capacity	256 GB (x2)
Max Sequential Write Speed	< 260 MB/s

Table 3 Summary of relevant SAS computer specs.

The single-board computer was powered by a 1.6 GHz single-core processor. This meant that the processing algorithm was time-sharing with command, telemetry and data storage functions. The clock rate was also low relative to other modern processors, which often operate at double that rate. The SBC was chosen for its wide range of operating temperatures and low cost, not necessarily for processing horsepower. Storage was chosen to be SSD, sized to be able to store PYAS data uncompressed for 8 hours of observation, and had a max write speed well above the desired 10 MB/s required to store full, uncompressed PYAS and RAS frames at 4 Hz.

2.6 PYAS Assumptions

The setup described above gives a baseline for what the PYAS image processing problem might have to handle. There are scenarios not described above, however, which could severely alter the scene observed by the PYAS camera. Accounting for every possible anomaly would very quickly push the algorithm's complexity to a point where it had no chance of meeting the cadence requirement. Instead some basic assumptions were made about how PYAS frames might vary during flight, with the goal of keeping the

PYAS problem tractable. These assumptions are listed in Table 4, and a discussion of the impact and verification of each will be given below.

	Assumption
1	There will be only one sun-like object present in each frame
2	Optics and camera are approximately parallel to the fiducial plate
2a	Solar image is circular rather than elliptical
2b	Projective affects from the camera orientation can be approximated with a similarity transform
2c	Clocking of fiducial plate relative to the camera is negligible
2d	Change in distance and clocking between fiducial plate and camera will be small
3	The projected solar image will not be under or overexposed

Table 4 PYAS Flight Assumptions

The first assumption was that there would only be one sun-like object in a given PYAS image. This was verified via design of the PYAS optical path. After installing baffles around the light path, and a cover box over the solar optics and camera, only light illuminating the fiducial screen would be light coming through the solar optics. The resulting image would then be free of bright objects resulting from reflections from the gondola body. Baffle leaks have been mentioned already as bright features that can be present in a frame, but they exist at a much smaller scale than the sun, making them easier to single out and avoid. This assumption was also verified via testing, since test images taken with a fully-assembled PYAS did not have any spurious sun-like objects. In terms of the image processing, this assumption says that there would be at most 1 large circular illuminated region in a PYAS frame. This assumption frees the algorithm from needing to handle multiple circles, and from having to distinguish between detected objects to determine which is the true sun.

The next assumption is that the fiducial plate will be approximately parallel to both the solar optics and the camera. By design the bulkheads housing the optics and plate should be parallel, and although the camera was mounted at an angle to the plate, this

angle was assumed to be small enough as to be negligible. Overall, projective effects in a PYAS image of the fiducial plate are assumed to be minimal beyond a similarity transform. This assumption was assumed to hold for optics and plate based on the mechanical design of the PYAS, as it was very unlikely for the two bulkheads to shift significantly relative to each other. Being able to assume this geometry vastly simplified the PYAS problem, making the sun circular rather than elliptical and fiducial markers fixed in orientation and spacing. Assumption 2 was verified during integration by analyzing PYAS frames. The solar image was circular, which indicated that the solar optics were indeed parallel to the fiducial plate, and fiducial pattern was easily identified, meaning the pattern was not rotated significantly relative to the camera.

The last assumption was that the projected solar image would be properly exposed. While the PYAS algorithm was designed to be robust to slight changes in brightness, it would run into problems if images became either very low contrast, or were saturated. Without this assumption an auto-exposure algorithm would have been needed to adjust camera settings. While on the ground, solar brightness changed as a function of cloud cover, requiring occasional adjustments to the exposure settings. On flight however, cloud cover ceased to be an issue, and these adjustments were required only once at the start of observation.

Given the design of the PYAS and the SAS as a whole, it is reasonable to make these assumptions about the scene, especially with regard to geometry. These assumptions allow much simpler approaches, which help to keep processing time down. However, without these assumptions the problem would still be tractable. The next section

considers methods that could have made the PYAS robust to the failure of these assumptions.

3 Related Work

3.1 Introduction

The problem here is framed as solely a computer vision problem, where the design of the SAS hardware, especially fiducial pattern and solar optics, is considered to be fixed. The larger problem is that of sun detection and aspect determination, for which there is an extensive history. These pointing problems are likely very similar to the problem faced by the larger SAS system, so it is important to examine how past missions have solved problems like the one faced by HEROES. Existing work in computer vision is also directly applicable, especially any solutions to similar shape-finding and image registration problems. Detecting the sun is at worst an ellipse-finding problem, and locating fiducials an arbitrary shape detection problem. After locating the sun and fiducial markers, image registration literature can be considered for mapping from camera coordinates to the coordinate system of the gondola for use by the gondola's pointing control system. This section addresses literature concerning both previous solar observatories and similar computer vision problems.

3.2 Sun Sensors

Sun sensing has applications across the space industry. First and foremost, any solar observatory requires some means for determining aspect relative to the sun. Sun sensing is also often part of a suite of sensors for pointing, even if the goal is not to point at the sun itself. Missions which use solar power may also incorporate sun sensors to determine the optimal way to point panels. The range of cadence, FOV, and accuracy requirements

for these applications varies quite widely, and as a result there are many approaches to sun detection.

3.2.1 Analog Sensors

One of the simplest sun sensing methods is the cosine sensor described in [2]. Cosine sensors are single photocells which generate an analog current based on the angle of incident light. Depending on how they're mounted, these sensors can have a very wide FOV, on the order of 160° cones. One of these sensors alone is not sufficient to provide angle knowledge in even a single axis, so generally a suite of these sensors are mounted at different angles on a single craft. For example, SDO employed 16 as part of its sensor suite [3], MAP had 12 [4], and TRMM used 8 [5]. Manufacturers list them as being capable of accuracies on the order of degrees, which makes them very coarse sun sensors.

Also in the realm of analog sensors would be quad-cell-based designs. These are composed of four photodetectors in a 2x2 grid, protected by an aperture of some kind. They are treated as differential sensors, so pitch and yaw angles are computed from the difference in current between two photodetectors along a given axis. Detectors of this style depend on careful calibration to account for sensitivity differences between detectors, and are adversely affected by temperature gradients across the sensor. An older example would be the Lockheed Intermediate Sun Sensor (LISS) [6], which offered a ±3° FOV, and measured accuracies down to the 7 arcseconds. A more recent design is described in [7], but the fundamental principle is the same. In a test flight this newer sensor boasted a ±60° FOV, but only a 0.15° accuracy.

3.2.2 Digital Sensors

Again, a basic digital sun sensor (DSS) is described in [2]. The style described therein restricts incident light with a slit shaped aperture, and then with a coded aperture over a series of large rectangular photodetectors. The coded aperture creates a digital signal with a binary arrangement of slits. For an n-bit digital sensor there would be n rows in the aperture, each with 2^{n-1} individual openings. As the sun sweeps across this FOV of the sensor, the main slit generates a line of light that sweeps across the coded aperture. The rows of openings in the coded aperture then block or admit light to the photodetectors below so that each detector generates one bit of the output angle.

Ideally the pattern would mimic a Gray code rather than straight binary, and at the lowest stage detectors can be spaced finely enough to use interpolation circuits rather than a simple on/off to further refine the reported angle. DSS sensors of this style have a fairly rich heritage, having flown on TRMM, SDO, and MAP for example. DSS sensors are able to provide angle about one spacecraft axis, so a pair would be required to provide both pitch and yaw. The Adcole version of these sensors advertises FOV of up to 128° for each axis, with accuracies down to a quarter degree.

Somewhere between a DSS like the one described above and the imaging sensors in the next section is the Solar Aspect System of the RHESSI spacecraft [8]. The RHESSI SAS employed three linear CCD sensors, each with 2048 elements. By imaging onto these sensors with a 1.55m focal length lens, the RHESSI SAS achieved a plate scale of 1.73 arcseconds/pixel, with each individual CCD having a FOV of about 1°.

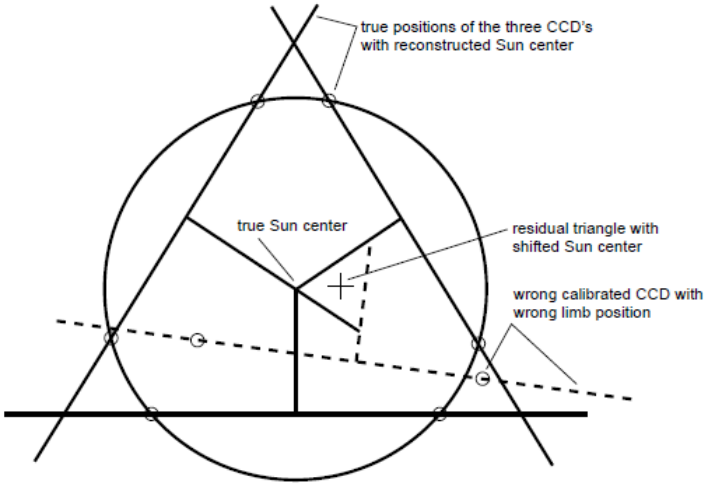

Figure 6 Effective layout of the RHESSI SAS CCD detectors [9]

As shown in Figure 6, these CCDs were each rotated 120° from each other. When the solar image was fully visible for a detector the SAS software could locate the locations of the solar limb and from these compute midpoint of the sun along that particular detector's chord through the solar disk. With the sun fully visible by two CCDs, the solar center could be estimated. Ideally the sun would be visible on all three. In that case, knowledge of midpoint of the sun on each CCD and the relative angle between CCDs would allow for a precise estimate of the spacecrafts orientation relative to the sun.

3.2.3 Imaging Sensors

Modern sensors are beginning to move toward dense sensor arrays, as opposed to setups built around a handful of large photodetectors. The resulting sensors move into the realm of computer vision, and away from analog approaches like cosine sensors or quadcells. Even compared to the problem faced by the HEROES SAS, the processing problems faced by these imaging sensors are very tightly constrained, making it feasible

to do large parts of the computation locally in an FPGA, rather than putting the burden on software.

One of the simpler imaging sensors is the Micro Digital Sun Sensor (μDSS) [10]. It is based around a 368 pixel square CMOs sensor mounted behind a pinhole aperture. The aperture was sized and located to generate an image of the sun roughly 10 pixels across. This projected image could be located coarsely by looking at winner-take-all projections of the image along its rows and columns, then refined to sub-pixel accuracy using multiple centroids in the ROI. In a lab setting the μDSS was determined to have a ±47° FOV, and was accurate down to 36 arcseconds.

A similar but more complicated design for an imaging sensor was implemented in [11]. This sensor, referred to as the Micro Sun Sensor, also uses an imaging sensor behind an aperture. The aperture in this case, however, has 21 separate 50μm pinholes, as opposed to the single 50μm pinhole of the μDSS. This provides robustness against particulate blocking the pinhole, at the cost of complicating the image processing problem. The apertures are arranged in three by six block of holes, with a column of three holes above the leftmost column of the block. To determine solar angle this sensor must now determine both spot location and spot ID. It appears that the ID of each aperture is somehow encoded in inter-aperture spacing along rows of apertures. Ultimately this design has been shown to be capable of better than 0.2° accuracy at the center of its field of view, with a full field of view spanning 120°.

3.2.4 Summary

There are a wide array of solutions to the problem of determining solar pitch and yaw, and the PYAS algorithm can stand to learn a great deal from existing systems. Some

statistics on the systems covered here are summarized in Table 3, with the PYAS requirements listed as well for reference.

	CSS	DSS	LISS	uSS	RHESSI SAS	µDSS	MSS	PYAS (flight)	PYAS (ground)
FOV	160°	128°	6°	120°	1.53°	94°	120°	2.79°	--
Accuracy (relative)	1°	.25°	7 arcsec	.15°	.4 arcsec	36 arcsec	.2°	20arcsec	15 arcsec
Cadence	cont	cont	cont	cont	128 Hz	10 Hz	2 Hz	1 Hz	4 Hz

Table 5 Summary of sun sensors FOV, accuracy, and cadence

Obviously not all the sensors covered here are applicable in the case of the PYAS. The CSS for example is not intended for fine pointing. In fact, many of the sensors covered here seem to have traded accuracy and fine plate scales for very wide FOV. The PYAS needed to tend the other way, and with a final design was very similar to the RHESSI SAS, it made sense to base the algorithm on the RHESSI algorithm. Although the exact details of the MSS aperture layout are not clear from the publication, the aperture spacing strategy described is not unlike the strategy used for spacing fiducial markers in the PYAS setup.

3.3 Existing Algorithms

The scene the PYAS observed was actually quite benign when compared to some in modern computer vision. Camera and target were coupled together mechanically, and illumination of the scene was very tightly controlled. The result is that the PYAS problem decomposed into very basic image processing tasks, each of which had a well-established history. The task of locating the sun is an ellipse finding problem, and can be simplified to the problem of finding a circle of known radius. The fiducial detection task is a

problem of detecting cross shapes, which falls under arbitrary shape detection. Detecting small geometric markers is another common machine vision problem. Finally the task of generating a mapping from pixel to gondola space is an image registration problem. This section will briefly review relevant literature for each of these tasks.

3.3.1 Circle or Ellipse Detection

Determining location of the sun in a frame can be considered an ellipse or circle-finding problem. There is a wide range of methods available for finding circles or ellipses in an image, many of which can even provide sub-pixel accuracy. In the case of the PYAS it can be assumed that the sun will take the form of a circle with known radius, which vastly simplifies detection compared to an ellipse. The problem of detecting ellipses is a common one in machine vision, and there are a wide variety of solutions, all of which apply to the case of circles as well. Among these are intensity-based methods like centroiding or template matching, most often used when the expected shape size is small. Edge-based methods like Hough transform or least-squares curve fitting are available in many flavors depending on the specifics of the problem. Many of these were considered for use in the PYAS, and all of those mentioned either offer sub-pixel resolution by default, or can be adjusted to do so.

3.3.1.1 Centroid

The simplest method for determining location of the solar image would be a centroid of pixel intensity. Excluding features like hot pixels and baffle leak, the PYAS image is a black frame with an illuminated circle. The centroid of pixels above an intensity threshold T would lie very close to the center of the projected solar image. Calling I(m,n)

the intensity of the pixel in row m and column n of the image, the centroid C of the binary image is computed as:

$$C = \left(\frac{\sum_P m \cdot I(m,n)}{\sum_P I(m,n)}, \frac{\sum_P n \cdot I(m,n)}{\sum_P I(M,n)}\right), P = \{(m,n) \mid I(m,n) > T\}$$

This should work well given that the sun has a symmetric brightness profile. Another more common method would be to use a binary image, where pixels take value 1 if they lie above the threshold, 0 if not:

$$C = \left(\frac{\sum_P m}{\sum_P 1}, \frac{\sum_P n}{\sum_P 1}\right), P = \{(m,n) \mid I(m,n) > T\}$$

Both of the imaging sensors in the previous chapter used this method to achieve sub-pixel accuracy [10],[11], and the same method has also been used in detection of circular fiducials where it is was favored for simplicity and accuracy [12]–[14]. Centroids can be sensitive to occlusions, irregular illumination, and sensor noise. Irregular illumination and noise can be remedied to a degree by using a binary image [13], but that still leaves the problem of occlusion. Any change to the shape of the illuminated region could change its centroid. The circles in most cases where a centroid was used were on the order of 10-20 pixels across, and were used in cases where partial occlusion of a circle was unlikely. In the case of the PYAS not only is the circle much larger (180 pixels across), it is also guaranteed to have occlusions in the form of fiducial markers. Additionally, unlike other methods, there is no guarantee that the centroid is the center of a circular shape. Other approaches like a curve fitting approach or Hough transform are tuned specifically to circles, and can provide additional information like radius and reliability of fit.

3.3.1.2 Template Matching

Although better suited to much smaller circles, template matching is another method that can be applied to circle detection. Template matching can be used to search for an arbitrary shape or sub-image. A template is generated which matches the sub-image to be searched for. At the simplest this template can be an ideal version of the sub-image itself. Alternatively the template could be replaced by some other design of matched filter, like the edge-based detection filters described in [15]. Given an image I(m,n) and a template T(k,l) of size K x L the correlation image R(m,n) is computed simply by:

$$R(m,n) = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} I(m+k,\ \ n+l) \cdot T(k,l)$$

which is effectively a cross-correlation of image I with template T. The response R is then searched for peaks in correlation. A peak in the correlation should correspond to the presence of the target shape or sub-image in the original image. This process is illustrated in Figure 7 for both a simple template and an optimized matched filter.

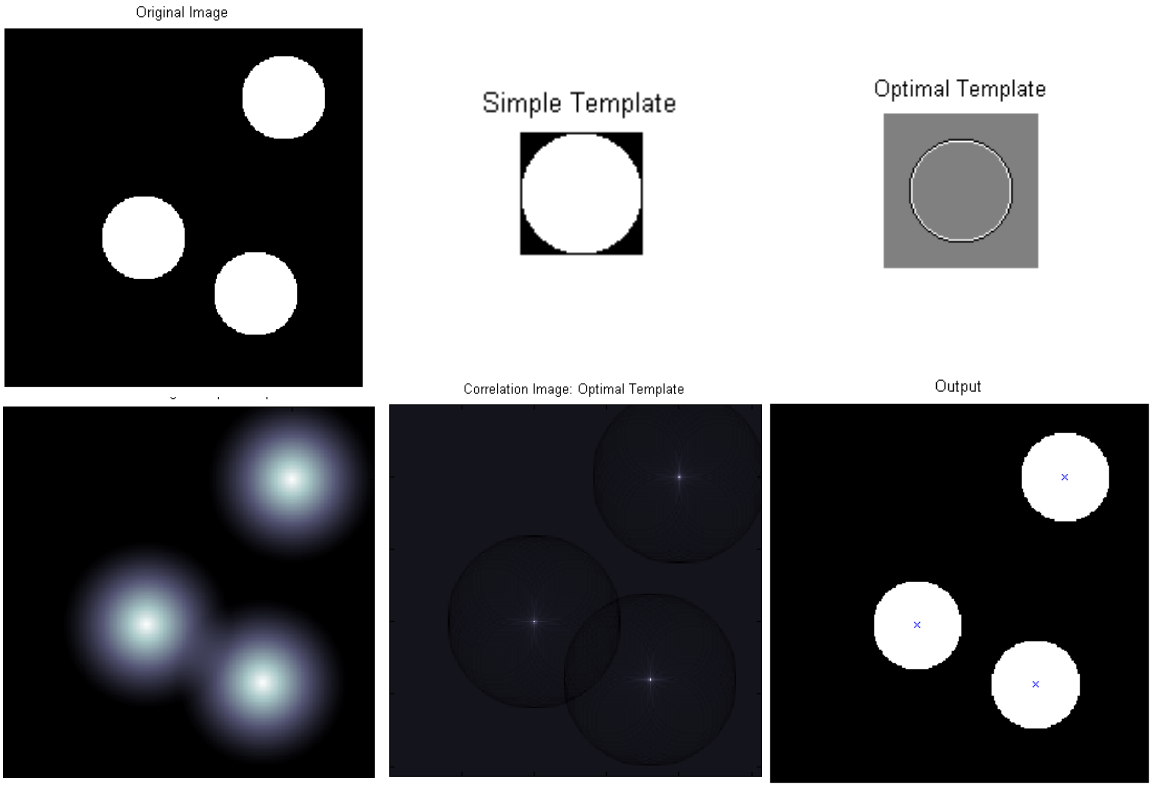

Figure 7 Illustration of Template matching. Top left is an example image with three circles. Top Center is an example of a simple template, top right is an optimized matching filter. Bottom left is the correlation from a simple template, bottom center is the correlation from the optimized filter, and bottom right shows the centers found with the matched filtering approach marked on the original screen.

Either template produces peaks in the correlation image that correspond to the locations of shapes in the original image. The optimized template, however, produces much sharper peaks. The diffuse response from using a simple template can make finding adjacent shapes difficult. In the bottom of the correlation for a simple template, responses for two circles are starting to blend together. This can complicate the task of discerning separate peaks. The edge-based filter on the other hand generates a very sharp response, making it easy to distinguish individual peaks, and to set a threshold on correlation value for detection.

This method takes the problem of shape detection and changes it to one of peak detection. For circles, using a template matching method requires that either multiple masks be used for different radii, or that the circle's radius is fixed. For ellipses, any possible orientation or set of ellipse radii must be used. By default this method will return a location only to pixel accuracy, and has to be refined by some other means to provide sub-pixel accuracy. While this method is not well suited to the sun-finding problem because of the sheer size of both the PYAS image and the required template, it is commonly used in the fiducial finding problem covered in the next section.

3.3.1.3 Hough Transform

Centroid and template matching methods worked entirely on raw image. The optimized matched filter described in the template matching section had an edge detection filter rolled into it, but otherwise these methods worked entirely on pixel intensity. The rest of the methods considered here will be edge-based. The first is the family of Hough Transform techniques for circle detection. Hough Transform is a classic approach to finding lines and circles in images, and is summarized in [16]. The standard Hough transform proceeds as follows. In the case of a circle with known radius, a 2D accumulator array is formed with axes corresponding to coordinates of the center (a,b) of possible circles and with all initial values set to zero. An edge detector is applied to the raw input image, and the Hough transform iterates over the resulting edge points. For each edge pixel, all the circles in the parameter space that could have that pixel as an edge point have their value incremented by 1. For a fixed radius circle this amounts to drawing a circle in the parameter plane for each edge point in the image. This process is illustrated in Figure 8. After all edge points have been visited, the space is searched for

either its max value or a set of maxima, depending on if the goal is to find a single circle or multiple. Ideally, each local maximum in the accumulator corresponds to a circle in the image. Adding a third dimension to the accumulator allows for searching for circles of arbitrary radius. Over multiple radii, a single point sweeps a cone in the parameter volume.

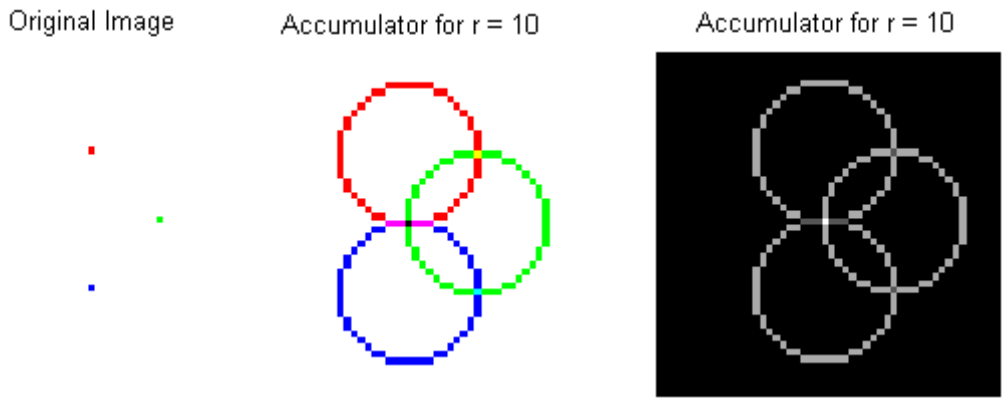

Figure 8 Hough Transform for circles. At left is a sample image with three edge pixels on a circle of radius 10. Center shows the accumulator for r=10, with the contribution from each edge pixel color-coded. Right shows the same accumulator in monochrome. The white pixel at the center is the peak in value, and corresponds to the likely center of the circle.

One of the major issues with the Hough transform is the accumulator matrix. The spacing of accumulator cells dictates what range of parameters is considered, and therefore how precise the transform can be. To find circles of fixed radius centered anywhere in the image frame down to pixel accuracy, the Hough accumulator would need to be at least equal in size to the original image. The size of the accumulator matrix increases as the square of any increase in accuracy, meaning that sub-pixel accuracy will quickly eat into any memory budget. Broadening the search to circles of unknown radius or to ellipses requires either one or three extra dimensions, quickly making the

accumulator unmanageably large. In spite of this memory requirement, Hough is very attractive for simple shape detection, and as a result there is a wide array of methods for reducing the memory requirements [17].

3.3.1.4 Curve-Fitting

The last family of methods to be considered here is curve fitting. As with Hough, this method starts with passing the image through an edge detection filter, and then working only with edge pixels. Once a list of edge points is available, the task is to somehow generate the parameters of a single ellipse or circle that match the data. In the case of ellipse-fitting, solutions to this problem generally reduce to solving a non-linear least squares problem. If the data is assumed to be well fit by a circle there are more options available, including linear least squares and more ad-hoc solutions.

Given a set of edge pixels, the ellipse curve fitting problem requires a cost to minimize. One of the simpler costs takes advantage of the fact that ellipses are conic sections. This leads to the "algebraic distance," which minimizes a cost based on the formula for a conic section.

$$C_A(\boldsymbol{a}) = \sum_{p=1}^{P} a_1 {m_p}^2 + a_2 m_p n_p + a_3 {n_p}^2 + a_4 m_p + a_5 n_p + a_6$$

Where a_i specify the parameters of the fit and there are P points (m,n). Other parameterizations of can be used a conic section by simply replacing the algebraic expression. This is a convenient cost algebraically, and leads to a fitting problem that can be solved as a constrained linear least squares problem [18]. The actual geometric meaning of this cost, however, is unclear.

Alternatively the cost can be set up in terms of the square of the residual between the data and the nearest point on the fitted curve, aptly called the "geometric distance." The expression for ellipses is rather complicated owing to their being a more complicated shape to parameterize, but the expression for circles is straightforward:

$$C_G(a,b,r) = \sum_{p=1}^{P} (\sqrt{(m_p - a)^2 + (n_p - b)^2} - r)^2$$

Where (a,b) is the center of the circle, and r is its radius. This is a far more intuitive cost, and the geometric meaning is immediately clear. The downside is that it does create a more complicated non-linear problem and generally requires an iterative approach like Gauss-Newton [19]. In many cases the solution to the algebraic cost is used as the initial value for one of these iterative methods.

When treating circles specifically, as opposed to fitting to an ellipse, the problem is simpler. Least squares fitting that works on ellipses can generally be adapted for circles, simplifying the problem considerably [19]. Simplifying to circle-finding as opposed to ellipse-finding also opens up a wealth of other methods. For example with a change of variables the non-linear least squares problem of minimizing geometric distance can be reworked into a linear least squares problem, and that this modified problem was shown in experiments to be more robust than its non-linear counterpart [20]. Beyond the least squares approach, there are methods that focus on determining the center of the circle described by a set of points, then deriving radius as the average of the distance from each point to this center. For example, the approach of "Average Intersection" uses triplets of non-co-linear points and treats them as endpoints of chords through a circle [21]. The bisectors of a circle's chords intersect at its center, so the average of these intersections

should provide a good estimate of the circle center. This method was shown to be sensitive to noise on the edge and to situations where chords were too short, i.e. points used to calculate an individual bisector were too close together.

3.3.1.5 Summary

The past section has touched on several methods for fitting a circle or ellipse to data. Centroid estimation provides reasonably good results for small shapes, but the effects of noise can become problematic for larger shapes. Template matching works well for small circles of a known size, but quickly becomes prohibitively expensive otherwise, requiring large or multiple templates, or both. Hough transforms biggest limitation is memory required for its accumulator, which is minimized if the target is a circle of fixed size. Least squares curve fitting opens a wealth of possibilities, although the best performing methods seem to be non-linear problems requiring iterative solvers.

Most of the literature is concerned with locating a shape, or with finding a “good” fit to a set of points. What is a good fit to a set of arbitrary points is a very qualitative assessment, but perhaps not relevant in problems of metrology. The only quantitative assessment is often given in terms of memory and processing power required, or details on how a specific iterative solver performed. A comparison of performance for measurements like the one needed in the PYAS problem would require extensive testing on images where locations and sizes of circles were known very precisely.

3.3.2 Fiducial Mark Detection

The task of finding PYAS fiducial markers is again a shape finding problem. Unlike the sun-finding task where it could be assumed there was only one circle in the frame,

here there are guaranteed to be multiple fiducial markers. As with circle-finding, the task of locating small geometric marks has been present in machine vision for a long time. Although there is no direct requirement on the accuracy of fiducial marker locations, the goal for this stage was to once again to resolve to sub-pixel levels. This is a fairly common requirement for fiducial markers, so most of the methods considered here aim to resolve fiducial location to better than pixel level.

When designing the fiducials for the PYAS, one of the drivers was that the fiducials be as small as possible. By nature of the setup fiducials would be obscuring part of the sun, so smaller fiducials would provide a smaller disturbance. Therefore fiducials for the PYAS need to be small, easily printed, and locatable to sub-pixel levels. These requirements are very similar to those found in vision-controlled manufacturing environments like pick-and-place circuit population and lens cutting [18][19]. Regardless of the marker shape, the method most often employed for locating these kinds of fiducials seems to be template matching. Common methods to further refine the locations from a template matching approach include centroiding, intensity interpolation, correlation interpolation, and curve fitting.

3.3.2.1 Reseau Marks

One of the earliest examples of fiducial markers (and more specifically cross-shaped fiducials) is the reseau. To compensate for the motion of film relative to camera optics, film cameras used in aerial and scientific photography can employ a glass plate marked with a grid of dark crosses. After the film is scanned, these marks are located precisely and used to generate point correspondences between individual stills or video frames. A method for accurately locating reseau centers is described in [22]. Images containing

reseau marks were hand-picked, meaning that the method described was useful only for refining location of a mark, not detecting it. This study was also carried out on real aerial photography data, meaning each image could potentially have a complex background. The reseau mark being analyzed was magnified to the point where each arm was 5 pixels wide. In order to determine the location of a cross to sub-pixel accuracy, a parabolic intensity model was assumed across each arm of the cross. Zero crossing of the raw intensity derivative was used to determine location of the mark, without any low-pass filtering or image conditioning. By averaging these estimates for multiple locations along the arm, location was determined to a tenth of a pixel.

3.3.2.2 Centroid

In the case of using a centroid, the approach is identical to the one described for circle-finding. Computing centroid of a binary image of each individual fiducial seems to be the most popular method, with many sources considering the centroid accurate enough on its own [12]–[14]. In the case of [14], the centroid of a bulls-eye-shaped fiducial was further refined by fitting a circle to the edge of the fiducial region. Although there isn't agreement on the specifics of the shape, users of a centroid method agree that a filled convex shape provides best results, be it circular or diamond shaped. Using a threshold makes the centroid sensitive to pixels at the perimeter of the shape: those barely above or below the threshold introduce a disturbance depending on which side of the threshold they happen to fall for a given frame. Convex, filled shapes have fewer perimeter pixels for a given area, making them less sensitive to this phenomenon than shapes like crosses.

Centroid estimation can also be used in a correlation image as opposed to the original intensity image. Centroid estimation has been considered in many fields as a method for

peak-estimation, making it applicable here. As mentioned previously, template-matching converts the shape finding problem to one of peak finding. The location of the peak can be refined many ways, and one of these is to take a centroid. With a sharp enough spike in correlation the problem becomes very similar to light stripe detection in 1 dimension, or to detection of point sources in 2 dimensions as would happen in a star tracker or a system using LEDs as active fiducials [23], [24]. Tests on centroid estimation in this context show it to be able to narrow down peak position to fractions of a pixel, but to be sensitive to window size, noise, and threshold [24]–[26].

3.3.2.3 Intensity Interpolation

An alternative approach is to up-sample both image and template. The naïve approach would be to up-sample the entire image, and the entire template, and then perform template matching on these two interpolated images. Accuracy in the estimate of a fiducials location should increase proportionally with the up-sampling factor. This approach is analyzed in great detail in [27], where an algorithm is devised that reduces the cost both in required computation and storage. The idea is to up-sample the image in stages, rather than all at once, and to consider a smaller ROI (in terms of pixels of the original frame) at each scale. By comparison to most other methods this approach is extremely accurate, showing errors on the order of .005 pixels. Correlation interpolation was an order of magnitude worse on the same data, and centroid methods analyzed in [13] have been shown to only be accurate down to 1/10 a pixel. This accuracy comes at the price of high computational cost. Even with the measures taken to reduce computation time, [27] concedes that correlation interpolation is faster.

3.3.2.4 Correlation Interpolation

The last method considered here is correlation interpolation. Most commonly, coarse fiducial finding is handled with template matching. The template matching method returns a correlation image whose peaks are assumed to correspond to locations of fiducials in the image. This gives location to the pixel level quite reliably. The previous methods then proceeded to define an ROI based on this coarse knowledge and further refine it. The approach taken in [28], and also considered in [27], is to assume the neighborhood around the correlation peak can be approximated with a paraboloid surface. A least squares fit to the neighborhood yields the parameters for a surface, and the parameters can be used to compute the maximum of the surface. The implementation in [28] used a simple linear least squares fit to the 9-point neighborhood centered at the peak correlation value. This method was shown to be accurate down to 1/16 of a pixel for a wide array of image features.

3.3.2.5 Summary

This is not a complete review of literature surrounding fiducial markers. There are wide array of fiducials in use in fields like augmented reality where markers carry data payloads, are larger, and exhibit a great deal more structure than those used in the PYAS. These styles of fiducials are often required to be locatable and identifiable even after projective transformation, scale change, and variation in illumination. The review here limited itself to fiducials used in much more controlled machine vision environments, and a summary of methods considered is given in Table 6.

Refinement	Accuracy (pixels)	Fiducial Shape
Averaged center of arm	1/10	Cross-shaped
None	1	
Intensity Interpolation	1/200	
Correlation Interpolation	1/16	
Centroid (Intensity)	1/10	Circular
Centroid (Correlation)	1/10	

Table 6 Summary of fiducial finding methods covered

A template matching approach followed by a local centroid appears to be the most popular fiducial detection method. This method is easiest when applied to small, filled convex shapes like circles. Circles seem to be the most popular due to their being easy to locate with a simple centroid and their being rotationally symmetric, something which cross marks clearly are not. Because of their poor performance under centroiding, cross shaped fiducials require either something like the method used on reseaus, or an interpolation of either intensity or correlation to refine their location to sub-pixel. Centroid of correlation information could in theory achieve comparable performance to correlation of intensity, and would offer a method that was shape-independent.

3.3.3 Image Registration

The last problem to be tackled with the PYAS is one of image registration. Camera position in the PYAS setup was somewhat arbitrary, and certainly not calibrated. All calibration of the PYAS was done to determine location and relative position of the fiducial screen pattern relative to the other HEROES systems. The last section considered how fiducials on the screen could be located, and the ad-hoc methods developed for identifying them will be described in the next chapter. Once identified, each fiducial in the PYAS image offered a single point correspondence between the image plane and the plane of the fiducial screen. At the worst then this should be a problem of computing a

projective transformation from the image plane to the screen plane, which is addressed in [29].

When a plane image like the fiducial plate is projected onto the sensor of a camera, it undergoes a transformation. In general this projection can potentially take one of several forms. The simplest would be an isometry, where the mapping from the image plane to the screen plane would amount to a simple rotation and translation. Determining a transformation of this type would only require two point correspondences. The same is true of a similarity transform, which would also include the possibility of a change in scale. Generalizing to an affine transform would mean this scale factor could vary along different axes. Effectively it is an arbitrary linear transform in x and y, along with a translation. Such a transformation would require three point correspondences to compute, and also require that the three points are not collinear. Finally, the worst case would be where the mapping is projective. Such a mapping would distort the image of the screen to the point where the only invariance between the two planes is that straight lines remain straight. A mapping of this form would require four point correspondence pairs, where no three are collinear.

Regardless of the model for the screen to image mapping, methods for computing the mapping are largely the same. In fact, the problem of finding linear mapping from one plane to another is very similar to the problem of estimating the parameters of a conic section. Again, [29] considers many of these methods in detail. As when considering conics, the problem can be framed in terms of multiple costs. The simplest is a cost based on “algebraic distance” which leads to a linear least squares problem, but has an unclear meaning geometrically. Multiple “geometric distances” are also available, depending on

the nature of the two planes being compared. In the case of a known calibration image with a noisy observation, the cost can simply consider the residual between the point mapped from the observation plane compared to its partner in the calibration plane. As with geometric distances in conic fitting, minimizing a cost based on geometric distance generally requires an iterative approach.

3.4 Summary

For the PYAS, it was assumed that projective transformation of the fiducial screen would be minimal, and that the sun would therefore be present in the image as a circle. This circle would likely have a fixed radius for the duration of the flight. It was assumed that fiducial markers would introduce error in a centroid by occluding parts of the disk, and Hough transform methods were shown in experiments to have an excessive computational cost, especially when applied to the entire frame. This leaves curve fitting solutions. The “Average Intersection” method was of interest because of its similarity to the strategy used in the RHESSI SAS [21][9]. This method is a desirable choice for the sun finding algorithm as it has been proven to work on a previous spacecraft with a similar sensor and plate scale. The Average Intersection approach was shown to have some weaknesses in the case of arbitrary edge points, but these are mitigated by controlling how edge points are selected for comparison. The details of the method used in the PYAS will be covered in the next section.

For fiducial marker detection, choices were once again narrowed down by the style of marker and available computer resources. Again, thanks to the tightly controlled geometry of the PYAS problem, concerns about scale factor and rotation distorting

fiducial marks were mitigated. To determine coarse location of fiducials, a template-matching method was used. An edge-based template was generated following [15], as this made the peak pixels in the correlation very sharp and easier to identify. To determine fine location, centroid estimation in intensity was shown to react poorly to non-convex fiducial shapes like those used in the PYAS. Intensity interpolation on the other hand required more processing power than was available. The method used for locating reseaus was ruled out because markers were narrower and noisier than the reseau marks in [12], with arms of half the width. Correlation interpolation with a linear least squares approach would have been ideal, but there were issues getting a well-conditioned problem. Ultimately the PYAS algorithm used a centroid in the correlation image to refine the fiducial location.

Finally in the case of the image registration problem, assumptions were made early on that because of mechanical design of the PYAS, it will be constrained to a similarity transform. Furthermore there is very little rotation between the fiducial plate and the camera. This means that the problem of finding the mapping between sensor and plate amounts to determining the scale and offset between the two coordinate systems, requiring only three parameters to properly characterize the mapping. This problem was solved in the PYAS with a linear least-squares approach.

4 PYAS Algorithm

4.1 Algorithm Structure

As mentioned in the introduction, the problem faced by the PYAS algorithm was to determine orientation of the sun relative to the HEROES payload. The input was an image of the sun focused onto a plate marked with a fiducial pattern. At a top level the algorithm can be broken into a few simpler problems. The solution to each individual problem can be developed separately, as long as it delivers the proper data to the next stage. This list of stages is detailed in Table 7, along with their expected input and output. Also, where relevant, the coordinate space is listed. The ultimate goal is to determine solar center in gondola coordinates, or at least in a space which can be easily mapped to the same space as the gondola controller's coordinate system.

	Stage	**Input (space)**	**Output (space)**
1	Determine sun center (pixel)	Raw PYAS frame	Sun center (pixels) Solar ROI
2	Locate visible fiducials (pixel)	Solar ROI	Fiducials (pixel)
3	Identify visible fiducials markers	Fiducials (pixel)	Fiducials (pixel) Fiducials (screen)
4	Transform sun center (gondola)	Fiducials (pixel) Fiducials (screen) Sun center (pixel)	Sun center (screen)

Table 7 PYAS Algorithm Stages

Before advancing from one stage to the next, several checks are performed on the data products generated up to that point. These checks ideally remove any unusable frames before they can report invalid data to other gondola systems. Upon tripping an error the SAS reported an error to the ground, and the PYAS would halt all processing and wait for the next frame to arrive.

4.2 Locating the Sun

4.2.1 Assumptions and Requirements

This stage takes in the raw PYAS frame and must determine both the location of the solar center, and the location of a ROI containing the illuminated patch of the fiducial screen. In order to meet the accuracy and FOV requirements for the PYAS, raw images from the camera needed to be very large, 1296x966 pixels. Determination of the solar center is expected to be down to 20 arcseconds on flight. Translated to pixels, this equates to knowledge of sun center to within ± 1 pixel. Ideally this knowledge would do better than the requirement, so this stage should really be able to determine the sun's location to sub-pixel accuracy.

As for the solar ROI, the sun's diameter is a predictable value given the time of year and optical setup, and is roughly 0.5° across in the sky. With the given plate-scale of the PYAS working out to about 10.6 arcseconds per pixel, we know the sun will be around 180 pixels across. In addition it would be useful if there was some assurance that the sun will still be in the ROI on the following frame, even if it is no longer centered. Past HERO flights never saw slew rates higher than 0.25°/second, which works out to a ±22 pixel slew per frame, bringing the ROI minimum size up to 224 pixels across.

4.2.2 Motivation

The combination of image size, desired cadence, and limits on processing power on the target hardware ruled out traditional methods like the Hough transform or some kind of template matching. In order to avoid operations on the entire PYAS image, sun-finding was based heavily on the SAS of the RHESSI spacecraft [9]. Recall that the RHESSI

SAS projected the sun onto three linear CCDs, each rotated 120° from the others. The location of the solar limb was found on each CCD, and the results for each were averaged to provide an estimate of the sun center. A similar tack was taken with the HEROES SAS. In addition to being less computationally intensive, this also offered the added confidence of being a proven method for generating sub arcsecond attitude at a high rate.

4.2.3 Ideal Case

Instead of three linear CCDs, the HEROES PYAS has an array. An approach very similar to the RHESSI SAS' can be taken if individual rows and columns of the PYAS detector are treated as independent linear CCDs. Ideally with just one row and one column, the center of the sun could be determined in the same way it was on RHESSI. The more robust approach taken with the PYAS was to take multiple rows and columns, locate the limb crossings in each, and average all of these to arrive at an estimate of the solar center. Rows and columns of the image could be treated identically, and will be referred to as chords, since ideally each will be a geometric chord through the circle defined by the solar limb.

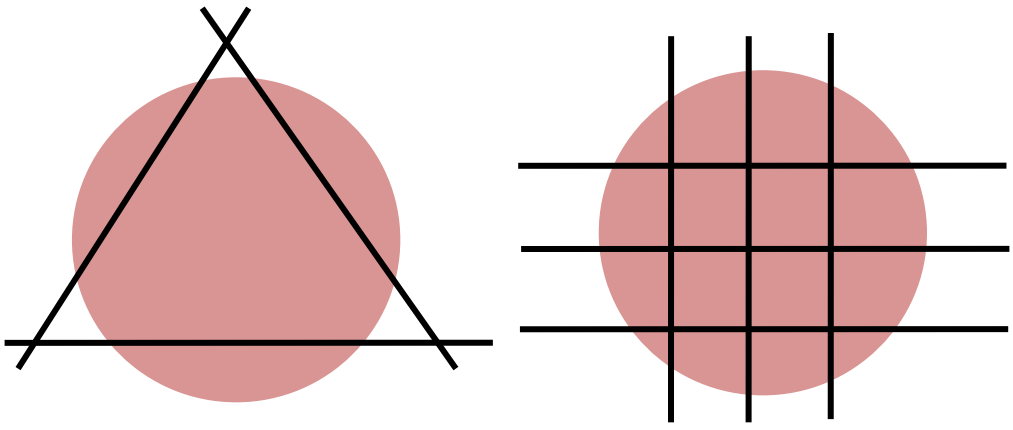

Figure 9 Effective chord placement in RHESSI (left) versus HEROES (right)

Potential limb crossings were determined by searching each chord for pixels immediately above a brightness threshold. Ideally there would be two of these per valid chord: one transitioning onto the limb, and one transitioning back down from the solar disk. If slope data at each point is included, the rules for a valid chord are simple and listed in Table 8.

Conditions for a valid chord (ideal)
Exactly two crossings are present
First crossing has positive slope (transitioning onto limb)
Second crossing has negative slope (transitioning off limb)

Table 8 Ideal criteria for a valid chord through the sun

Ideally all chords would either meet these criteria, or have no crossings at all. A reasonable policy then would be to simply reject any chord which did not meet the criteria in Table 8. The two threshold crossings are considered to be approximate limb crossing locations. Here the algorithm can proceed as RHESSI did. For each of these crossings, a local linear fit is generated. This is used in an attempt to refine the threshold crossing to the exact point at which the solar limb crossed the brightness threshold. Since the solar limb has a very stable brightness profile, points of identical brightness on the limb are spaced the same distance from the center of the sun. By refining the edge location in this way, even the naïve case of having two chords would still produce a valid center location to sub-pixel accuracy.

This does not address how to place these chords: how to pick which rows and columns of the image to analyze. Ideally all chords are placed on the sun, which is possible if we have prior knowledge about the sun's location. Otherwise chords need to be placed evenly across the entire image. To take advantage of any prior knowledge there

were two states. If the previous frame had determined a valid solar solution, then the previous ROI location could be recycled. In this case chords were placed evenly through the ROI. If the most recent frame had not produced a valid solution, chords were placed evenly across the entire image.

4.2.4 Edge Cases and Disturbances

This naïve approach needs some refinement. There were a variety of features which did not meet this model, foremost among these fiducial marks. Unlike RHESSI, the HEROES PYAS had to contend with its own fiducial markers. As these were black marks against a white background produced spurious crossings whenever they crossed a chord. Because of the rotated fiducial grid, it was actually fairly unlikely for a chord to be free of fiducial marks. The solution taken here was to try to set the threshold below the brightness of a fiducial marker, but also to reject any pairs of threshold crossings with slopes and spacing that could correspond to a fiducial.

In addition to fiducial markers, there were other non-ideal features which cropped up in both tests and during flight. Although the optical path was enclosed by baffles, stray light was still present due to bleed through joints in the baffling. These were visible as bright patches around the edges of the fiducial screen. As much as possible these were mitigated in hardware, by sealing the baffle joints to light.

Hot pixels and scratches on the screen manifested the same way, as saturated features in the image. Obviously the hot pixels were extremely small features, but while scratches were generally of very small width (1 to 2 pixels), they were often significantly longer. The threshold for what brightness value was considered solar limb would ideally be set as a percentage of brightness of the solar disk. Saturated pixels would skew this value to the

high side, so a simple maximum couldn't be used to determine solar disk brightness. Instead a histogram-based method was used to determine brightness of the disk: the "max" was set to be the cutoff for the 99th percentile, throwing away the last 1% of pixel values as outliers. This same value was substituted for the image max in fiducial detection as well. As far as edge detection went, hot pixels are easily avoided by neglecting any crossing pair with two short a distance between them. As for scratches, these were only visible when illuminated. Since the only appear on the solar disk in regions with bright backgrounds, they don't affect detection here.

4.2.5 Pseudocode and Failure Modes

The algorithm for sun-finding can be summarized by the pseudo-code below. This code requires just the current frame and the previous ROI location as inputs. It also needs a threshold, K, for defining limb location as a function of total brightness of the sun, and a number of chords to use per axis, N.

```
1.  Input: past_ROI_location, current_frame, K, N
2.  robust_max = 99th percentile of pixel values of current_frame.
3.  Robust_min = 1st percentile of pixel values of current_frame
4.  threshold = K% of robust_max
5.  if past_frame produced valid solution
        chords = N rows and N columns evenly across past_ROI_location in current_frame
    else
        chords = N rows and N columns spaced evenly across entire current_frame
6.  for each chord in chords
        a.  for each pixel in chord
                if current_pixel > threshold AND past_pixel < threshold
                    edges <- current_pixel_index
                if current_pixel < threshold AND past_pixel > threshold
                    edges <- -1*past_pixel_index
        b.  for each edge in edges
                if |current_edge| – |past_edge| <  fiducial_length
                    remove current_edge and past_edge from edges
        c.  if number of elements in edges is 2 AND first_edge > 0 AND last_edge < 0
            for each edge
                nbhd = pixels before and after pixel at edge
                line = fit line to neighborhood
                limb = index where line(index) equals threshold
                if chord was row
                    row_limbs <- limb
                else chord was column
```

```
                    col_limbs <- limb

7.  sun_center =  [(average of col_limbs) (average of row_limbs)]
8.  sun_error = [(std dev of col_limbs) (std dev of row_limbs)]
9.  ROI = 255 pixel square centered around sun_center
10. Output: sun_center, sun_error, ROI, robust_max, robust_min
```

Outputs of this stage are the center of the sun and a ROI which should contain the patch of frame illuminated by the sun. It also returns the 1st and 99th percentile cutoffs for pixel brightness (`robust_min` and `robust_max`), and value called `sun_error`. The percentiles are used for determining dynamic range of the image, while `sun_error` gives an estimate of the spread of chord midpoint estimates in each axis. These values were compared to a threshold so that cases where chords were too spread out to correspond to a valid solution could be rejected. A summary of the error checks after this stage is given in Table 7.

Error Case	**Explanation**
`robust_max – robust_min < range_threshold`	Dynamic range too low

`size( col_limbs ) < 2`
`OR`
`size( row_limbs ) < 2`
	Too few limbs to find a valid solution

`sun_error < error_limit`	Chord midpoints are too spread out

Table 9 Error checks after the sun-finding stage of the PYAS algorithm

If any of these errors are found to have occurred, the algorithm fails at this stage and waits for the next frame. Each is considered an indicator that further processing would only generate an invalid solution. If none of these cases are realized, then processing can continue with fiducial detection.

4.3 Locating Fiducials

4.3.1 Assumptions and Requirements

The ROI determined in the previous stage should contain the only illuminated fiducials in the PYAS frame. The goal in this stage is to locate fiducial markers in the ROI, again down to a sub-pixel level. As mentioned in the introduction, fiducials were cross shaped markers with arms 12 mils wide and 90 mils long tip to tip. This equates to 2 pixels across and 15 pixels long. They are spaced unevenly and identified based on their neighbors. In order for fiducial identification to be successful, three non-collinear fiducials need to be located, one pair in the same row to determine column IDs, and vice versa for rows.

4.3.2 Motivation

Several potential methods were entertained for locating fiducials, but ultimately the simplest was deemed to be a form of template matching. While other considered methods might have been significantly faster, template matching proved to be extremely effective without requiring complicated logic and error checking that more ad-hoc methods would have needed. Thanks to the reduced size of the solar ROI and the small size of fiducial markers, convolution with a mask has a low enough computation time to be viable.. An edge-based template based on the work in [15] proved to provide extremely strong responses to fiducials, and to be very easy to refine down to sub-pixel level.

4.3.3 Ideal Case

In the ideal case the edge-based template matching algorithm mentioned above worked basically out of the box. The image could be convolved with a pre-generated

matched filter to generate a response image. Setting a threshold on response and searching for local maxima above that threshold was sufficient to find nearly all fully-illuminated fiducials, and even some partially-illuminated marks. A thresholded centroid of the response image around each local max refined the location of the peak response, and therefore of the fiducial, to better than pixel level.

4.3.4 Edge Cases and Disturbances

As in sun-finding stage, there are a few edge cases here that can hinder determination of fiducial locations. There are also several threshold values, which need to be set in an intelligent way. Thankfully, however, some disturbances cease to be a concern. Baffle leaks as well are not an issue. The matched filter really only generated spurious detections for sharp edges, and baffle leaks were too dim and dispersed to generate a strong enough response. On the other hand hot pixels and scratches proved to have much more impact.

Scratches, and occasionally hot pixels, were often able to produce responses to the fiducial template that were large enough to register as fiducial locations. This seemed to be a result of their simply having a significantly higher brightness value than their neighbors. To avoid this complication, any point above the 99th percentile was set to the 99% value. This was very effective at removing undesirable peaks in the image while affecting a small number of pixels overall. With these points removed, thresholds for detection could be tuned to produce reliable detection with very few false-positives. Thresholds were set in terms of standard deviations above the mean pixel brightness value, which made them fairly robust to changing intensity of the solar image. To further protect against false-positives, if multiple detections occurred within a fiducial length of

each other, only the highest of these responses was kept as a valid response. The rest were discarded.

4.3.5 Pseudocode

The following pseudocode describes this stage in the algorithm. The inputs to this section of code are the ROI from the previous stage, the robust max computed in the previous stage, and a few parameters. The `K1` and `K2` parameters specify how many standard deviations above the mean to set thresholds for detection and centroiding respectively. The `NBHD_SIZE` variable sets the size of the region around each fiducial to use when computing a centroid.

```
1.  Input: ROI, robust_max, K1, K2, NBHD_SIZE
2.  for each pixel in ROI
        if pixel_value > robust_max
            pixel_value = robust_max
3.  mean = mean of ROI pixel values
4.  std = standard deviation of pixel values
5.  threshold = mean + K1*std
6.  load stored fiducial_template
7.  convolve ROI with fiducial_template -> response
8.  for each response_pixel in response
        if response_pixel_value > threshold
            if response_pixel_value is a local max
                for each fiducial_pixel in fiducials
                    if fiducial_pixel - response_pixel < fiducial_length
                        if response_pixel_value > fiducial_value
                            fiducial_pixel = fiducial_value
                        else
                            skip to next response_pixel
                    else
                        add response_pixel to fiducials
9.  if number of fiducials > 12
        sort fiducials by value
        keep 12 with highest response value
10. threshold = mean + K2*std
11. for each fiducial_pixel in fiducials
        nbhd = subimage of response with size NBHD_SIZE centered on fiducial_pixel
        refine = centroid of pixel in nbhd with value > threshold
        fiducial_pixel = fiducial_pixel + refine
12. Output: fiducials
```

The output to the next stage is the fiducials variable, a list of the coordinates of fiducials in pixel space. As with the last stage, there are a series of checks done at this

point to make sure the requirements of the next stage will be met. In this case, because the end goal is to build point correspondences between pixel and fiducial screen space, the algorithm fails here if there aren't at least three fiducials. This is the minimum for identification.

4.4 Identifying Fiducials

4.4.1 Assumptions and Requirements

Once fiducials are located, they must be identified. As discussed in the introduction, fiducials are spaced on a rotated grid. For each row of fiducials, vertical spacing between fiducials increases by a fixed amount, while horizontal spacing between fiducials is unique depending on location in the grid. The reverse is true down columns of the grid. The idea then is to find pairs of fiducials which meet the fixed spacing, and then identify the pair based on the varied spacing.

4.4.2 Ideal Case

Identification revolves around finding pairs of neighboring fiducials. These are spaced by a fixed distance in one axis, and a varying distance in the other which is used for identification. Identification proceeds by searching the entire list of fiducials for pairs of the correct fixed spacing either along either pixel rows or pixel columns. The result is two lists of pairs, one where the fixed distance along rows is valid, and one for column spacing. Each list is then searched for pairs with valid fiducial spacing in the other axis. If a valid pair is found, both members are given the ID for that distance. For example, if a pair of fiducials was found to be spaced by 15 pixels along rows, it would be deemed a "column pair". If the spacing along columns was 45 pixels, then this pair would get

column IDs of 0 and 1 respectively. Each would need to be a member of a valid “row pair” to get the other half of its ID.

If after this first pass there are fiducials which were not deemed to be members of any valid pairs, they are discarded. Usually these were false-positives from the previous stage. Tight thresholds were necessary on allowable row/column distances to prevent false positives, since false positives had the potential to adversely affect the mapping generated in the next stage. For this reason it was preferable to potentially have more missed detections as opposed to false positives.

After attempting to ID all the fiducials present, and dropping those without even a partial ID, there still are likely some with half of a valid ID pair. These are fiducials which only belonged to one pair, or to two co-linear pairs. To attempt to get around these partial IDs, fiducials missing say a row ID can take the value of an adjacent fiducial’s row ID if the two are in a column pair, and vice versa. Propagating IDs like this does have a drawback because a single incorrectly identified fiducial has the potential to corrupt the IDs of adjacent fiducials. It is also possible for adjacent fiducials to disagree over which row/column a common neighbor belongs to. In cases like these it is almost guaranteed that one of the adjacent fiducials has an invalid ID, but it is impossible to tell from this dispute alone which of the two is incorrectly identified. At absolute most there were only ever 12 fiducials visible, and often this number was closer to 4 or 5. With so few, it becomes very difficult to use any kind of majority vote to determine which fiducials were identified correctly. Ultimately the simplest strategy was taken for any disputes over ID of a given fiducial: if full ID for a fiducial could not be determined, the fiducial was simply ignored.

4.4.3 Pseudocode

The pseudocode below summarizes the logic used when identifying fiducial markers. As input it takes simply the locations, in pixel space, of identified fiducials. Also required at this point are the actual spacings for the fiducial grid. The `fixed_spacing` is the vertical spacing between adjacent pixels on the same row, or vice versa for columns. The `id_spacings` are the spacing between rows or columns of fiducials. In addition to this information, we add a `tol` variable, to capture how close to the expected distances pairs need to be.

```
1.  Input: fiducials, tol, fixed_spacing, id_spacings
2.  all_pairs = list of all pairs of fiducials
3.  for each pair in all_pairs
        if (row_spacing(pair) – fixed_spacing) < tol
            add pair to col_pairs
        if (col_spacing(pair) – fixed_spacing) < tol
            add pair to row_pairs
4.  for each pair in row_pairs
        for each row_id_spacing in id_spacings
            if (row_spacing(pair) – row_id_spacing) < tol
                for each member in pair
                    add row_id to member_row_ids
5.  for each fiducial
        if mode(member_row_ids) is unique
            fiducial_id_row = mode(member_row_ids)
        else
            fiducial_id_row = unknown
6.  Repeat 4 and 5 for columns
7.  for each pair in row_pairs
        for each member in pair
            if member_id_col is unknown
                if other_member_id_col is known
                    member_col_ids <- other_member_id_col
8.  for each fiducial
        if fiducial_id_col is unknown
            if mode(fiducial_col_ids) is unique
                fiducial_id_col = mode(fiducial_col_ids)
9.  Repeat 7,8 for columns
10. for each fiducial in fiducials
        if fiducial_id_row is unknown or fiducial_id_col is unknown
            remove fiducial from fiducials
11. Output: fiducials, fiducial_ids
```

Output here are two lists, one of fiducials, and the other a list of equal length with corresponding fiducial IDs. The list of fiducials is ideally identical to the one given as

input, but in the event that any fiducial was not able to be identified in the previous step, it is removed from the list. As with previous stages, these outputs need to be vetted before continuing to use them to generate a mapping, and mapping the solar center to screen coordinates. To generate a mapping, there should be at least 3 fiducials in the list, and their IDs should indicate that they are not co-linear: all three IDs should not belong to the same row or column. As long as this holds true, then it should be possible to map from pixel space to screen coordinates given the fiducials and IDs found here.

4.5 Mapping to Screen

4.5.1 Assumptions and Requirements

Once fiducials are identified, their true location in screen coordinates can be generated from calibration data. Their locations on the screen are known from the printing process, and calibrations before flight. The transformation from screen coordinates to gondola coordinates was also measured pre-flight. The goal at this stage is to generate a mapping from pixel to screen coordinates and to apply this mapping to the sun location found in the first stage. The assumption that mapping from plate to sensor is a scale and shift is used in this stage. This means that a simple linear fit can be used in each axis independently to model the change in coordinates from pixel space to fiducial screen. The only requirement levied on the previous stage is that there be a sufficient number of pairs to do a proper fit.

4.5.2 Pseudocode

As mentioned previously, this stage accepts a list of fiducial locations in pixel space and their corresponding IDs. The ID locations on the fiducial screen, and therefore

relative to the gondola coordinate system, are pre-determined by calibration on the ground. Mappings were generated with a simple linear least squares fit.

```
1.  Inputs: fiducials, fiducial_ids, screen_locations, sun_center
2.  screen_fiducials = screen_locations(fiducial_ids)

3.  for each axis in [row, column]
        mappings_axis = linear_fit(fiducials, screen_fiducials)
        screen_center_axis = mapping_axis(center_axis)
4.  Output: mappings, screen_center
```

This stage outputs the linear mappings generated, as well as the sun center location in mils relative to the center of the fiducial grid. The mappings are checked to verify that the identified scale factor is approximately correct. The `screen_center` is passed to another module for final adjustment into gondola coordinates, and then used to generate offset from the desired solar target.

4.6 Summary

The PYAS algorithm outlined above was used to generate pointing solutions for the HEROES balloon on its 2013 flight. The basic flow was established early in development, and functionality of each stage was fleshed out in sequence. Changes were made to each stage to account for disturbances, misalignments and the like as they were detected during testing. Much of the code was prototyped in MATLAB, but ultimately implemented in C++ using the OpenCV library. The next section will go over testing of the algorithm against artificial data in MATLAB, as well as performance and testing pre-flight with the OpenCV implementation.

5 Testing and Performance

5.1 Introduction

The task of verifying PYAS performance is about as complicated as the PYAS problem itself. Most of this arises from difficulty in getting a ground-truth for the instrument. In the case of the sun and fiducial finding, which were inspected most heavily, artificially generated data offers the only chance to test the algorithm against a known solution. The downside is that it is difficult to generate good approximations of the images actually generated by the PYAS. Verifying the PYAS on actual test observations obviously does not have this problem. On the other hand, while test observations are obviously an excellent representative of what will be seen on flight, they lack any ground truth at all. Inspection can't really offer solutions to a sub-pixel level, which leaves either a priori knowledge of the pointing of the PYAS system during a test, or another algorithm which has already been proven to work. The latter is impossible while only using images generated by the PYAS, and the former is impossible without an elaborate test setup able to adjust PYAS position relative to the sun in a precise way.

To verify requirements for the algorithm, sun tracking and fiducial detection are verified against synthetic test data. Description of how this data was generated and how it was used to test each segment of the algorithm is given in section 5.2. In contrast to this, the effects of image registration will be considered directly on data from full system tests. Finally, overall system performance will be assessed based on data from the 2013 HEROES flight itself.

5.2 Synthetic Data

5.2.1 Sun Finding

In the case of the sun finder, artificial data was created by drawing a filled circle at a scale of 1 arcseconds per pixel then down-sampling the image by a factor 10.6 to approximate the ideal PYAS image scale. After a simple Gaussian blur and the addition of some white noise, these images are a fair approximation of actual PYAS images, with ground truth given by the scaled coordinates of the sun center in the original image. The PYAS sun tracking algorithm was tested against these images, and performance was measured against number of chords and level of white noise. Noise level varied, but the parameters below describe the other parameters of the set of test images.

Parameter	**Value**
Range of centers	-5 to 5 arcseconds in each axis
Sun diameter	180 pixels
Oversampling factor (arcsecond to pixels)	10.7

Table 10 Parameters for synthetic sun test images

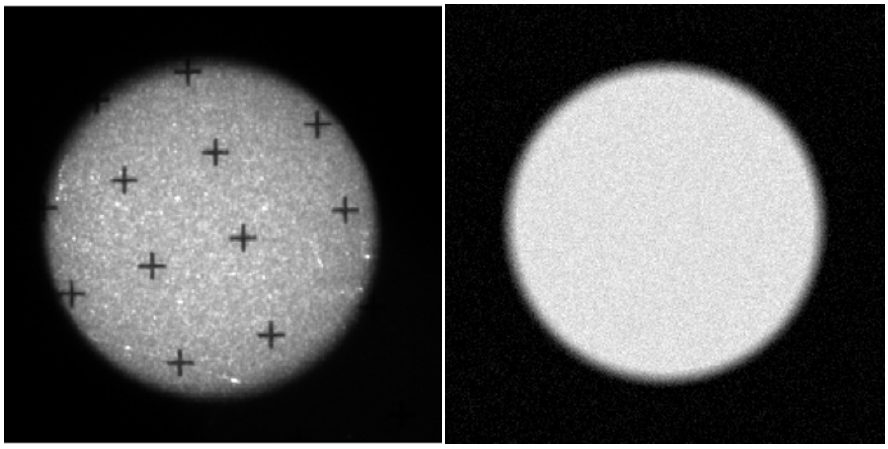

Figure 10 Cropped PYAS image from sun test data (left). Synthetic sun image used in testing (right)

This stage of the PYAS algorithm has two parameters which can be tuned to affect response: threshold level K and number of chords N. In all tests the threshold K was set to 50% of max brightness. In this test, N was varied to see the effect of the number of chords on accuracy. The standard deviation of the white noise used in this portion of the test was held fixed at a level similar to that of ground test data. The parameters for this test are tabulated in the table below.

Parameter	**Value**
Noise Std Dev	5 counts
Threshold	105 counts
Number of Chords (N)	3 to 90 per axis

Table 11 Parameters for test of performance vs number of chords

The results are shown below. The center determined by this algorithm was compared to the ideal center, and the magnitude of this error was used to measure performance. The plot below shows the 3-sigma error in arcseconds over all positions for a given number of chords. There's clearly a sharp improvement in performance initially, which starts to flatten out after 10 chords per axis. Based on this test the number of chords to be used in flight was set at 10.

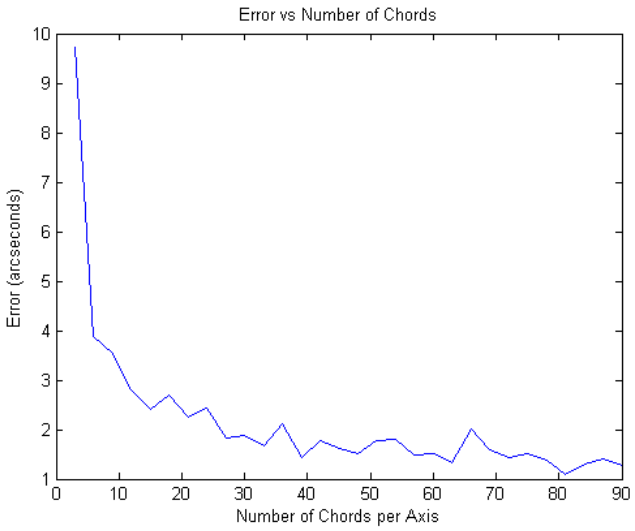

Figure 11 Performance vs. number of chords. Error shown is 3sigma of error values for each parameter value.

The next test used a fixed number of chords and varied the noise level to see how sensitive this method is to increased noise. A range of values were used for sigma: from noise-free to an order of magnitude worse than the expected noise level. Again, parameters for this test are tabulated in the table below.

Parameter	**Value**
Noise sigma	.01 to 100 counts
Threshold	105 counts
Number of Chords (N)	10 per axis

Table 12 Parameters for test of chord finding vs noise

As before, performance can be measured in terms of 3-sigma error versus noise level. Also of interest, however, is number of chords dropped. The likelihood of a chord failing to meet the criteria listed above for a valid chord increases with noise level. At a certain point no valid chords will be found in the image. Although it is unlikely that such a high level of noise would be encountered on flight, it was of interest in development to see

how robust this method is to high levels of noise. Plots of both error vs. noise and dropped chords vs. noise are shown below.

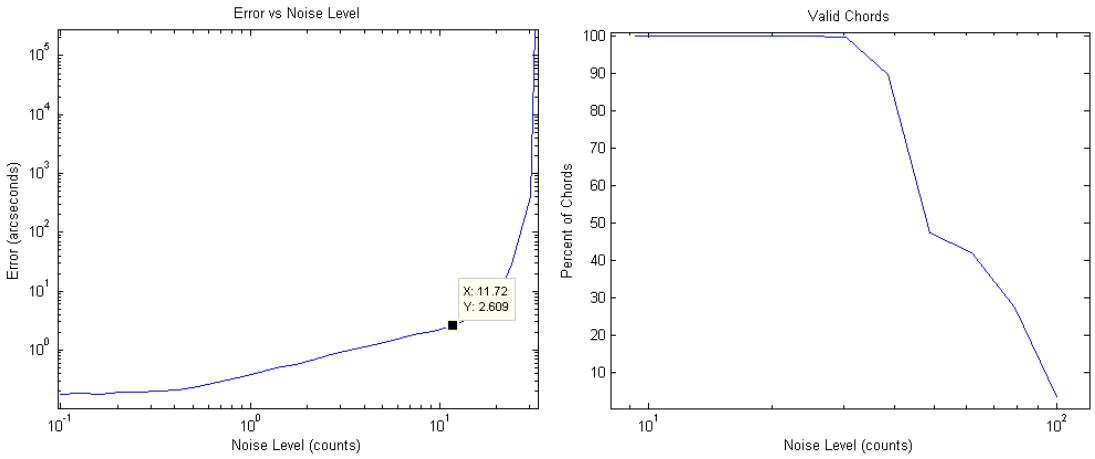

Figure 12 Plots of error and percentage of dropped chords vs. noise level. Noise level is measured in bits in the intensity, and chords are dropped when they fail to meet the criteria outlined in Chapter 4

From the plot of error, we see that beyond noise levels of 15 counts or so, the sun tracking algorithm fails to meet requirements at all. At the even higher level of 30 counts, we see that the number of valid chords plummets, corresponding to a spike in error. The hope was that noise levels in flight data would be significantly less than this.

The last test of this algorithm was designed to see how it behaves in the presence of fiducial markers. For this, a simulated screen image was generated using the same method used in generating the sun image: an image was generated at higher scale and then scaled down to true size. Portions of this screen were then “illuminated” by multiplying regions of the screen image with the synthetic solar image, giving test images like the one shown in the figure below.

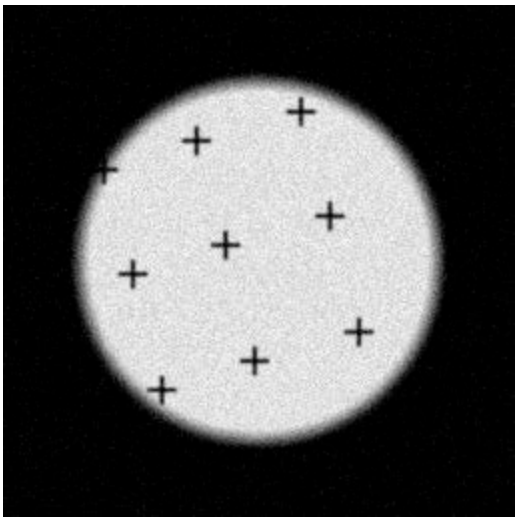

Figure 13 Test image containing fiducial markers

The parameters for this battery of testing are listed in the table below. The goal here was to see how well this algorithm can handle encountering fiducial markers. While certainly not perfect, comparing this to the flight data at the start of this section shows that it is a reasonable approximation of what true PYAS images look like.

Parameter	**Value**
Noise Std Dev	5 counts
Threshold	105 counts
Number of Chords (N)	10 per axis
Sun center	(0 arcseconds, 0 arcseconds)
Background	One of 81 subsets of fiducial pattern
Trials per background	10

Table 13 Parameters for test of sun-detection in the presence of fiducial markers

The results of this test are tabulated in the histogram of RSS error values shown below. Recall that the goal here is to have an RSS error below 10 arcseconds.

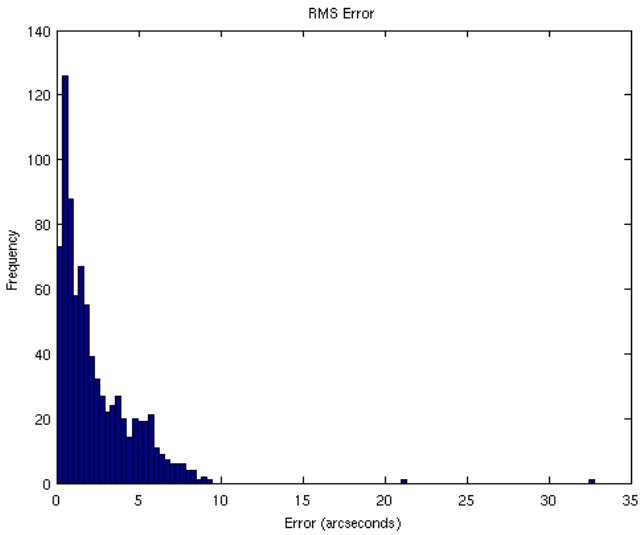

Figure 14 Histogram of error levels for trials in the test of chord-based sun-detection in the presence of fiducial markers. Almost all tests fell within the 10 arcseconds requirement.

The bulk of values fall below this threshold, but it is clear that there is a subset of frames where this algorithm fails to meet the requirement. Even though most values fall below the threshold, the bound on error here is much higher than what was seen in tests without fiducials present. The reason for this is visible in the example frame shown above, and was overlooked in the planning stages of the PYAS code. Looking at the top left fiducial in the sample image above, it is clear that if a chord were to fall horizontally across the fiducial it would be impossible to tell where the true edge of the sun was. The right edge of the horizontal arm of that fiducial would be detected as the edge of the sun, while the actual edge lies closer to the left edge of the marker. This chord would introduce an offset, and given that there are only 10 chords at most through the whole image, one error can introduce a significant pull on the estimate of the center.

5.2.2 Fiducial Detection

Artificial fiducial marker data was described briefly in the previous section, where an artificial fiducial screen was used to test the sun tracker. In this section rather than the entire screen, only images of a single fiducial will be considered. With the threshold set properly, the fiducial detection was able to determine fiducial location to a pixel level for fully-illuminated fiducials without any issues. Of interest here is how accurately the refinement method described in Chapter 3 can resolve sub-pixel locations.

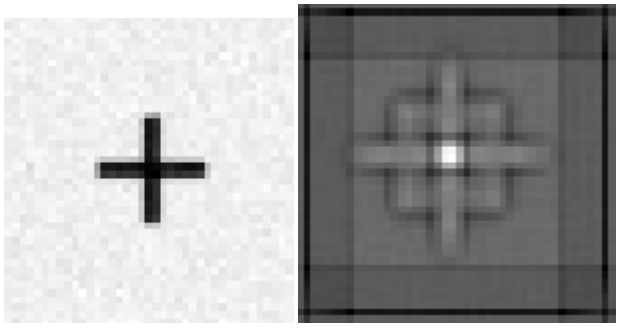

Figure 15 Example of a fiducial test image, and the corresponding correlation image

To test this, a set of test images was generated where the location of the fiducial mark varied across 6 different locations in mils on the fiducial screen in each axis. All 36 of these locations would produce a peak in the correlation response at the same pixel. The error in fiducial location without any refinement is listed in the table below. Also included are the results of refining with a centroid as described in chapter 3, which was the method taken with this algorithm.

	No Refinement	Centroid (FLIGHT)	Centroid (Improved)
3sigma RSS error	4.2 arcseconds	1.55 arcseconds	.68 arcseconds

Table 14 Performance of various fiducial location methods. Flight method is marked (FLIGHT), and an improvement which accounts for some of the systematic error is given as (Improved)

Error for taking no action is what would be expected, the equivalent of about half a pixel. Refining the location of the peak with a centroid was able to cut that value in half. Centroiding to determine correlation peak was chosen because the method is fast, and because during development there were problems forming a well-conditioned paraboloid fitting problem. Paraboloid fitting would have been the preferred approach here, and was shown in the literature to perform nearly an order of magnitude better than the method chosen here. The reason for poor performance of the centroid is a systematic error described in the graphs below.

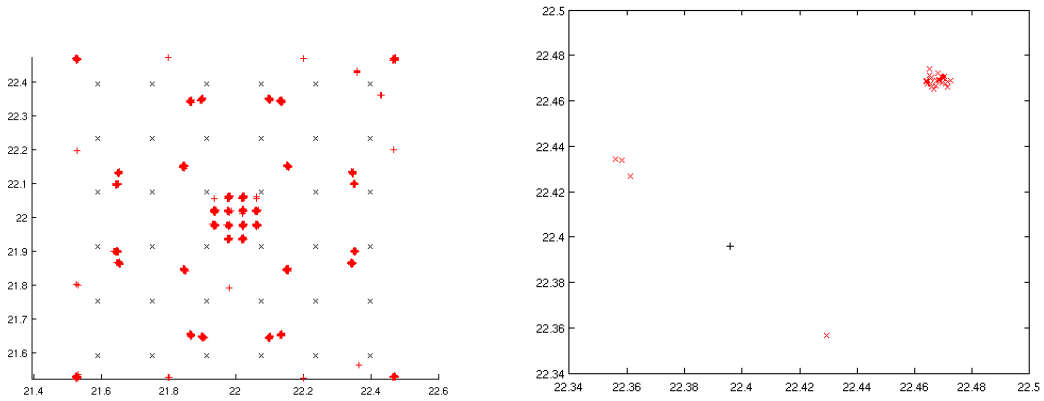

Figure 16 At left, true fiducial locations are shown in black, all corresponding to the same integer pixel location. Attempts to refine fiducial location to sub-pixel are shown in red. Clearly there is systematic error present. At right, the same is shown for a single point.

The graph on the left shows a plot of predicted locations in red vs. true locations in blue for all the locations tested. The right shows an example for a single point. Part of this seems to be intrinsic to using a centroid to refine position of a peak, but there is also a component here caused by not removing the threshold value from the pixel values before computing the centroid. Points farther from the center have a greater moment on the resulting solution, and by not subtracting the threshold, they also have more "mass." The results of subtracting this value when computing centroid are shown below.

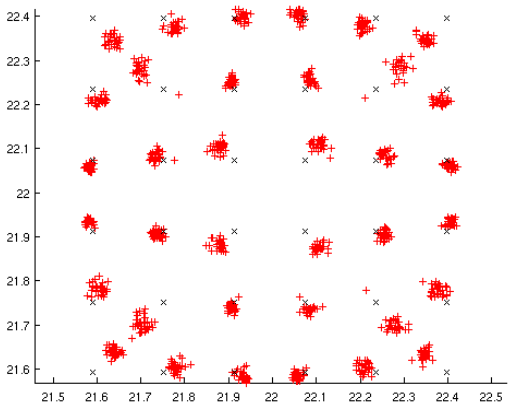

Figure 17 Same as figure 16, but with an improvement on the method used on flight.

Although clearly an improvement, there is still systematic error in this estimate. Several alternate methods for refining fiducial locations were considered earlier, and one of these would likely be a better candidate than the centroid-based refinement described here.

5.3 Test Data

Several tests were done of the PYAS system before flight, the vast majority of which were performed while attached to the HEROES payload. Performance of the sun tracking

portion of the code cannot readily be assessed because the PYAS is folded into the larger HEROES control loop. On the other hand, the fiducial plate and camera are fixed relative to each other, so performance of the mapping computation can be assessed directly. For the fiducial location and identification, inspection shows that missed detections and false-positives are uncommon, but quantifying these values would require inspection of every frame. Instead, this section will look at the jitter in fiducial locations and at how jitter in fiducial locations propagates through to the final aspect solution.

5.3.1 Fiducials

Although the sun may move while the gondola pans to track a solar target, the fiducial pattern is fixed relative to the camera. Therefore motion of the pattern relative to the camera should be fairly small. Although some of the jitter on fiducials may come from motion of the gondola or flex of the PYAS, assuming all noise present is due to the detection algorithm is a fair worst-case assumption. The plots below show scatter of fiducial measurements and RSS error vs. the mean fiducial location for what should have been a static observation.

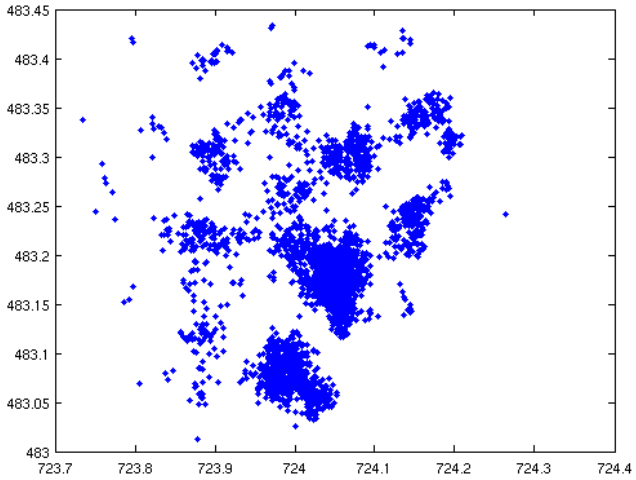

Figure 18 Spread of fiducial locations from a test of the PYAS system

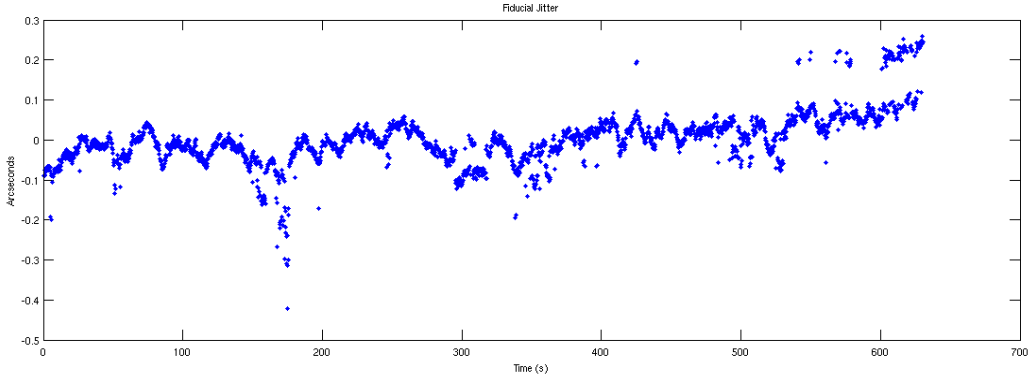

Figure 19 Fiducial location from a ground test, in a single dimension, plotted as a time series

It is interesting to see that there is some structure to the data. The quantization seen in the 2D scatter is likely a result of the systematic error shown to be present in the centroid method for refining fiducial location. This same quantization is even visible in the time series data after 800s, where there the data is clearly bimodal. As for the periodic nature of the time series, a 25s cadence could reasonably be motion of the gondola.

5.3.2 Registration

In tests with synthetic data it was concluded that the sun tracking algorithm in the presence of fiducials and noise is only trustworthy to about a pixel, or 10 arcseconds. This is an approximation which does not factor in effects of mapping from pixel space to the coordinate space of the fiducial screen. The effect of the mapping could be computed analytically, either assuming a Gaussian distribution on the location of the fiducials, or approximating one by computing the standard deviation of the jitter plotted above. Instead, however, direct measurements of jitter on the parameters of the fit can be used. The mapping in question takes the form of two linear fits, one in each axis. The parameters of these fits were computed for every frame in an observation of a fixed target, and their variances are given in the table below, along with an estimate of the variance of the points in the sun center.

	Variance
$\sigma_{m_1}^2$	1.045 (mils/pixel)^2
$\sigma_{m_2}^2$	1.579 (mils/pixel)^2
$\sigma_{b_1}^2$	5.089 mils^2
$\sigma_{b_2}^2$	3.111 mils^2
σ_x^2, σ_y^2	.0978 pixels^2

Table 15 Measured noise values for parameters in the image registration stage

Hartley and Zisserman give an expression for the covariance matrix of a noisy point mapped with a noisy transformation **H** [29]. The covariance for the estimate **x'** of the correspondence for **x** under **H**, denoted $\boldsymbol{\Sigma}_{\mathbf{x}'}$, is given by:

$$\Sigma_{x'} = J_h \Sigma_h J_h^T + J_x \Sigma_x J_x^T$$

In their notation, the mapping **H** is a matrix of the form

$$\begin{bmatrix} m_1 & 0 & b_1 \\ 0 & m_2 & b_2 \\ 0 & 0 & 1 \end{bmatrix}$$

This means that the vectorized version **h** takes the form

$$\begin{bmatrix} m_1 \\ m_2 \\ b_1 \\ b_2 \end{bmatrix}$$

And the parameters in the expression work out to be:

$$J_h = \begin{bmatrix} x & 0 & 1 & 0 \\ 0 & y & 0 & 1 \end{bmatrix}$$

$$J_x = \begin{bmatrix} m_1 & 0 & b_1 - x' \\ m_2 & 0 & b_2 - y' \end{bmatrix}$$

The values for Σ_x and Σ_h are diagonal matrices populated by the table above. Combining all these values together, the worst case estimate of covariance for the coordinates of the sun center is 14.4 mils^2. Adjusting for plate scale the total error estimated to have standard deviation of 6.512 arcseconds. Comparing to the 3σ bounds being used earlier, that puts the estimate for error in the computed solution at 19.5 arcseconds, just barely under the 20 arcsecond requirement.

5.4 Summary and Flight Performance

To review, there were a handful of requirements levied on the SAS which were relevant to the PYAS algorithm. Accuracy was the driving requirement and has been the focus of this chapter. Software also had a direct impact on cadence, and this requirement has been qualitatively addressed throughout this work. The design of the algorithm was made with cadence in mind, which helped in making image saving, rather than processing, the tall pole from a cadence standpoint. Finally FOV was mentioned as a

requirement but also not addressed explicitly. The PYAS frame was sized so that the entire sun was fully visible anywhere in a +/-1 degree range. Outside of this range the chord-finding method will return a degraded solution, and it becomes increasingly unlikely to have sufficient fiducials to register the PYAS frame to the fiducial screen.

Requirement	Minimum	Target	Verification	Result
Cadence	1 Hz	4 Hz	Test Data	3.97 Hz
Pitch/Yaw Knowledge (flight)	< 20 arcseconds	< 10 arcseconds	Synthetic Data	10 arcseconds (sun tracker)
			Mapping	See 4.3
			Total	19.5 arcseconds
Knowledge (post-flight)	< 15 arcseconds	< 15 arcseconds	--	--
FOV	2°	2.79°	SAS Design	2°

Table 16 Summary of algorithm requirements and measured performance

Ideally the flight processing algorithm would have enough margin to double as post-flight algorithm as well. Unfortunately the PYAS flight algorithm overshot its target accuracy level and appears to only barely meet the knowledge requirement for flight. The flight was successful however, and lessons learned in development and testing of the algorithm apply directly to the post-flight data processing. Recommendations for improved accuracy when processing PYAS frames will be described in the next and last chapter.

As for the flight, the HEROES payload had a successful launch in September of 2013 [30]. The flight lasted over 24 hours, and the payload was at float for 21 of those. Solar pointing time was 7 hours, and the PYAS system provided aspect solutions for the duration. During these observations, the HEROES control loop was required to point the payload with pitch-yaw jitter having a 50^{th} percentile of 1'. Measurements taken by the PYAS-F show that over the duration of the solar pointing period, the HEROES payload was within 10 arcseconds of the target in elevation and 30 arcseconds of the target in

azimuth 50% of the time. Even factoring in 20 arcseconds of potential error in aspect knowledge, the HEROES payload still meets pointing requirements. The flight knowledge requirement levied on the PYAS-F is rolled into this jitter requirement, and meeting it means the HEROES telescope was pointed successfully.

6 Conclusion

6.1 Ground Processing

Any lessons learned from the flight processing algorithm for the PYAS can immediately be applied to the ground processing algorithm. These range from minor changes in the flight code to a completely new algorithm. Some aspects of the flight algorithm worked adequately: fiducial identification and mapping from pixels to screen. However the algorithm would benefit from a major change to its overall structure, and then adjustments to the first two stages.

First and foremost, the sun tracking code needs to change. The chord-based method was attractive because of its nature as a "heritage" approach, being very similar to the method used by RHESSI in its SAS. RHESSI did not have to contend with fiducial marks, however, which can corrupt the solar limb and provide a disturbance when they occur on the body of the sun. Fiducials at the edge of the sun will offer a disturbance to any edge-based circle-fitting method, meaning that curve-fitting and Hough transform would both encounter similar problems.

There are two potential solutions to this problem. One is moving away from edge-based methods. Circle-enclosing methods like the one described in [31], for example, are a simple option. With an appropriate threshold this method should be able to provide a reasonably good estimate of the sun center. Fiducials effectively subtract portions of the sun, but enough of the edge is present that an enclosing method should give a good estimate of the center. Another alternative would be to revisit the use of a simple centroid. Centroids were ruled out early in the development of the PYAS algorithm

because fiducial markers would occlude portions of the sun and affect the centroid. This may have been more of a problem if large convex shapes like circles or diamonds were used as marks. The cross marks used in the PYAS darken at worst 56 pixels per mark. At most that means about 600 pixels are darkened on the sun. With a diameter of about 180 pixels, the sun illuminates 24300 pixels, meaning at worst about 3% of the sun is occluded. It's possible that the affect of this on the centroid of the sun would be minimal. Converting to a binary image and using a morphological closing could further reduce the impact of fiducials on a centroid estimate.

The other potential solution would be to change the basic structure of the algorithm. Edge based methods could still be used if there was a way to predict the location of fiducials. Changing the structure of the algorithm could make it possible to predict fiducial locations, and then mask these regions when searching for edge pixels. An improved algorithm might proceed as follows:

	Stage	**Input (space)**	**Output (space)**
1	Determine coarse center	Raw PYAS frame	Coarse center (pixels) Solar ROI
2	Locate visible fiducials (pixel)	Solar ROI	Fiducials (pixel)
3	Identify visible fiducials markers	Fiducials (pixel)	Fiducials (pixel) Fiducials (screen)
4	Predict fiducial locations	Fiducials (pixel) Fiducials (screen)	Fiducial Mask (pixel)
5	Determine fine center	Solar ROI Fiducial Mask	Fine center (pixel)
6	Transform sun center (gondola)	Fiducials (pixel) Fiducials (screen) Fine center (pixel)	Sun center (screen)

Table 17 Steps for a possible improved PYAS algorithm

Coarse center could be found with something like the winner-take-all method from [10]. After fiducials are found and identified, they could be used to predict regions in the

ROI where fiducials may appear, providing some measure of which portions of the edges are trustworthy for circle fitting. An edge-based circle finding method like curve-fitting or Hough could then ignore edge pixels that potentially correspond to fiducial marks.

In addition to modifying how the sun center is located, the current method of refining fiducial location needs to be changed. The systematic error present in the current method could be eliminated with any of the refinement methods described in Chapter 3. A correlation-based method for locating fiducials worked extremely well, and a parabolic surface fit to the correlation should be sufficient to provide an estimate of sub-pixel location free of the errors present in the centroid method. With these two modifications, it ought to be possible to get to the original target for flight pitch-yaw knowledge.

6.2 Hardware

The changes described above should be implemented for post-flight processing, and for any future systems which aim to build on the HEROES SAS. In the vein of future systems, there are changes to both the SAS electronics and fiducial pattern that could potentially have made the software problem easier and even allowed for improved performance. The first of these would be to try to upgrade the SAS computer. The initial goal of a 10 Hz cadence proved to be impossible with the current hardware. A faster processor would certainly help here, and there are certainly processors more powerful than the Atom processor used by the SAS computers. Additionally, more time needs to be devoted to the problem of storing images. Images in the SAS were stored as uncompressed FITS files, and file I/O ended up being a major bottleneck in the processing pipeline. PYAS frames have a great deal of empty space, and if not video,

even lossless image compression would cut down on raw file size and might help to improve file write times.

In addition to improvements to the electronics, the fiducial pattern could benefit from some modification. Cross-shaped fiducial markers are easy for humans to locate, but circular markers may be more suited to machine vision applications. They are easier to locate precisely via a simple centroid, and although the PYAS did not encounter problems with rotations, circular marks would be much more robust to potential rotation between the camera and screen. The method of encoding fiducial ID in inter-fiducial distances should also be re-addressed. The current method is very sensitive to changes in scale, and required adjustment to parameters in the identification code between test configuration and mounting on the gondola. Augmented reality literature is full of fiducial markers that carry information payloads, and there are even patterns of relatively small marks that still carry information without relying on inter-fiducial distance. For example, the fiducial marks used in the 2-axis encoder described in [32] rely on a pattern of small marks which can deliver position knowledge when only part of the pattern is visible.

The PYAS system successfully provided fine pitch-yaw knowledge to the HEROES pointing control system during solar observation. It exceeded cadence requirements, and analysis shows that it managed to meet pointing knowledge requirements, if barely. Solar pointing data from flight appears to have met jitter requirements in both axes as well. Because it was being developed in parallel with the PYAS hardware, the PYAS algorithm had to be capable of handling a wider array of possible scenes. Now that the PYAS optics, fiducial pattern, and camera are finalized, however, and the concerns about

processing time have been completely removed, it should be possible to extract much finer pitch and yaw knowledge from the recorded PYAS frames. Further improvements could be made in future systems based on the PYAS by tailoring the electronics or to the fiducial pattern to make the computer vision problem easier.

Bibliography

[1] B. D. Ramsey, C. D. Alexander, J. A. Apple, C. M. Benson, K. L. Dietz, R. F. Elsner, D. E. Engelhaupt, K. K. Ghosh, J. Kolodziejczak, Jeffery, S. L. O'Dell, C. O. Speegle, D. A. Swarts, and M. C. Weisskopf, "FIRST IMAGES FROM HERO -- A HARD-X-RAY FOCUSING TELESCOPE," *Astrophys. J.*, 2002.

[2] J. R. Wertz, *Spacecraft Attitude Determination and Control*. Springer, 1978.

[3] W. M. Morgenstern, K. L. Bourkland, O. C. Hsu, K. Liu, and P. A. C. Mason, "Solar Dynamics Observatory Guidance , Navigation , and Control System Overview," *AIAA Guid. Navig. Control Conf.*, no. August, pp. 1–18, 2011.

[4] G. T. Ward, David K., Davis, "The Microwave Anisotropy Probe Guidance, Navigation, and Control Hardware Suite."

[5] B. Robertson, S. Placanica, and W. Morgenstern, "TRMM On-Orbit Attitude Control System Performance," *21st Annu. AAS Guid. Control Conf.*, 1999.

[6] T. T. TARSHIS and G. T. SAKODA, "A SECOND GENERATION SUN SENSOR FO SOUNDING ROCKET APPLICATIONS," pp. 85–90, 1979.

[7] P. Ortega, G. López-rodríguez, J. Ricart, M. Domínguez, L. M. Castañer, S. Member, J. M. Quero, C. L. Tarrida, J. García, M. Reina, and A. Gras, "A Miniaturized Two Axis Sun Sensor for Attitude Control of Nano-Satellites," *IEEE Sens. J.*, vol. 10, no. 10, pp. 1623–1632, 2010.

[8] R. Henneck, J. Bialkowski, F. Burri, M. Fivian, W. Hajdas, A. Mchedlishvili, P. Ming, K. Thomsen, J. Welte, A. Zehnder, B. R. Dennis, G. Hurford, D. Curtis, and D. Pankow, "The Solar Aspect System (SAS) for the High Energy Solar Spectroscopic Imager HESSI," *SPIE Conf. EUV, X-Ray Gamma-Ray Instrum. fo Astron.*, vol. 3765, no. July, pp. 771–776, 1999.

[9] M. Fivian, R. Henneck, and A. Zehnder, "RHESSI Aspect System and In-flight Calibration," *Proc. SPIE*, vol. 4853, pp. 60–70, 2003.

[10] N. Xie and A. J. P. Theuwissen, "A Miniaturized Micro-Digital Sun Sensor by Means of Low-power Low-noise CMOS Imager," *IEEE Sens. J.*, vol. 14, no. 1, pp. 96–103, 2014.

[11] S. Mobasser, C. C. Liebe, and J. Naegle, "Flight Qualified Micro Sun Sensor for Mars Applications," *Proc. 2nd Int. Conf. Recent Adv. Sp. Technol. 2005. RAST 2005.*, vol. 3, pp. 234–239, 2005.

[12] X. Fernàndex and J. Amat, “RESEARCH ON SMALL FIDUCIAL MARK USE FOR ROBOTIC MANIPULATION AND ALIGNMENT OF OPHTHALMIC LENSES,” *IEEE Conf. Emerg. Technol. Fact. Autom.*, pp. 1143–1146, 1999.

[13] C. B. Bose and I. Amir, “Design of Fiducials for Accurate Registration Using Machine Vision,” *IEEE Trans. Patter*, vol. 12, no. 12, pp. 1196–1200, 1990.

[14] M. Tichem and M. S. Cohen, “Sub micrometer Registration of Fiducial Marks Using Machine Vision,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 8, pp. 6–9, 1994.

[15] H. Moon, R. Chellappa, and A. Rosenfeld, “Optimal edge-based shape detection.,” *IEEE Trans. Image Process.*, vol. 11, no. 11, pp. 1209–1226, 2002.

[16] R. C. Gonzalez and R. E. Woods, *Digial Image Processing*. Upper Saddle River, NJ: Prentice Hall, 2008.

[17] H. K. Yuen, J. Princen, J. Dlingworth, and J. Kittler, “A Comparative Study of Hough Transform Methods for Circle Finding,” *Procedings Alvey Vis. Conf. 1989*, pp. 29.1–29.6, 1989.

[18] A. Fitzgibbon, M. Pilu, and R. B. Fisher, “Direct Least Square Fitting of Ellipses,” *Pattern Anal. Mach. Intell.*, vol. 21, no. 5, pp. 476–480, 1999.

[19] W. Gander, G. H. Golub, and R. Strebel, “Least-squares fitting of circles and ellipses,” *BIT*, vol. 34, no. 4, pp. 558–578, Dec. 1994.

[20] I. D. Coope, “Circle Fitting by Linear and Nonlinear Least Squares,” *J. Optim. Theory Appl.*, vol. 76, no. 2, pp. 381–388, Feb. 1993.

[21] D. Umbach and K. N. Jones, “A Few Methods for Fitting Circles to Data,” *IEEE Trans. Instrum. Meas.*, vol. 52, no. 6, pp. 1881–1885, 2003.

[22] W. W. Seemuller and P. Chen, “Aerial Photograph Reseau Mensuration with Sensing Arrays,” *IEEE Trans. Instrum. Meas.*, vol. 26, no. 1, pp. 29–32, 1977.

[23] B. F. Alexander and K. C. Ng, “Elimination of systematic error in subpixel accuracy centroid estimation,” *Opt. Eng.*, vol. 30, no. 9, pp. 1320–1331, 1991.

[24] S. S. Welch, “Effects of Window Size and Shape on Accuracy of Subpixel Centroid Estimation of Target Images,” *NASA Tech. Pap.*, no. 3331, 1993.

[25] S. Lee, “Pointing Accuracy Improvement using Model-Based Noise Reduction Method,” *Proc. SPIE*, pp. 65–71, Apr. 2002.

[26] D. K. Naidu and R. B. Fisher, “A Comparative Analysis of Algorithms for Determining the Peak Position of a Stripe to Sub-pixel Accuracy,” *Procedings Br. Mach. Vis. Conf. 1991*, pp. 28.1–28.9, 1991.

[27] Q. Tian and M. N. Huhns, “Algorithms for Subpixel Registration,” *Comput. Vision, Graph. Image Process.*, vol. 35, pp. 220–233, 1986.

[28] S. S. Gleason, M. A. Hunt, and W. B. Jatko, “Subpixel measurement of image features based on paraboiloid surface fit,” vol. 1386, pp. 135–144, 1990.

[29] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd Editio. Cambridge, UK: Cambridge University Press, 2000.

[30] S. Christe, A. Shih, M. Rodriguez, K. Gregory, A. Cramer, M. Edgerton, E. J. Gaskin, B. O. Connor, and A. Sobey, “A Solar Aspect System for the HEROES Mission,” *IEEE Aerosp. Conf.*, 2014.

[31] E. Welzl, “Smallest enclosing disks (balls and ellipsoids),” *New Results Trends Comput. Sci.*, vol. 555, pp. 359–370, 1991.

[32] D. B. Leviton, T. Anderjaska, J. Badger, T. Capon, C. Davis, B. Dicks, W. Eichhorn, M. Garza, C. Guishard, S. Haghani, C. Hakun, P. Haney, D. Happs, L. Hovmand, M. Kadari, J. Kirk, R. Nyquist, F. D. Robinson, J. Sullivan, and E. Wilson, “Cryogenic optical position encoders for mechanisms in the JWST optical telescope element simulator (OSIM),” *Proc. SPIE*, vol. 8863, Sep. 2013.