# Development and Testing of core Flight Software Applications, and Experimentation of AI Fiducial and Lens Calibration Methods for Application in a Drone Mapping Operation

Jaspreet Ranjit; Mentor: Alessandro Geist; Code 587: Science Data Processing; University of Virginia; Project Category: Computer Science/IT

## Timeline

- 2024
- 2026 — Outbound cruise to 67P
- 2028
- 2030 — Comet Ops
- 2032
- 2034 — Inbound cruise to Earth
- 2036

# 1. Overview

## Mission CAESAR

### Goal

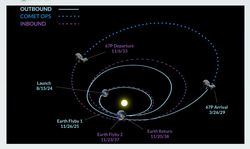
Mission CAESAR (Comet Astrobiology Exploration Sample Return)'s main goal is to collect a sample from comet 67P/Churyumov-Gerasimenko and return it to Earth. It will be the most extensive study done on cometary material and the farthest reaching sample return mission in history. The first four and a half years will be spent traveling to Comet 67P, passing by Earth after one year to gain speed with a gravity assist, and to calibrate the cameras by imaging the Earth and Moon. After arrival, another four and a half years will be spent imaging the comet's surface and conducting trial runs of descent to the surface to ensure the sampling device can touch the surface (Figure 2). CAESAR will perform a touch-and-go maneuver at a target location that has smooth terrain rich in organic materials and ice. As the spacecraft reaches the target, the solar arrays will tip backward and the robotic arm will transfer the sampling device to the target where gas will be used to deliver the sample into the container (Figure 1).

### Comet 67P

Comet 67P has been studied in the past by the Rosetta spacecraft and its Philae lander. As a result, extensive knowledge of the comet's surface already exists, and CAESAR can focus on sample return with a reduced payload consisting of cameras, a sampling device, and a containment system. The preserved samples would give insight into what role comets played in bringing life about to Earth. Comets contributed to the birth of the solar system, and could provide valuable information about organic material that was present when the planets were formed.

Figure 1

Figure 2




http://caesar.cornell.edu/overview/

### Significance

Comets are considered to be one of the building blocks of planets, contain Earth's water and organic molecules, and are like time capsules that contain preserved molecules form the beginning of our time. The can provide valuable insight regarding the earliest formation of solid objects around the Sun. CAESAR would help progress towards questions regarding the birth of the solar system, how comets contributed to early life and if they delivered water to earth, and information regarding organics that could have contributed to early life. By expanding the current knowledge base about how the planets were formed, mission CAESAR could contribute to the search for habitable planets in other solar systems.

## SpaceCube

### Origin

The SpaceCube project originated from a need for a highly efficient and compact processing system with capabilities of operating in hostile environments for future space exploration missions. The processing system would ideally mitigate the effects of radiation and provide radiation hardening to preserve the device under severe conditions as experienced during space exploration missions. There are a wide range of processors including circuit boards and wired/fixed or removable electrical components with the capabilities to handle data, and perform quick computations. However, current processors are lacking in their ability to protect against the adverse effects of radiation that can lead to errors in processing. Modern radiation tolerance in electrical components consists of three levels: unprotected, tolerant, and radiation hardened. Although radiation hardened components are favorable, the processing power of these components is compromised for radiation mitigation.

### Design

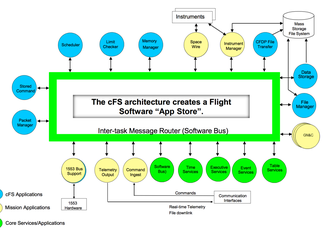
The SpaceCube is unique in its capability to provide highly efficient processing power and functionality while embodying the characteristics of a radiation hardened electrical component. The SpaceCube is a reconfigurable field programmable gate array (FPGA). An FPGA is a semiconductor that consists of configurable logic blocks with programmable interconnects. An FPGA can be reprogrammed for individualized applications, and is radiation tolerant with capabilities for image processing, and efficient reconfiguration. The greatest advantage of the FPGA is its ability to withstand the effects of radiation. The SpaceCube uses the radiation-tolerant Xilinx Virtex FPGA with upset recovery architecture giving it the ability to correct and recover errors in autonomous missions. Future models of the SpaceCube aim to provide up to 100x improvements in computing power over modern leading processors in the market potentially capable of producing data at rates of 10^8 to 10^11 bits/second. With its reprogrammable nature, the SpaceCube has the ability to house multiple, versatile applications such as image processing, robotic servicing, real time event detection, data compression, calibration and correction, and other control functions.

## core Flight Software

### Overview

The core Flight Software (cFS) system is a custom NASA, independent software platform and framework consisting of reusable software applications to be used in space missions. The flight software controls the mission, and is responsible for handling real time errors that may occur in flight. cFS consists of a run-time environment, a layered software architecture, and a component based design. Its flexible architecture allows cFS applications and software to be reused on NASA space missions which saves both time and money (Figure 3). The development of cFS was in direct response to the rapid improvements in hardware, and the increasing complexity of flight software that supports large space exploration missions. cFS consists of foundational software applications that can be reused, and configured to be unique for each software environment. One of the greatest advantages cFS provides is the ability to develop, run, and test the software applications on a developer's software environment and transfer the changes to an embedded environment that will be used in the mission. Software maintenance, one of the costliest parts of the software development cycle, is also simplified with the ability to make modifications on board during flight. The main advantages of the core Flight Software system streamline the project schedule and deployment of flight software, promote and simplify software reuse by providing a standard for missions across all NASA centers, and provide a simplified and efficient platform for prototyping new software applications.

Figure 3




https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf

# 2. Plan

## Project Goals

Mission CAESAR will be using the SpaceCube to conduct experimental studies for the duration of the flight. SpaceCube will be running the core Flight Software system with various cFS applications that perform tasks necessary to the mission. Two core Flight Software applications were modified and tested for the purposes of this study: Retina and GEONS. With the ability to test the applications on a developer's platform, Retina was benchmarked and tested with different processors to ensure quality results. Retina was benchmarked by cross compiling with the Microblaze processor on the virtex 5 card. The output was compared to cross compiling with the PowerPC processor to find the most efficient platform for the applications to run on, and to verify the accuracy of the output. The application was evaluated based on runtime and accuracy of the output. A method for testing the GEONS application was developed from a MATLAB API, and various python scripts that outputted commands and files that could be fed into GEONS. Because GEONS requires commands and CCSDS packets, these files had to be generated in some manner in order to run and test GEONS to verify the accuracy of the output. A method was developed to modify and feed data to the GEONS application, and verify the output. The application will also benchmarked with timer statements embedded in the code to determine the efficiency of each function. A preliminary drone test will be conducted as a proof of concept for the camera suite that will be used in CAESAR. A camera calibration and AI fiducial marker tracking method was developed for this phase of the project (Figure 4).

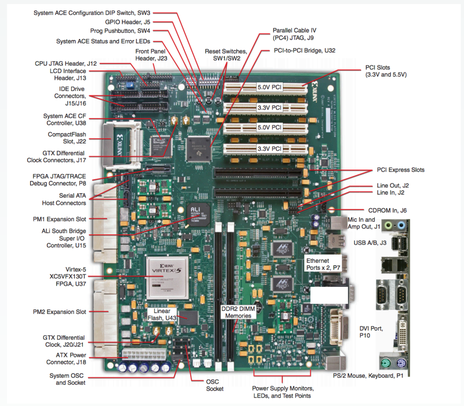
Figure 4


## Workstation Setup

### cFS Applications

The ML510 development board belongs to the Virtex 5 FPGA family that provides optimized high-performance logic, serial connectivity, and embedded signal processing. The ML510 board makes use of the Embedded Development Kit and the Base System Builder to create an embedded processing system. A Xilinx processor can be selected, and the Microblaze soft processor or the embedded PowerPC 440 block can be used. For the purposes of this experiment, both the PowerPC and the Microblaze processors were tested with the Retina cFS application. However, GEONS was tested only with the Microblaze processor for consistency with previous experimentation with the application. The build environment consisted of a Linux CentOS 7 virtual machine, and an ML510 development board with the Microblaze and PowerPC toolchains. cFS (Retina and GEONS applications), CAESAR, and SpaceCube Linux had to be compiled on the Linux virtual machine. cFS and CAESAR Flight Software were compiled within SpaceCube Linux (SCLinux) (on the Linux virtual machine). The flight software and SCLinux ran on the ML510 development board, with a terminal output via a JTAG cable (Figure 5).

Figure 5

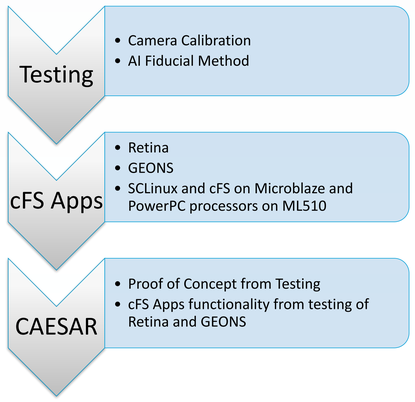



https://www.xilinx.com/support/documentation/boards_and_kits/ug356.pdf

### Development of Camera Procedures

A drone test will be implemented to test mission CAESAR's camera hardware and image processing system through a simulated descent. A commercial unmanned aerial vehicle platform will be developed and will support the imaging tests, along with custom onboard interfaces to support the image capture from camera hardware. A DMK 33GP5000e Monochrome Camera will be used for testing image applications. A camera calibration with a checkerboard pattern will be performed to estimate internal camera model parameters. A survey of artificial reality fiducial markers will be used to measure the local position of markers and to find a location to land.

Figure 6




# 3. Phase 1

## Camera Calibration

### Purpose

Camera calibration is the process of acquiring precise and accurate intrinsic and extrinsic properties of a camera. Intrinsic properties are characteristic of the camera and do not change. Examples of the intrinsic properties relevant to this study include focal length, optical center, principal point and the skew coefficient. The calibration procedure relates world coordinates, camera coordinates and pixel coordinates (camera's natural unit pixels and real world units). 3D world coordinates are mapped to 2D images to find quantities internal to the camera that affect the imaging process including scale skew, and lens distortion. For example, radial distortion results from light rays bending more near the edges of a lens than the optical center, similarly, tangential distortion is when the lens and image plane are not parallel. It is necessary to perform camera calibration to account for image distortion that can impact the quality of images and degree of precision and accuracy in computer vision applications. For the purposes of this study, a calibrated camera will be vital to ensuring the success of the drone mapping operation.

### Methods

Two methods, both with a similar underlying calibration algorithm were compared in this study: the MATLAB Calibration App and the Camera Calibration Toolbox.


### MATLAB Camera Calibration Application

The calibration procedure involves preparation of images, importing the images in the workspace, calibration, evaluation, and further optimization. The calibration grid used in this study is a 9 x 8 checkerboard with an alternating pattern and 30 mm x 30 mm squares. The camera calibrator app uses automatic corner detection with precise sub-pixel localization in the calibration procedure.

The calibration procedure was repeated until desired results were achieved. Evaluation consisted of examining re-projection errors: the distances between the detected and the re-projected points. The world coordinates, as defined by the checkerboard are projected into image coordinates; the re-projected points are compared to the corresponding detected points and the re-projection error is calculated for each image. The extrinsic parameters plot illustrates a camera-centric and pattern centric view of the camera. These views illustrate obvious errors in calibration (pattern being behind the camera). Further optimization is done with the addition or removal of images, modifying the initial intrinsic matrix and radial distortion coefficients, and computing skew and tangential distortion.

The intrinsic matrix is a 3x3 matrix: $[f_x \ 0 \ 0; \ s \ f_y \ 0; \ cx \ cy \ 1]$. $(c_x, c_y)$ represent the optical center or the principal point, s is the skew parameter and the angle between the x and y pixels (ideally this is 0 indicating a 90 degree angle), $(f_x \ f_y)$ is the focal length in pixels and is computed as $f_x = F*s_x$ and $f_y = F*s_y$ where F is the focal length in world units and $[s_x, s_y]$ are the number of pixels per world unit. Radial distortion occurs when light rays bend more near the edges of a lens than they do at its optical center and can be defined by 2 or 3 coefficients for optimization. The skew parameter estimates the angle between the image axes that can be distorted due to imperfections in a camera lens whereas the tangential distortion parameter estimates coefficients that correlate to the orientation of the image plane with respect of the lens (Figure 8). After optimization, the camera parameters can be exported to the workspace for analysis.

### Camera Calibration Toolbox

The camera calibration toolbox does not use automatic checkerboard detection; instead, the user specifies the corners of the checkerboard pattern manually and uses secondary optimization to re-compute and refine detected corners (Figure 7). The rest of the procedure is synonymous with the Matlab calibration app.

Figure 7

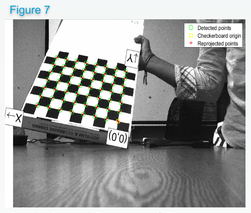
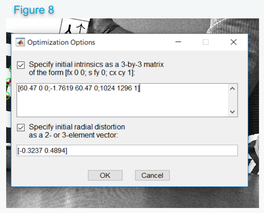



Figure 8




### Analysis and Sources of Error

The camera calibration toolbox produced results with the smallest amount of error: [.80, 1.78] pixels in the x direction whereas the camera calibration app produced a mean re-projection error of about .90 pixels as seen from Appendix A. Both approaches produced a focal length of about 3100 x 3100 pixels which is a reasonable result given a physical focal length of 16.8 mm / pixel pitch (4.8 um) = 3500 pixels. The principal point of ~ 1380x1020 pixels is also reasonable given the image size of 2592x2096 pixels. Although the two calibration approaches fundamentally use the same calibration algorithm, they produce slightly different results due to the extraction of the corners. Possible sources of error include the lighting conditions in the used image data set, and the need for more data points (more squares). The reflection of the light off of the checkerboard causes distortion in the resulting image, and can lead to a larger error in pixels. Furthermore, more data points (more squares) can result in more accurate intrinsic parameters and less distortion.

### Conclusion

Although the camera calibration toolbox approach produced a smaller error in the x direction, both approaches were precise and fairly accurate in estimating the intrinsic and extrinsic camera properties. Future experimentation would involve the use of a larger calibration grid, and more ambient lighting conditions to allow for more precise extraction of corners. A larger data set of images and more squares should also be used to minimize the re-projection error. Given the experimental conditions and setup, the error was minimized through recalibration, and re-extraction of corners. Accurately estimating the camera intrinsic properties is critical in ensuring the success of computer vision applications.

Figure 9

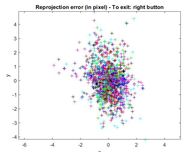



Figure 10

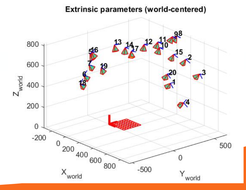



Figure 11

# 4. Phase 2

## cFS Applications

### Retina

#### Purpose

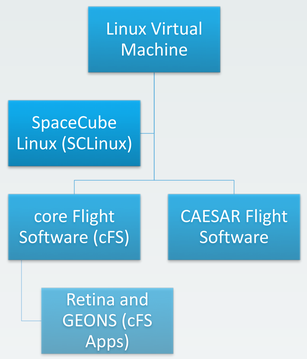
Retina is a cFS application that was developed and modified for image processing on board. Mission CAESAR's camera suite consists of six cameras that will be actively involved in surveying the comet's surface, finding a sampling site, for navigation, and for documentation. The Retina application will play a primary role in interfacing with the cameras to perform on board image processing.

#### Development

The application was tested on the ML510 development board to ensure it ran properly and produced accurate output. A SpaceCube Linux operating system was compiled on the ML510 development board with both the Microblaze and PowerPC toolchain processors. The Microblaze is a soft processor and can be implemented on any Xilinx FPGA. The PowerPC is a hard processor and can be implemented only in virtex devices. The PowerPC processor is preferred for computationally intensive tasks over the Microblaze processor. As a result, the retina application needed to be tested on both the Microblaze and PowerPC processors to determine which processor was best suited for quality results.

#### Results

The Microblaze and PowerPC processors were built and compiled on the ML510 Development board. The application ran most efficiently on the PowerPC processor. On the Microblaze processor, the retina application froze after it ran. Although the application was loaded and created, it did not initialize, process the image, or "sleep". The application is set to sleep after running, however, the "sleep" messages do not appear on the SpaceCube Linux terminal. Because of its computationally intensive nature, the Retina application freezes on the Microblaze processor. However, Retina runs successfully on the PowerPC processor, and the application could be easily exited with a keyboard stroke. The build environment is illustrated:




### GEONS

The GEONS application performs on-board filtering of the processed image that it receives from Retina. GEONS has to be compiled on the Microblaze processor in order to run.

#### Development

The GEONS application was tested and benchmarked with inputs from a Matlab API and a conglomeration of python scripts. GEONS requires commands in the form of ccsds data packets as inputs in order to run. The output is another ccsds file containing the image that is received from Retina. A MATLAB API was used to run a simulation of GEONS, and the python scripts were used to generate inputs to feed to the simulation. The simulated inputs were then fed into the SpaceCube Linux platform running GEONS to get a more accurate idea of how the GEONS application is running. The Matlab API creates a binary file of raw GEONS ccsds commands. The binary file is read by a python script that generates and builds the command/telemetry database. Replay.py reads the packets and commands and converts them to cFE commands that get sent into cFE. A geons_le.ccsds file gets generated that is sent into GEONS, which outputs the result in the console. This process was completed in two main parts: running GEONS as a simulation and afterwards, running it on the Microblaze processor on the ML510 development board in SCLinux.

#### Results

In order to run GEONS as a simulation, a shared library was built This version of GEONS is an independent platform that does not need cFS to run and can be tested from a windows desktop, unlike the real version of GEONS that requires SCLinux and an ML510 development board to run. The GEONS running in the shared library outputs a log that writes the telemetry outputs to a binary ccsds file. After the GEONS simulation was running, the replay.py scenario runs with a big endian 32-bit flag that generates a ccsds file of what GEONS outputted. File commander (a Linux application) reads the ccsds file and outputs a csv file with what GEONS outputted. The Matlab API was used to test GEONS on mission RESTORE. The Matlab API was not modified, however, the replay.py scenario was changed to be compatible with the big endian 32 bit infrastructure on SCLinux. Furthermore, the file was modified to write to a serial uart/tcp port that could be transferred to the Microblaze processor instead of writing to a ccsds file like it was in the simulation; print statements were added GEONS to display the output from GEONS on the terminal window. The file commander framework was also developed for easier access to the GEONS output by reading the ccsds file after it was generated. The GEONS application will be modified to include benchmarking statements for timing how long the processes and functions took to execute. The functions will be surrounded by timer statements to get an accurate reading of how long each function took to execute.

## AI Fiducial Method

Fiducial markers are widely used for optical tracking in many unmanned aerial vehicle operations. Fiducials give insight into the location and orientation of virtual objects. AR environments requires real time pose tracking of devices; tracking fiducial markers is a quick and common strategy to determine this information in changing environments. The markers are encoded with unique identifiers that give accurate information about the 3D pose of an object. The coding system of a fiducial is based on a lexicographic coding system that is robust to lighting conditions. Fiducials can be detected from long rangers, and designed for high localization accuracy allowing for the computation of the 3D position of the fiducial with respect to the camera. A fiducial tracking system will serve two main purposes in the initial drone testing. The fiducial markers and software will be used to initially survey the quarry site to provide the relative position and orientation information to the target and quarry. Furthermore, the fiducials will be used to determine real-time relative orientation information during a test flight, that will be more precise and accurate than the information received from the drone. The software will be hosted on the ground system, and the lens flying on the drone will capture images in real time. The pose between the drone platform and the ground station, as well as between the camera and the fiducials on the ground and ground station will need to be determined (Figure 11).

## References

Information
https://www.nasa.gov/press/goddard/2015/march/nasa-goddard-releases-open-source-core-flight-software-system-application-suite-to
http://caesar.cornell.edu/overview/
https://spacecube.nasa.gov/pat8484509.pdf
https://spacecube.nasa.gov/pat8484509.pdf
https://solarsystem.nasa.gov/missions/caesar/in-depth/
https://www.researchgate.net/figure/Basic-workflow-of-an-AR-application-using-fiducial-marker-tracking_fig1_216813818?_sg=z97PBNtk4etQqNx80UhTtgHdfXXfio6a2qrJHbroOU78ODT8fsjpIW-GY3IjpVmegam8pVr5wAaKGGdc4Esw