JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 10, No. 8, August 2013

# Spacecraft Attitude Determination System Using Nano-Optical Devices and Linux Software Libraries

Christopher M. Shake,* Kelsey Saulnier,† and Riccardo Bevilacqua‡
Rensselaer Polytechnic Institute, Troy, New York 12180

DOI: 10.2514/1.I010049

**This paper presents a new spacecraft attitude determination system based on small optical devices and Linux-based software. This technology intends to support nanosatellite operations by providing low-cost, low-mass, low-volume, low-power, and redundant attitude determination capabilities with quick and straightforward onboard programmability for real-time spacecraft operations. The chosen commercial-off-the-shelf optical devices perform sensing and image processing on the same circuit board, and they are biologically inspired by insects' vision systems, which measure optical flow and/or track objects while navigating the environment. The firmware on the devices is modified here to enable communication with PC/104 form-factor embedded computers running RealTime Application Interface for Linux. Algorithms are developed for operations using optical flow mode and point-tracking mode, and an application programming interface, along with Simulink® S-functions, is created. The performances of the proposed system, used in optical flow mode, point-tracking mode, and a combination of the two, are assessed using a spacecraft simulator at Rensselaer Polytechnic Institute, and they are compared with measurements from the PhaseSpace® motion-tracking system.**



## I. Introduction

NANOSPACECRAFT hold the potential to replace expensive big platforms for Earth-orbiting missions, as well as for outer planet exploration and colonization, due to the advantages of multivehicle operations versus a centralized approach [1]. The benefits include on-orbit reconfiguration, ease of replacement of a single malfunctioning unit (e.g., thanks to the use of commercial components versus space qualified expensive systems), and the possibility to develop and launch the spacecraft in a short timeframe. It is also worth mentioning the educational impact that platforms such as CubeSat have had in recent years, when the space community has experienced a rapid increase of university-developed space missions [2]. An additional appeal of working with small vehicles is the possibility of testing their guidance, navigation, and control (GNC) using on-the-ground spacecraft simulators. This testing represents the missing link between computer numerical simulations and on-orbit operations by partially reproducing the dynamical conditions of space flight in a laboratory environment. By its nature, on-the-ground testing is a low-risk low-cost high-return tool for spacecraft operation certification.[§]

Attitude determination is an important aspect of satellite operations, with higher-accuracy systems allowing for different types of scientific missions. The current state of the art for CubeSat-sized satellites is relatively low accuracy and generally purpose built for each mission. The most common methods used on CubeSats involve sun sensors, Earth sensors, and/or magnetometers for initial and absolute state determinations, then propagation of the state with rate gyro information from an inertial measurement unit, resulting in accuracies in the $\sigma = 1$–$5$ deg range after filtering [3]. Higher-fidelity systems like star trackers can provide 0.01 deg accuracy, but they are currently too large to fit on CubeSats, or at least take up nearly the full payload space of a one-unit (1U) CubeSat while requiring more power than can be typically generated [4].

In an effort to support the nanospacecraft community, this work presents the development and on-the-ground experimental validation of an attitude determination system (ADS) based on commercially available bio-inspired optical devices and real-time application interface (RTAI) Linux-based Simulink software libraries. The main drive for this work is to provide nanospacecraft developers with an ADS that can be straightforwardly reproduced using commercial off-the-shelf (COTS) hardware and quickly (re)programmed. In addition, the cost, size, weight, and power consumption of such a system should be minimized for potential implementation on very small vehicles. The preceding illustrated features also allow for redundancy; that is, multiple sensors could in principle be operated to improve navigation accuracy and/or for safety in the event of malfunctions, as they will take up a small enough portion of the respective spacecraft budgets to allow for multiple units.

Using two nonaligned optical sensors, it is possible to reconstruct the orientation of the platform hosting the sensors themselves by observing a known environment (i.e., a star pattern). The chosen optical devices are the CYE8 sensors from Centeye, Inc.,[¶] which consist of a low-resolution camera connected to an Atmel AVR-series microcontroller that performs the image processing. The firmware used on the microcontrollers is open source, allowing modification and addition of image-processing algorithms. This firmware is available online[**] under an open-source license. The device choice is justified by ease of commercial availability, flexibility of software modification, small size, and low mass and power requirements. Though the sensors have limited resolution and processing power, this paper is intended to demonstrate attitude determination capabilities using small optical sensors that could be used on a nanosatellite, and the methodologies illustrated here can scale to higher-accuracy devices in the same-size family.

The chosen software infrastructure is based on RTAI Linux [5], since it allows for custom design of real-time software for a variety of devices, as compared to the proprietary nature of real-time operating systems (OSs) like XPCtarget [6]. In addition, RTAI Linux is one of the available target OSs for the Simulink Coder™ (formerly Real-Time Workshop™), a software tool that generates real-time executables from Simulink models.

Received 8 May 2012; revision received 15 October 2012; accepted for publication 17 December 2012; published online 9 August 2013. 

*Graduate Student, Department of Mechanical, Aerospace, and Nuclear Engineering, JEC 1034; shakec@rpi.edu.
†Graduate Student, Department of Electrical, Computer, and Systems Engineering, JEC 1034; saulnk@rpi.edu.
‡Assistant Professor, Department of Mechanical, Aerospace, and Nuclear Engineering, JEC 5048; bevilr@rpi.edu. Member AIAA.
§Data available online at http://ssco.gsfc.nasa.gov/facility.html [retrieved 7 May 2012].
¶Data available online at http://centeye.com/ [retrieved 15 October 2012].
**Data available online at http://code.google.com/p/centeye-8bit-sensor/ [retrieved XXXX].

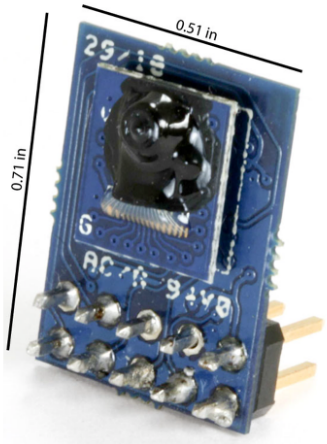



**Fig. 1 CYE8v2 device with lens and pin header.**



This last feature led to the choice of Simulink S-function development. Using RTAI Linux and Simulink libraries enables nanospacecraft developers to quickly (re)program the devices if needed, and to easily integrate them into existing GNC algorithms. Last, but not least, the open-source nature of the Linux solution allows for public access and contributions from all over the world. More reasons for the choice of RTAI are presented in previous work [6].

Experimental validation of the ADS is performed employing a robotic spacecraft simulator at the Advanced Autonomous Multiple Spacecraft (ADAMUS) laboratory at Rensselaer Polytechnic Institute [7]. In particular, the ADS data are compared to the data from a commercial IR light-emitting diode (LED)-based motion-capture system from PhaseSpace, Inc. The motion-capture system is characterized by high accuracy, and it is used to provide the reference truth attitude. The simulator's hardware is designed solely for GNC on the ground testing, and as such has not been tested for thermal, vacuum, or radiation resistance, and neither have the COTS sensors used herein.

This paper presents original contributions in the following areas: 1) development of a nano-ADS with a potential for implementation on nanospacecraft; 2) development of the software infrastructure to seamlessly (re)program the ADS; 3) experimental validation of the ADS in a laboratory space-simulated environment operating in optical flow mode (angular velocity computation), point-tracking mode (quaternion computation), and a combination of the two modes; 4) conclusion that the proposed ADS has the potential to replace both star trackers and gyroscopes on nanospacecraft; and 5) detailed instructions to reproduce the system, and online software available to the nanospacecraft community.

This paper is organized as follows. Section II describes the optical devices and their related developed software. Section III contains information about the PhaseSpace motion-capture system and associated S-function. Section IV is an overview of the spacecraft simulator of the ADAMUS laboratory. Section V provides experimental validation results, and Sec. VI presents the conclusions.

Appendices are included with detailed steps used to reproduce all presented data and results. The source code and the Simulink library are downloadable from the MathWorks® file exchange. The source code for the optical devices firmware is available from the centeye-8bit-sensor Google Code repository (see footnote [**]) with modifications available from the authors.

## II. Optical Devices

The Centeye CYE8v2 model, shown in Fig. 1, was used for the presented research. The device consists of a small optical chip connected to a microcontroller that communicates with other devices over the interintegrated circuit (I2C) two-wire interface. These sensors have not been space qualified or used in any previous space mission. With a target use on CubeSats flying in low Earth orbits, radiation is not as much of a concern as higher-altitude orbits. The relatively simple electronics are based around Atmel microcontrollers, which have a relatively large process size (electron pathway width) compared to state-of-the-art desktop processors and produce far less heat, which makes them inherently more suitable for space even before radiation hardening. Recent missions using non-space-qualified hardware in CubeSats have been successful, notably STRaND-1 with a smartphone as the central avionics [8]. While in use, the sensors do not require a heat sink and the surface temperature of the Atmel chip rises only slightly above room temperature, showing good potential for in-space thermal feasibility.

### A. Hardware

Technical specifications for the sensors are presented in Table 1. The vision chip provides a binning ability where pixels in the two, four, or eight neighboring rows and/or columns can be read as a single larger pixel. Using this feature creates smaller images with less noise that are used in the interests of memory use and processing speed. The brightness of each pixel is available as an analog signal in the vision chip itself. When reading

**Table 1 CYE8v2 specifications**

| Parameter | Value |
|---|---|
| Resolution | $64 \times 64$ pixel, monochrome |
| Camera output | Log-response photodiodes, onboard 8-bit ADC |
| Image reading | Sequential or direct pixel addressing |
| Microcontroller | Atmel AVR ATmega644P |
| Interfaces | I2C/system management bus (SMBus) and serial peripheral interface bus (SPI)[a] |
| Processor speed | 8 MHz (default), 20 MHz (high-power mode) |
| Memory | 64 KB flash, 4 KB static RAM, 2 KB EEPROM |
| Output speed | 100 kHz (SMBus), effective 6 kB/s shared between all sensors on the SMBus interface |

[a]SPI not yet implemented in firmware.

**Fig. 2 CYE8 vision chip with lens.**

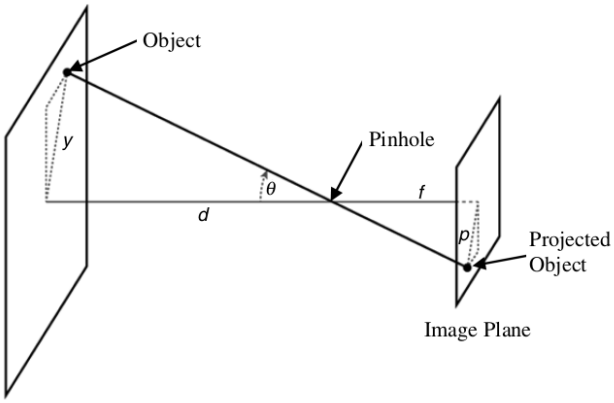



**Fig. 3 Pinhole image-projection diagram.**



that value, the onboard analog-to-digital converter (ADC) uses user-defined upper and lower voltage limits to produce a scaled 8bit digital value that spans a subset of the available analog range, defined by the user as the expected brightness range of the scene. With the limits on memory, the default operation mode uses a binned $8 \times 8$ image so that a fixed-pattern noise mask can be stored in the electrically erasable programmable read-only memory (EEPROM) and all working variables can be stored in RAM during processing.

The lenses on the CYE8 vision chip (Fig. 2) exploit the entire photosensitive area of the chip, and they allow for a full $64 \times 64$ image to be used. Pinhole optics are also available that create a pinhole image over a $16 \times 16$ pixel area of the sensor, trading lower resolution for a wider field of view and no added optical distortion. For the data obtained in this work, the pinhole optics were used and an $8 \times 8$ pixel image was used from the center of the pinhole, giving horizontal and vertical fields of view of 22.6 deg. With the log-response pixels, a wide range of intensities can be imaged, with the only change necessary for going from sun sensing to point lights on a foreign satellite being the exposure time brackets. If the sensor is set to track stars and the sun crosses the field of view, it will register as a very bright saturated point, but it will not damage the sensor and normal tracking will resume as soon as it is outside the field of view.

The firmware provided with the sensors has been released by Centeye, Inc., under an open-source license. The provided C code communicates with a host computer over a two-wire (I2C) interface. As part of this work, modifications were made to use the higher-level SMBus subset of the I2C protocol that the researchers' PC/104+ hardware supports, available in revision 7 of the firmware on the Google code site.[††]

### B. Angular Measurements

When using images taken by a camera that is moving with pure rotation in a stationary environment, the images can be thought of very generally as snapshots of the inside of a large sphere, much like the visible sky is approximated as a sky sphere. When determining the motion of the rotating body that the camera is attached to, angular values must be used. However, the image that the camera produces is the projection of the curved spherical surface onto its flat image plane, so the angular distance between two objects seen as $x$ pixels apart in a corner of the image is different than when they are seen as the same $x$ pixels apart but in the center of the image.

To accurately measure angular positions (and thus movements), the image must be distorted appropriately to extract the measurements. The pinhole projection diagram is presented in Fig. 3. Any object can be thought of as being on a flat surface parallel to the image plane, with $p/f = y/d$, where $p$ is the distance from the center of the image to the point in the image plane, $f$ is the focal length, $d$ is the distance from the focal point to the object, and $y$ is the distance from the camera center line to the object. From this, $\vartheta = \tan^{-1} y/d = \tan^{-1} p/f$, where $\vartheta$ is the angular coordinate of the object. As the physical objects being imaged move toward and away from the camera, the angular location does not necessarily change with respect to the camera central axis, allowing the angular coordinates of any point on the image to be found without knowing $y$ or $d$, as long as the focal length and pixel pitch (physical distance between pixel centers) are known. Using only the angular value also removes the need to approximate the object being on a plane parallel to the image plane, as the distance to the camera is no longer needed to locate the object. Using a camera with a lens instead of a pinhole does not change the transformation as long as the lens has negligible distortion and the focal length is known.

[††]Data available online at http://code.google.com/p/centeye-8bit-sensor/source/detail?r=7 [retrieved 15 October 2012].

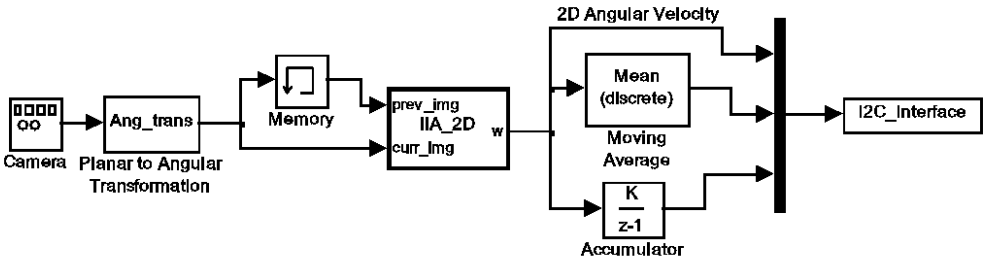



**Fig. 4 Optical flow block diagram.**

### C. Firmware and Algorithms

Two different aspects of the images are captured by the firmware: optical flow and point-tracking. For navigation relative to generic objects with no predefined characteristics, optical flow is used to measure the rate at which a texture moves in front of the camera. This mode was available in the CYE8 sensor firmware when the sensors were purchased. This rate represents an angular velocity because both the foreign object size and its distance are not inherently known, but the pixel pitch and focal length are known.

Point-tracking is the method of identifying a point of interest in the image and tracking its location between frames. The algorithm for this mode was developed for this paper. For the point-tracking application, the authors assume that points of interest are stars or light beacons on target spacecraft, and the detection scheme is designed with that in mind.

The original firmware uses the two-dimensional (2-D) variation of an optical flow algorithm from Srinivasan called IIA [9]. The designer of the sensors previously published a dissertation on optical flow sensors for use with micro air vehicles [10], including a thorough review of available algorithms and justification for the case to implement IIA. In this work, we retain that algorithm for the optical flow mode.

Figure 4 is a block diagram representing the data flow for the optical flow sensor mode. IIA produces a value representing 2-D motion in the image plane between every two frames, in units of pixels per frame. Knowing the physical size of each pixel in the vision chip, this measurement is converted directly into distance units, and with the approach given above in Sec. II.B, angular motion is found. Since the flow is computed faster than it can be read out, internal accumulators are used to record cumulative motion for odometry. This cumulative data, along with the instantaneous flow and a rolling average, are read from the sensor and sent to the host computer. The discrete derivative of the cumulative flow is taken in a real-time environment to give a more precise angular velocity measurement that, by design, allows for variable read rates and smoothes out zero-mean higher-frequency noise. This algorithm runs at approximately 3.4 kHz on the sensors on the $8 \times 8$ pixel image, although the data are read out at a fraction of that speed due to constraints from the SMBus interface. Given that the IIA algorithm assumes a motion of up to one pixel displacement per computation, with a 22.6 deg horizontal field of view, the maximum flow rate is theoretically over 150 rad/s, much faster than could be tested.

For the current research, an additional algorithm was implemented for point finding and tracking. The difference in brightness of a point and its nearest neighbors (curvature) increases at object edges and corners in the image, described by [11]. It is known that the points of interest in the images for this application are bright, so a weighted combination of the brightness and curvature is used to identify the points with acceptable accuracy when applied to both a simulated image sequence with known correct positions and visual comparison by overlaying the point locations on a live image stream from the CYE8 sensors.

In the current research, the point lights represent stars or possibly a bright beacon on another spacecraft; in both cases, the rest of the image is not expected to move relative to the point target, and the test environment for this proof of concept experiment is designed specifically with a high-contrast ratio. The reason for including curvature in this algorithm is that a point-tracker using only brightness values returned a high percentage of false points due to image noise. By varying the relative weightings of brightness and curvature manually for the algorithm, as it ran live on the CYE8 sensors, and overlaying the located point on the video, weights were found that returned point locations matching those determined by visual inspection. The method developed is presented as Algorithm 1.

The identified points in each frame are compared to the locations of the points in the previous frame based on distance. The points are considered the same if a new point is found within a specified radius of the previous location. This criterion for matching is rudimentary, but due to the frame rate being high relative to the image motion, it does not lose any points in this application as long as they are still visible in the frame. Rates up to 0.5 rad/s have been used during tests, and the sensors have not yet exhibited any loss of tracking. This rate is already five or more times faster than fast, commercially available star trackers running in coarse mode.

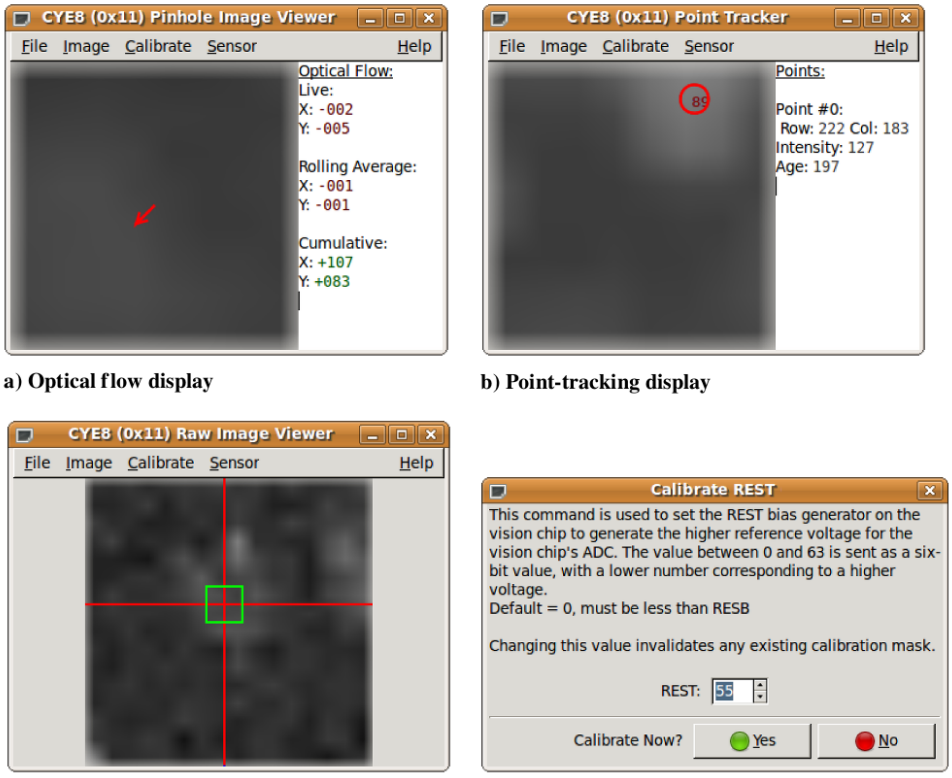
This algorithm has been found to track points with accuracy in the one-eighth-of-a-pixel range for a single bright LED on a dark background, using an $8 \times 8$ image from the CYE8 sensors, giving an angular resolution of just under 19 arcmin. Because of the additional memory and computational resources required for this point-tracking algorithm compared to the optical flow method, it runs at a lower frequency of approximately 1.2 kHz, which is still faster than the sensor can send data back to the host computer. The internal code is written to be able to track motion of up to 2 pixels per frame for a feature point, which results in a theoretical maximum tracking rate of almost 120 rad/s, much faster than could be tested. Even at 1 pixel displacement per frame, a feature point will cross the entire 8 pixel, 22.6 deg, horizontal field of view in under 7 ms, so the theoretical limit is considerably higher than would ever be encountered in a feature tracking scenario.

### D. Sensor Interface and Calibration

The Linux kernel includes an I2C interface that can be accessed using low-level system calls. To make it easier to use the bus in user applications, a package known as I2C-dev is available that represents the bus as a character device on the system, similar to a hard drive or a

**Algorithm 1 Point finder**

1. Calculate curvature $c$ at each point in the image $c_{i,j} = 4b_{i,j} - (b_{i-1,j} + b_{i+1,j} + b_{i,j-1} + b_{i,j+1})$ from brightness $b$.
2. Add curvature $c$ and brightness $b$ with relative weights to create matrix $M$.
3. Locate maximum values of $M$ and $P_k$, not counting points within a minimum radius $\rho_1$ of each other
4. Find the center of each point $r_k = \frac{\sum M_{i,j} r_{i,j}}{\sum M_{i,j}}$ for $i$, $j$ within a radius $\rho_2$ of $P_k$, where $r$ represents the image coordinates at location $i$, $j$ in $M$.




**a) Optical flow display** **b) Point-tracking display**

**c) Raw sensor output with pinhole location** **d) Sample calibration dialog**

**Fig. 5 Calibration GUI screenshots; further description follows in the text.**

memory card.[‡‡] Instructions for installing the I2C-dev interface are included in Appendix B. A set of C functions is defined by this package that calls the internal kernel functions for interacting with the bus.

Using this C interface, a library of functions was written for interacting with the CYE8 sensors. Using this function library, a graphical user interface (GUI) was written for both monitoring and calibration purposes. While there are many frameworks available for creating GUIs, GTK+[§§] was chosen because of its simplicity and portability. The intent of the program was to allow for quick monitoring of sensor operation and calibration of commonly modified values, so it is not a fully featured application for managing all aspects of the sensors. For all operation modes, the GUI displays the live image from the sensor used for calculations at a 10 Hz refresh rate. It reads the flow or point location information corresponding to the image and shows a graphical overlay on the image with that information, as well as the numerical values in an adjacent text box.

An overview of the available monitoring modes can be seen in Figs. 5a–5c. Note that the vision chip in use at the time of the screenshots was using the pinhole optics instead of a lens, which was what the sensors came with off the shelf; the resulting data are no different for the $8 \times 8$ pinhole image compared to an image using a lens with pixel binning to create an $8 \times 8$ image. In Fig. 5a, the IIA optical flow algorithm is in use, showing the motion in the field of view. An arrow is superimposed on the image to show the direction and magnitude of the flow, and the raw numbers can be seen to the right. Figure 5b shows the point-tracking mode, with a circle superimposed on the centroid of the located point and additional information to the right in text form. The number in the circle is an internal identification number shown for debugging purposes. The internal identification number changes whenever the point cannot be matched between subsequent frames. In the firmware, the points are matched between frames based on proximity, so jumps across a large distance will register as a different point. Figure 5c shows the raw $64 \times 64$ image, sampled every 4 pixels, to assist in locating the pinhole on the vision chip. The crosshairs show the current center of the pinhole, and the surrounding box shows the field of view used for flow computation and point finding.

The dialog in Fig. 5d is representative of the interface for changing many of the calibration options for the sensor, with instructions and an explanation of the parameter effects. A full list of configurable parameters and the effects of each are available in Centeye documentation,[¶] and other similar dialogs have been omitted here for brevity. The parameter that does not have documentation from Centeye is the dialog for changing the intensity and curvature weights for the point finding algorithm (as it was developed in house).

Dropdown menus (Fig. 6) contain options for various common commands such as saving a copy of the image, reading from a different sensor, changing sensor operation mode, and calibration. For each of the menu options, a dialog box pops up with a description of the setting and what the valid range is for calibration values, as well as any additional steps needed.

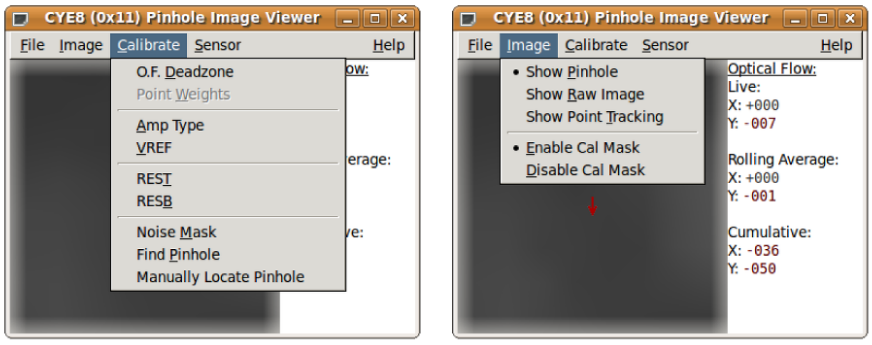
### E. S-Function

Simulink from MathWorks is a graphical programming tool based on blocks that, when used with Simulink Coder, can generate real-time executable programs from models that can run on a real-time OS (target). While primarily used for numerical simulations in the desktop environment, custom blocks can be written in C, C++, and FORTRAN that can interact with hardware, and thus be used to control live systems. These blocks, called S-functions, link to functions that can contain user code that the simulation engine calls at various events. Detailed

[‡‡]Data available online at http://www.kernel.org/doc/Documentation/i2c/dev-interface [retrieved 2013].

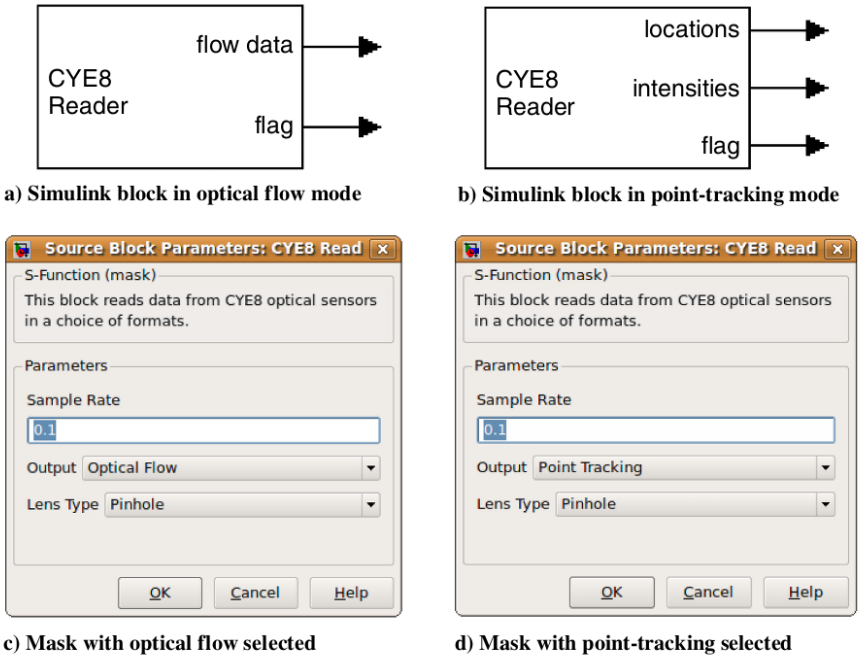
[§§]Data available online at http://www.gtk.org/ [retrieved 2013].

[¶]Data available online at http://code.google.com/p/centeye-8bit-sensor/source/browse/trunk/docs/AVR8_v2_Firmware-Instructions_Application-Prototype-1_2010-08-18.pdf [retrieved 2013].




a) Calibration options menu b) Image display menu

**Fig. 6 CYE8 GUI menu examples.**




a) Simulink block in optical flow mode b) Simulink block in point-tracking mode

c) Mask with optical flow selected d) Mask with point-tracking selected

**Fig. 7 Block and mask for CYE8 reader in both modes.**



instructions for how to use these blocks are included in Appendix C. A library for robotics applications was developed for previous research [6], and in this work, more blocks are developed and presented as a separate library.

The first block in this new library (Fig. 7) is an implementation of the CYE8 SMBus interface, which reads data from a set number of sensors with predefined bus addresses and provides two different outputs to the rest of the model depending on the sensor operation mode. When in optical flow mode, the block reads the cumulative optical flow and it outputs the current cumulative flow, accounting for sensor output integer overflows, the current flow (defined as the difference between the current and previous accumulated flow), and a debug flag indicating if the data were successfully acquired from the sensor or retained from the previous read. In the point-tracking mode, the block takes the image coordinates provided by the sensor, and then it calculates and outputs the three-dimensional (3-D) unit vector pointing toward the tracked point in the image frame. Additional debugging information is also included, such as point intensity and the internal identification (ID) number. This block can then be used in any Simulink model as a data source in a model for use on a Linux target. When the reference implementation is on a real-time target and is set to read the sensors at a rate that the SMBus limitations can handle, the inclusion of the "valid data" flag as an output allows the block to provide what is effectively a clock pulse to the rest of the simulation. The block has a different number of outputs depending on the sensor mode desired, demonstrated by Figs. 7a and 7b. The block mask (Figs. 7c and 7d) takes the sample rate, desired output type, and type of optics on the connected sensor(s) as parameters.

## III. PhaseSpace System

The PhaseSpace motion-capture system is a multicamera system that tracks the location of multiple active LED beacons in the experimental area. These LEDs blink a unique binary ID so they can be identified by each camera. The system has been tested to have comparable accuracy to the popular Vicon vision-based navigation system by Davis et al. [12].

### A. Hardware

The cameras in this system (Fig. 8) consist of two perpendicular linear imagers to locate the LED beacons in the equivalent of a $3600 \times 3600$ pixel image. The system installed in the ADAMUS laboratory consists of eight cameras positioned around the experimental area, each with a 60 deg field of view, as seen in Fig. 9.

**Fig. 8 PhaseSpace camera (from www.metamotion.com [retrieved 2013]).**

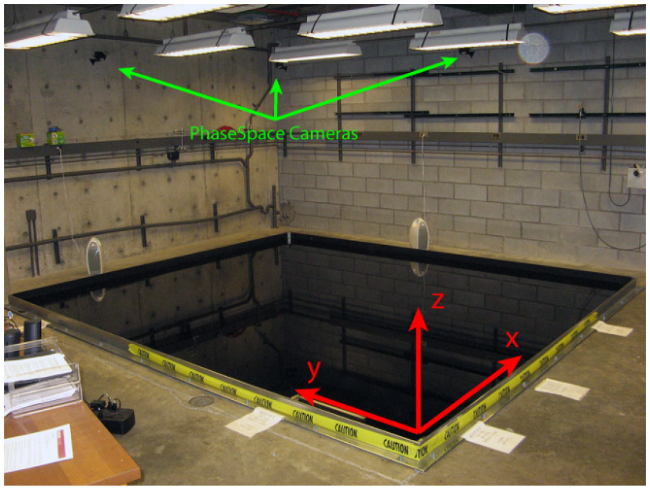



**Fig. 9 Experimental space, with cameras and reference frame.**

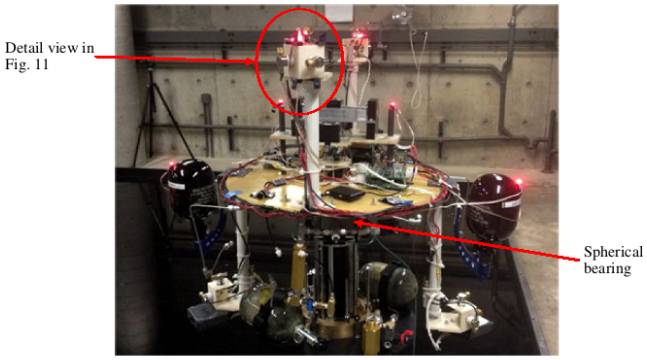



**Fig. 10 Spacecraft simulator with PhaseSpace beacons illuminated.**

The spacecraft simulator mounts six LED beacons (Fig. 10) and is free to rotate in three axes on a spherical bearing. Also visible in Fig. 10 are four CYE8v2 optical sensors, the dark chips near the top center of the image. A detailed view is provided in Fig. 11, showing how the sensors are attached around the thrusters.

### B. S-Function

The second new Simulink block is an interface for the PhaseSpace motion-capture system (Fig. 12a). The PhaseSpace system's server has a network visible interface using a proprietary protocol with a shared dynamic library that can be linked to in Linux to read data from it. The block functions similarly to the CYE8 sensor block, polling the system and outputting a data array and an update flag. The data consists of seven values, containing the spatial position and an attitude quaternion. The C source code for the S-function currently must be modified and recompiled for tracking different objects, as the location of each LED on the rigid body is hardcoded in the source. In the future, the block could be modified to take those values as parameters in the mask. The block mask (Fig. 12b) contains parameters for the Internet Protocol (IP) address of the PhaseSpace server and the sample rate.

PhaseSpace provides a standalone GUI for the system that reads the same data as the S-function, and both have been run side by side to verify the output of the Simulink block. The S-function reads the exact same data as the proprietary GUI, with the only difference being the parent software environment.

## IV. Spacecraft Simulator Overview

The spacecraft simulator in the ADAMUS laboratory at Rensselaer Polytechnic Institute is a novel 6-degree-of-freedom (6-DOF) system (Fig. 13 and [7]). This system consists of an attitude stage (component listing in Table 2) supported by a spherical air bearing, resting on a vertical

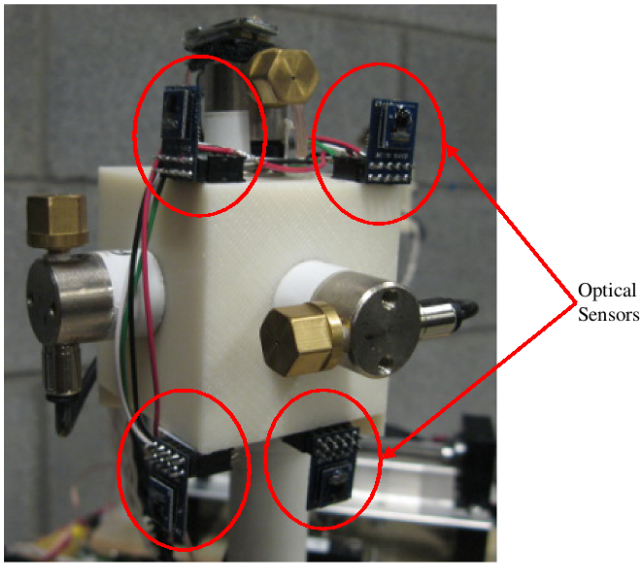
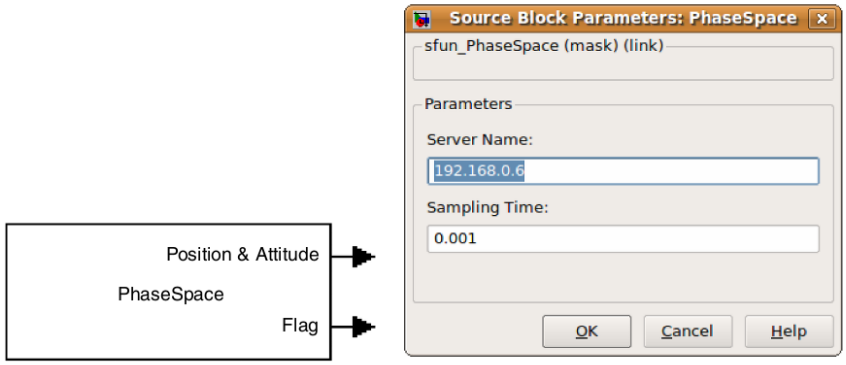



**Fig. 11 Sensor mount detail.**




**a) Simulink block** **b) Mask for block in a**

**Fig. 12 PhaseSpace S-function block and mask.**

counterbalanced base that translates over an epoxy floor (Fig. 9). The onboard computer is PC/104 form factor and runs RTAI Linux, which is a patch for the Linux kernel that allows for hard real-time applications by preempting kernel calls, letting programs run based on wall time instead of CPU cycles [5]. Instructions for installing and using RTAI are included in Appendix A.

Communications between the onboard computer and other computers in the laboratory network (including desktops and future additional spacecraft simulators) use IEEE 802.11 Wi-Fi wireless networking, using a D-Link Pocket Wireless adapter connected to the wired ethernet port on the PC/104. This configuration is used instead of a universal serial bus (USB) Wi-Fi adapter because it was found in previous research that RTAI does not handle USB devices without dropping out of the real-time task to access them [6]. This also has the advantage that no additional driver or configuration is needed for the wireless adapter since the PC/104 communicates through the wired port. Using Wi-Fi instead of other communications options makes it possible for any other computer to communicate with the simulator using Transmission Control Protocol/Internet Protocol (TCP/IP) or User Datagram Protocol (UDP), so the system can be accessed remotely over SSH (Secure Shell) [***] and query other networked servers for data with various protocols. In the current work, TCP/IP is used as the control channel by logging into the PC/104 with SSH, while live data are streamed over UDP. Previous work [6] also demonstrated that this specific wireless adapter and communication method is robust with no significant delay or data loss being introduced, at speeds up to 100 Hz.

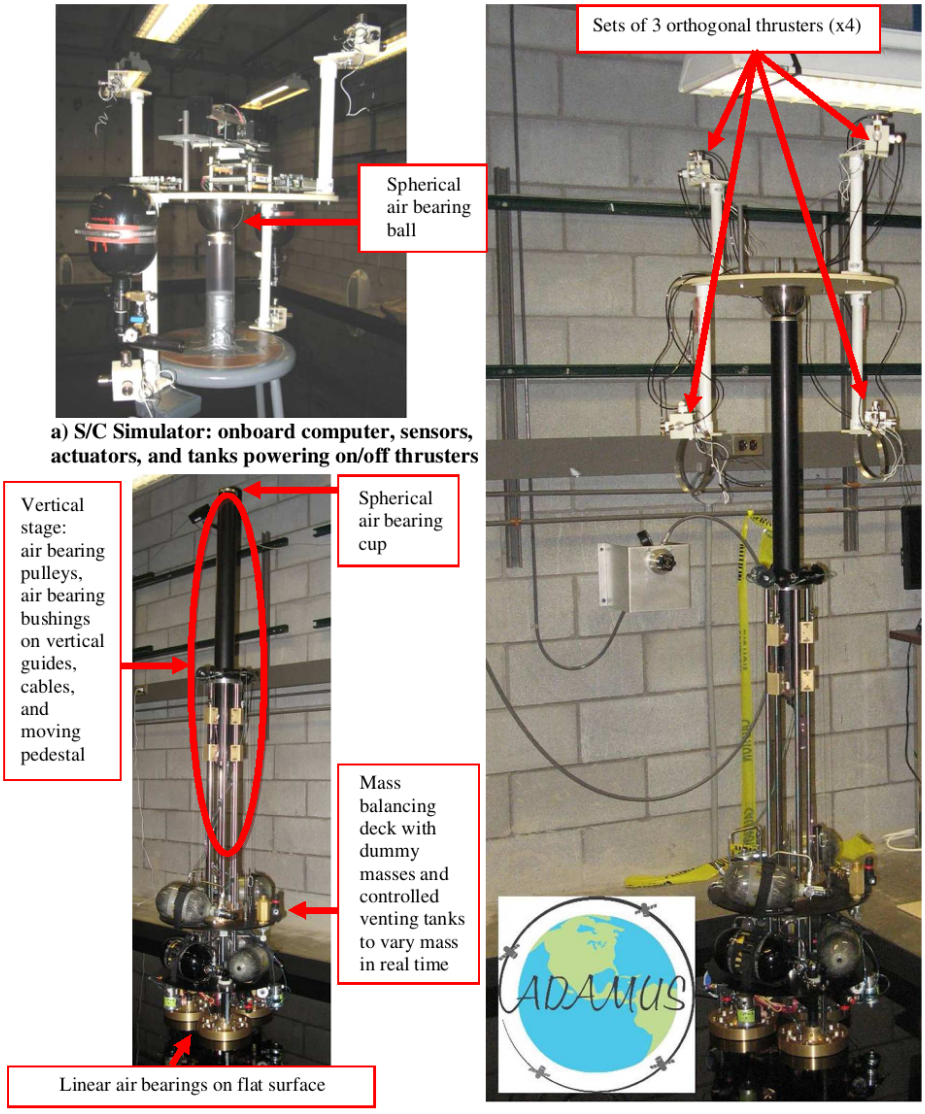
## V. Experimental Results

This section presents data obtained using the previously described ADS. The comparison against data from the PhaseSpace motion-capture system is used to assess the performance of the proposed ADS. The tests use the same hardware and setup in both sensor operating modes, i.e., point-tracking and optical flow modes.

The roles of the three computers involved in the test are shown in Fig. 14. The solid lines indicate wired data connections between the PhaseSpace computer and cameras, dashed lines are wireless links between the computers and router, and dotted lines are used for labels. A desktop computer running Linux is the operator station, which is running MATLAB® and Simulink. The models are created on the desktop computer, and from there, C code is generated with Simulink Coder and the executable is uploaded to the PC/104 to run. During the test, the desktop runs another Simulink model that receives data live from the PC/104 to show live-updating figures and plots to monitor the operation. This monitoring function is not required for the PC/104 system to operate, but it is currently required for saving simulation data for later analysis. Instructions for this procedure are included in Appendix D.

The PC/104 runs the RTAI patched Linux kernel and is used remotely with SSH over the laboratory Wi-Fi connection. If it were using only data from onboard optical sensors instead of also comparing with position information from PhaseSpace, it would be entirely self-sufficient once the

***Data available online at http://www.snailbook.com/protocols.html [retrieved 2013].




**a) S/C Simulator: onboard computer, sensors, actuators, and tanks powering on/off thrusters**

**b) 6DOF stage**

**c) S/C mounted on the 6DOF stage**

**Fig. 13 ADAMUS spacecraft simulator: a) attitude stage (spacecraft), b) translation stage (it allows 3-D translational motion, plus it hosts the cup for the spherical attitude motion), and c) together.**

controller program is started, and not requiring external input or even network connections (see Fig. 14) except to start or stop the program. For the data presented here, the motion comes from manual manipulation of the attitude stage on its spherical bearing without translational motion. Because of this hand operation, there is no truth data for the motion, so the PhaseSpace system with its known accuracy is used as a reference. As discussed previously, the maximum theoretical tracking rate of the sensors is much faster than any motions possibly used in the experiment, so the rates given by hand were not artificially sped up or slowed down. As the image acquisition itself is faster than the algorithm as a whole, the image is estimated to be acquired at 5 kHz or faster, at which rate any possibility of movement-induced light streaking across multiple pixels is assumed to be so low as to be negligible.

A bird's eye view of the simulator setup is illustrated in Fig. 15: two optical sensors are mounted perpendicular to each other on the attitude stage, and two lights ("stars") are fixed at known locations in the laboratory.

**Table 2 Attitude stage components**

| Component | Model | Company |
|---|---|---|
| Thrusters, 12x | EH2012 | Gems Sensors And Controls |
| Battery management system | MP-04R | OceanServer Technology Inc. |
| DC–DC converter | DC123R | OceanServer Technology Inc. |
| Li-ion batteries, 2x | ND2054 | Inspired Energy® |
| Motion-tracking system | Impulse | PhaseSpace Inc. |
| Compressed air tanks, 2x | Ninja 4550 | Ninja Paintball |
| Relays module | IR104-PBF | Diamond Systems |
| Wireless-fidelity (Wi-Fi) module | DWL-G730AP | D-Link |
| Onboard computer | ADLS15PC | Advanced Digital Logic |
| Motor controller card | DCM-2133 | Galil Motion Control |
| Motor drives | SDM-20242 | Galil Motion Control |
| Mass balancing motors, 3x | 35F4N-2.33-0xx | Haydon Kerk |

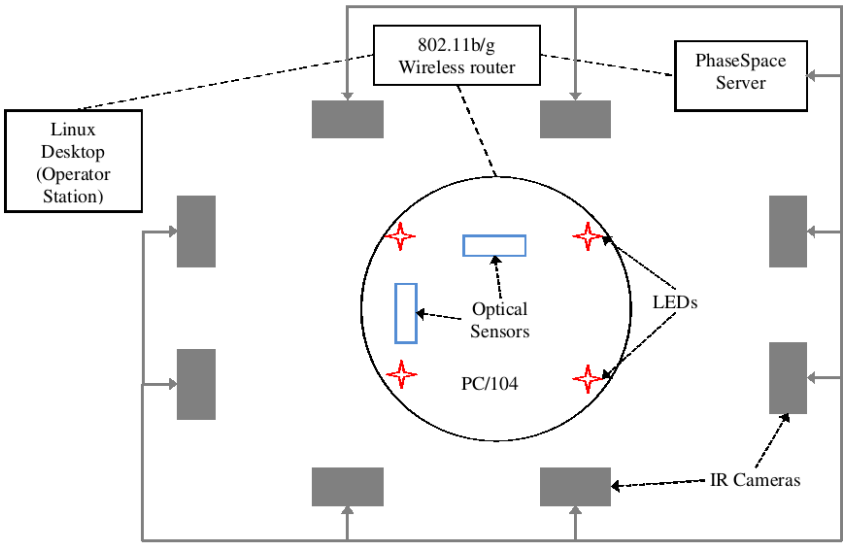



**Fig. 14 Experimental setup representative sketch: PhaseSpace IR cameras and related image-processing server, operator desktop, and spacecraft simulator mounting optical devices and motion-tracking LEDs.**

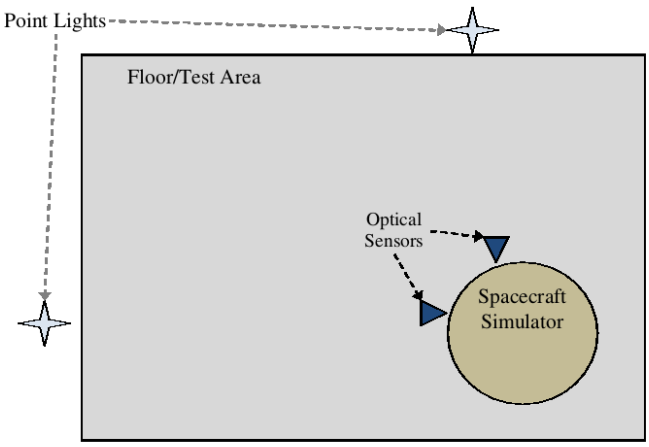



**Fig. 15 Representative sketch of optical sensor and star (point light) locations for the experiments.**

LEDs are mounted on the spacecraft simulator attitude stage for the motion-capture system. This LED system consists of a controller and integrated battery pack provided by PhaseSpace that produces the different flicker patterns to uniquely identify each light. The multicamera system records the data, which are then processed and provided over Wi-Fi from the base station server to whichever computer needs the data: in this case, the PC/104. As mentioned in the preceding section, the PhaseSpace system has been shown to be accurate and is used as the truth model.

### A. Optical Flow to $\omega$ Computation

The first sensor operation mode tested was the optical flow computation. Two CYE8 sensors were mounted on perpendicular outside faces of the attitude stage, and the point light sources were turned off. The reference voltages were calibrated to obtain images with sufficient contrast in the laboratory lighting conditions, and the fixed-pattern noise masks were saved using the previously mentioned GUI. A Simulink model was created to obtain data from the optical sensors and the motion-capture system and to send those data over the Wi-Fi connection to the desktop. This model was compiled with Simulink Coder and run on the PC/104. The desktop computer ran the data acquisition program, and the spacecraft simulator attitude stage was moved about each of the three axes by hand. Nothing was done to change the walls or laboratory environment to make it easier or harder for the optical sensor to track motion, except for making sure that nothing was moving in the field of view relative to the walls.

The angular velocities obtained from both sensors were transformed into the body frame and combined into a single measurement. This produced three angular velocities that were used to find the quaternion derivative [13]. The quaternion at the first time step was taken from the PhaseSpace data, and then it was numerically integrated with the derivatives from the optical sensors as time progressed. The test results are shown in Figs. 16 and 17, as the attitude stage was moved about each axis sequentially: first with a 360 deg rotation about the vertical axis, and then smaller cyclic motions in the other two body axes, returning to approximately neutral after each. The data calculated from the optical sensors show a large tendency to drift, which was caused by the noise in the images showing up as phantom motion. Since no filtering was done in the system yet, this drift was anticipated, so the data indicate that the system functioned as expected. While the difference in Euler angles (Fig. 16) shows a large error, a visual comparison of the quaternions (Fig. 17) shows that the trends match and the output should become significantly better after implementation of drift control. The third test scenario will present a method of addressing this drift error by combining the angular velocities from the flow with the point-tracking mode.

### B. Point-Tracking to $q$ Computation

The second test was for the point-tracking sensor mode to verify that the algorithm worked as intended. The sensors were mounted and calibrated the same way as in the previous test, with no physical changes to the attitude. The environment was modified by placing two bright point

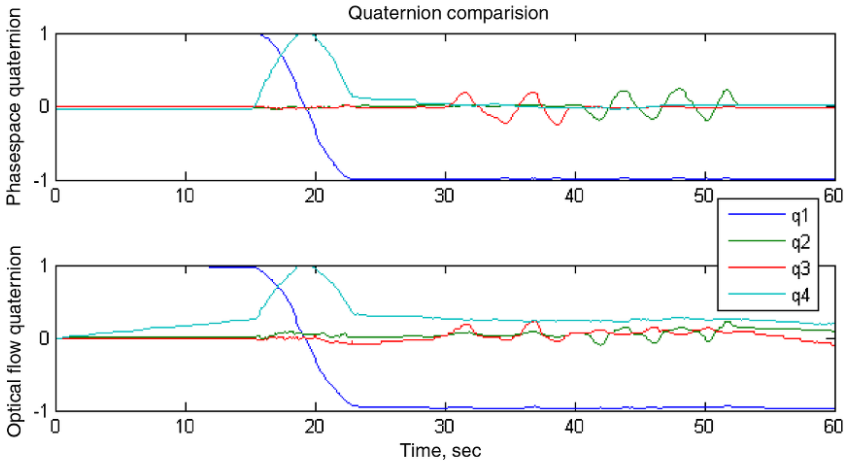



**Fig. 16 Quaternions obtained from Motion Capture (top) and Optical Flow (bottom).**

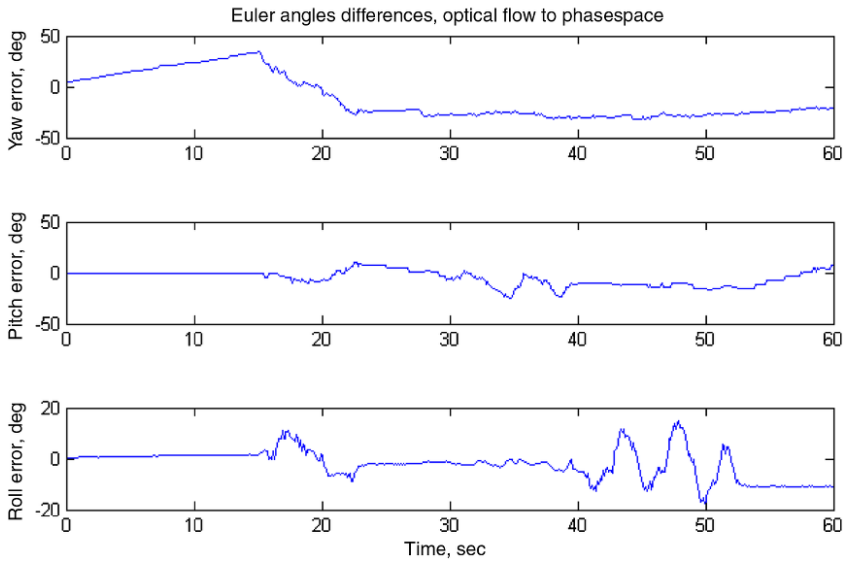



**Fig. 17 Euler Angles Comparison for quaternions in Fig. 16.**



lights on the walls that face the sensors in the attitude stage rest position, and turning off the ceiling lights to simulate a space setting with a known star field. All other aspects of the test were equivalent, with the quaternion being calculated from the known light locations in the reference frame and the calculated body frame vectors with the QUEST algorithm [14,15], with a modification from Markley and Mortari [16] to handle singularities near 180 deg rotations.

In this test, the motion was again controlled by hand, but it was moved around in arbitrary directions to cover the full field of view of each sensor. The results for this test had a much smaller error than the previous experiment, as seen in Fig. 18. The quaternion components were plotted with point-tracking overlaid on the PhaseSpace data to show that not only were the motion trends similar, the values were also very close. Plots of the error between the Euler angle representations of the attitudes are presented in Fig. 19, showing that the error is considerably lower than in the optical flow mode.

Care was taken in this test to ensure that the rotation was not so large that the lights went outside the field of view of one or more sensors; else those data points would be meaningless. Such out-of-frame points can be detected by checking the brightness of each identified point against the average for the whole data set and rejecting data when the brightness is too low.

### C. Point-Tracking with Higher-Frequency Flow Updates

The optical flow method is able to run much faster than the point-tracking: it works regardless of tracking points being in the field of view, and it captures the direction of the motion even when the magnitude is off. While the point-tracking mode runs more slowly, it is not subject to drift, only random error from image noise. A third test was then performed to combine the advantages of both sensor modes while mitigating the disadvantages. By calculating the point-tracking quaternions at a low frequency and integrating the quaternions using the angular velocities from the flow data at higher frequency, attitude data can be constructed at the higher frequency. To demonstrate the viability of this method, the data from the point-tracking run (Fig. 18) were combined with optical flow data that were also captured during that run.

The data for both point-tracking and optical flows were obtained at 10 Hz in that test (limited by the bandwidth of the SMBus interface), so to demonstrate the combined method, the point-tracking data was only used every 1 Hz, and then the angular velocities from optical flow were used to update the quaternion at 10 Hz. This result is shown in Fig. 20, with the Euler angle error for the quaternions compared to the PhaseSpace motion-capture data in Fig. 21. It is visually evident from the plots that the error does grow during the flow-only times, but it is pulled back down each second as the quaternion resets to the value from the point-tracking data. It is also visible that the computed quaternion is not constant during the flow update time, so the accuracy is improved compared to only using the low-frequency point-tracking data. While it was not necessary for these results, the method also allows for the quaternions to be updated with flow data in the event that the points being tracked go outside the field of view of the sensors, integrating the state until the points can be reacquired.

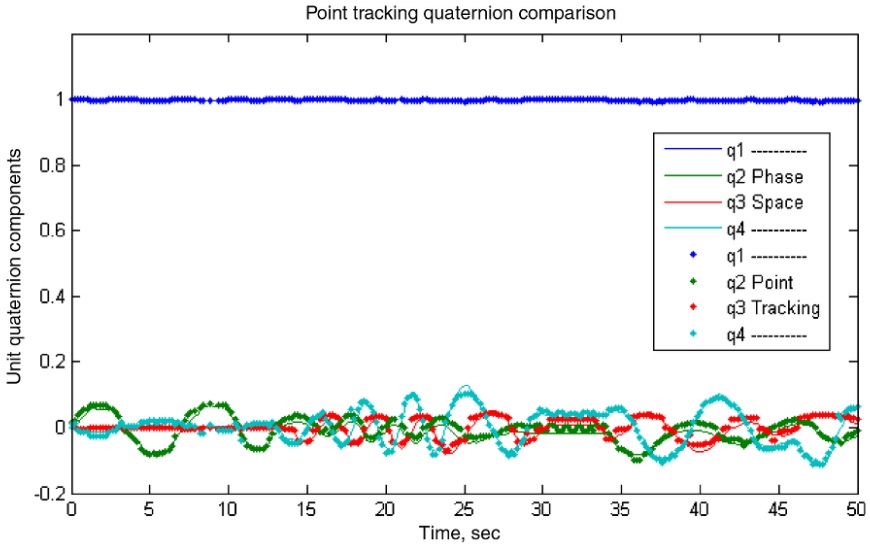



**Fig. 18 Quaternions obtained from point-tracking (points) and motion capture (lines).**

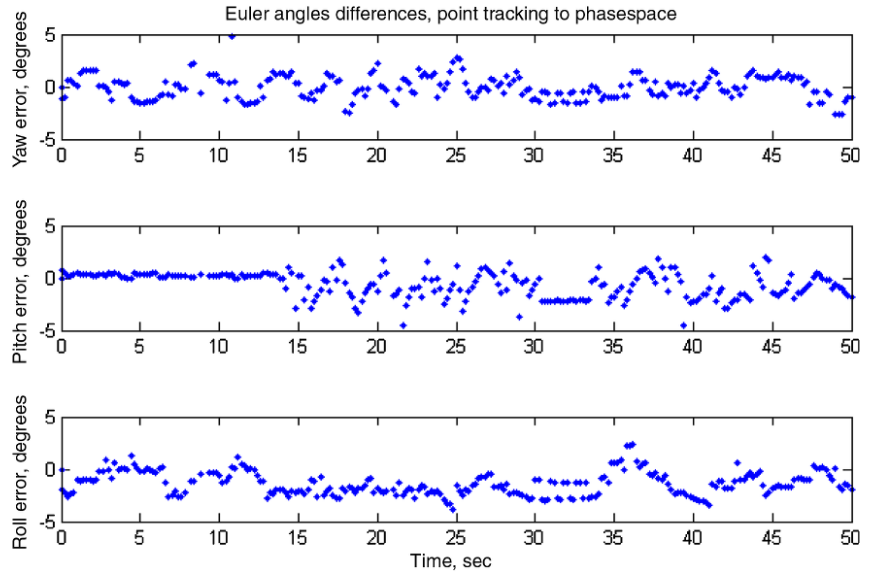



**Fig. 19 Euler angles comparison for quaternion in Fig. 18.**

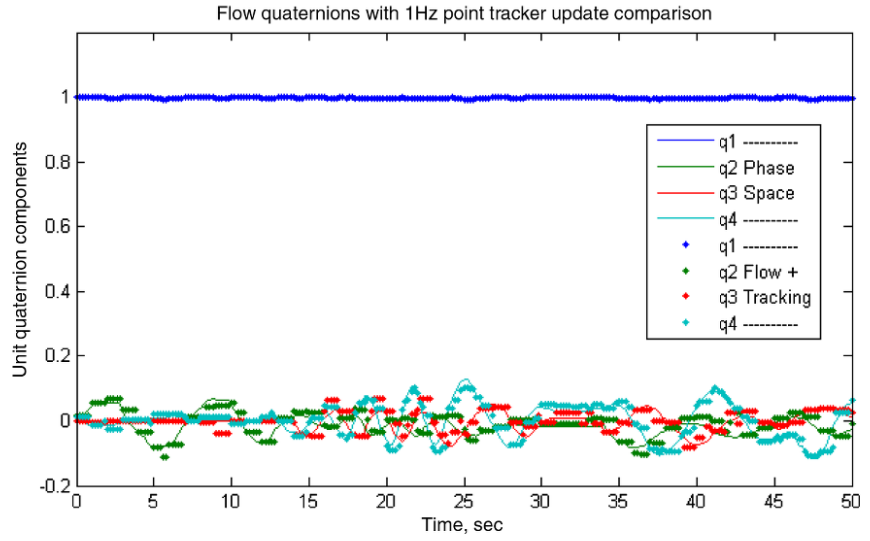



**Fig. 20 Combined flow and point-tracking quaternions (solid) with reference motion (dotted).**

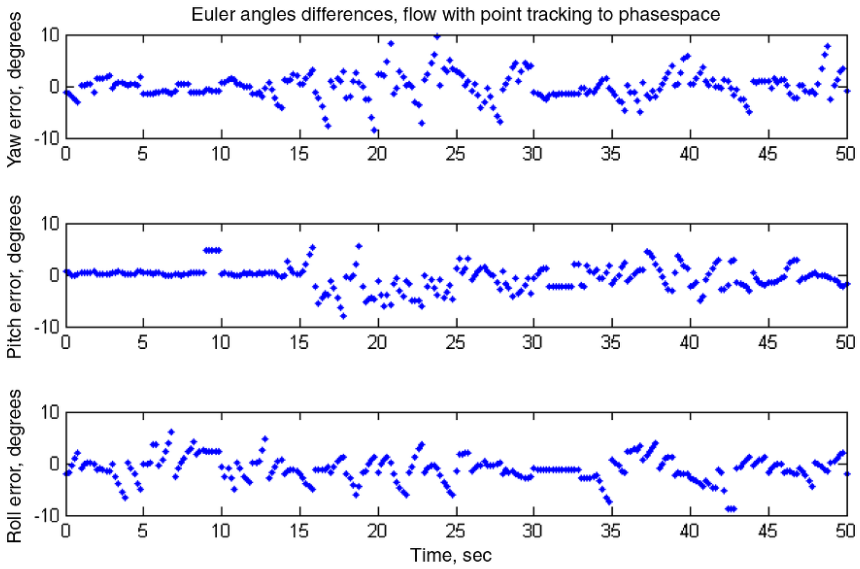



**Fig. 21 Combined flow and Point-tracking Euler angles errors for quaternions in Fig. 20.**

### D. Results Overview

The results from the optical flow comparison (Figs. 16 and 17) show that the optical flow method successfully captures the general direction of motion, but without any filtering or drift compensation, the direct numerical comparison shows a very large error. Table 3 contains a comparison of the Euler angle errors for the three test cases. All three angles have similar error magnitudes within each test, so pitch was chosen to be representative of the results.

While errors are indeed high for these results, they are presented as a proof of concept for sensors of this type, and a significant improvement to as low as 1–2 deg errors is expected if images larger than $8 \times 8$ are used and filtering is implemented, e.g., extended Kalman filtering. For a rough task such as solar panel pointing to increase power-generation efficiency, [4] suggests an attitude determination error of less than 15 deg, which is already met by both existing sensor systems and this new approach, and better accuracy would allow for missions such as diagnostic imaging of other satellites. With the addition of an EKF and expected reduction of error to the 1–2 deg range, this system will be at or better than the level of the most accurate sensors currently available for CubeSats.

### E. Comparison with Existing Star Trackers

Table 4 contains a comparison of these results to a range of existing star trackers that have flown. The Miniature Star Tracker (MST) is the smallest star tracker commercially available that could be found, and the CT-633 and SED26 are included as a reference for high-fidelity trackers available on large satellites. While the CYE8 sensors presented here compare extremely favorably in mass, size, power usage, tracking rate, and update rate, are they are equivalent in field of view, they are an order of magnitude less accurate than the only other tracker that could possibly fit on a CubeSat. As mentioned before, the accuracy is presented here directly out of the sensors without any filtering, which will be implemented in

**Table 3 Numerical comparison of attitude determination modes**

| | Data frequency, Hz | Mean error, deg | Maximum error, deg |
|---|---|---|---|
| Optical flow (pitch) | 10 | −5.177 | −24.59 |
| Point-tracking (pitch) | 10 | −0.267 | 4.53 |
| Flow with point update | 10(flow), 1(point) | −0.498 | −8.02 |

**Table 4 Comparison to existing star trackers[a]**

| | CYE8v2 pinhole | MST[b] | CT-633[c] | SED26[d] |
|---|---|---|---|---|
| Mass, g | 0.7 | 375 w/o baffle 575-775 w/ baffles | 2500 w/o baffle | 3100 w/30 deg baffle 3300 w/25 deg baffle |
| Size (w/ baffle), mm | $5 \times 18 \times 13$ | $\sim 95 \times 50 \times 50$ | $135 \times 135 \times 142$ | $160 \times 170 \times 290$ w/30 deg<br>$160 \times 170 \times 350$ w/25 deg |
| Power, W | 0.15 | 1 | 8 to 9 | 7 to 9 |
| Resolution, pixels | $8 \times 8$ | $1024 \times 1280$ | Unlisted | Unlisted |
| Field of view, deg | $22 \times 22$ | $24 \times 30$ | $17.5 \times 17.5$ | 25–30 |
| Accuracy, arcsec | $\sim \pm 700$ (unfiltered) | $\pm 70, 150(x + y, z; 1\sigma)$ | $\pm 4, 38(x + y, z; 1\sigma)$ | $\pm 3, 15(x + y, z; 3\sigma)$ |
| Update rate, Hz | 100 (1k internal) | 2 | 5 | 10 |
| Tracking rate, deg /s | >3000 (theoretical) | 10 (goal) | 0.1 0.8 w/reduced accuracy | 20 |
| Environmental tolerance | Untested | 30 + krad tolerance | 12–15-year lifetime, hardened | 18-year lifetime, hardened |

[a]w/ denotes with, and w/o denotes without.
[b]Data available online at http://www.spacemicro.com/Comtech_Areoastro/CAA_div.htm [retrieved 2013].
[c]Data available online at http://www.ballaerospace.com/file/media/D0399_CT-633.pdf [retrieved 2013].
[d]Data available online at http://www.sodern.com/sites/en/ref/SED26_154.html [retrieved 2013].

software and already exists in the other sensors. With CubeSat restrictions at $10 \times 10 \times 10$ cm and 1.3 kg per 1U, providing only a few watts of power total, the high-accuracy trackers are unsuitable and the MST is only viable if it is the entirety of the payload for the mission, which is undesirable.

## VI. Conclusions

This paper presents a new attitude determination system (ADS), based on nano-optical devices and real-time application interface Linux software libraries. The goal of this work is to provide nanospacecraft with attitude determination capability, characterized by quick and affordable integration. In particular, C interfaces for the optical devices and the PhaseSpace motion-capture system are developed and readily integrated into a Simulink environment. Firmware modifications for the optical devices are also presented, and new algorithms are tested. An overview of each hardware component is included along with a description of how it is applicable to a ground-based spacecraft simulator.

Validation of the proposed solution is performed via hardware-in-the-loop experimentation using a newly built six-degree-of-freedom spacecraft simulator at Rensselaer Polytechnic Institute, employing only three-axis attitude motion. The comparison between the new system's data and the PhaseSpace's data shows satisfactory agreement. In particular, two modes of operation are tested for the optical devices. The first one computes angular velocity for the platform hosting the ADS, through optical flow measurements. The second extracts quaternions from point-tracking of known light sources in the laboratory environment. A combination of the two modes is used to highlight the advantages of each while mitigating the relative disadvantages, using optical flow measurements to update quaternions produced by the lower-frequency point-tracking mode. The proposed ADS method holds the potential for replacing star trackers and gyroscopes on very small space platforms.



## Appendix A: Installation of Real-Time Application Interface Linux on a PC/104

The following assumes that a basic installation of Ubuntu 10.04 Long Term Support has been completed on the PC/104, including setting up an open-SSH server for remote logins, for which there are guides and installation media available at http://www.ubuntu.com/ (retrieved 2013). Other versions of Linux will most likely work but may require changes in the following steps. General instructions for installing RTAI in modern versions of Ubuntu are also available from the HART Toolbox website (http://hart.sourceforge.net/ [retrieved 2013]). If this appendix is to be followed in the future and the referenced versions of programs are out of date, that project will likely have up to date instructions.

Lines with the >> prefix are to be entered at the command line. The commands are intended to be run as a nonroot user that has root permissions (through the sudo command); if running as root, that part of the commands can be omitted. The outermost parenthesis surrounding any filename or option listed in this Appendix is for identification in the text, and it is not a part of said filename or option and should be omitted when entering on the computer. Any text in an instruction shown as <text> is a placeholder for something that is specific to the user's computer and should be replaced with the appropriate text as indicated, without keeping the <> brackets. The instructions in this and the following appendices are to be performed on the PC/104 system.

1) Install required packages used in installation:

```
>> sudo apt-get install cvs subversion build-essential libtool automake libncurses5-dev
```

2) Download an RTAI-patched version of the Linux kernel from the LinuxCNC project:

```
>> echo deb http://www.linuxcnc.org/lucid lucid base emc2.4 > /tmp/linuxcnc.list
>> echo deb-src http://www.linuxcnc.org/lucid lucid base emc2.4 >> /tmp/linuxcnc.list
>> sudo mv /tmp/linuxcnc.list /etc/apt/sources.list.d/
>> gpg –keyserver pgpkeys.mit.edu –recv-key 8F374FEF
>> gpg -a –export 8F374FEF | sudo apt-key add -
>> sudo apt-get update
>> sudo apt-get install linux-headers-2.6.32-122-rtai linux-image-2.6.32-122-rtai
```

3) Reboot into the newly installed RTAI kernel; the option will show up in the bootloader menu.

4) Download and configure the latest RTAI software package:

```
>> cd /usr/src
>> sudo cvs -d:pserver: anonymous@cvs.gna.org:/cvs/rtai co magma
>> sudo ln -s magma rtai
>> cd /usr/src/rtai
>> sudo make menuconfig
```

5) Make sure the following options are correct in the configuration tool: a) installation location /usr/realtime, b) kernel source tree location /usr/src/linux-headers-2.6.32-122-rtai, c) verify that the number of processors matches how many cores your system has (usually 1 on a PC/104).

6) Compile and install the modules:

```
>> sudo make
>> sudo make install
>> sudo sed -i 's/\(PATH=\"\)\1\/usr\/realtime\/bin:/' /etc/environment
```

7) Log out and back in again (or just reboot the PC/104).

8) Create an RTAI module load script:

```
>> sudo nano /usr/local/bin/start_rtai
```

9) Add the following lines to that file:

```
sync
/sbin/insmod /usr/realtime/modules/rtai_hal.ko
/sbin/insmod /usr/realtime/modules/rtai_up.ko
/sbin/insmod /usr/realtime/modules/rtai_fifos.ko
/sbin/insmod /usr/realtime/modules/rtai_sem.ko
/sbin/insmod /usr/realtime/modules/rtai_mbx.ko
/sbin/insmod /usr/realtime/modules/rtai_msg.ko
/sbin/insmod /usr/realtime/modules/rtai_netrpc.ko ThisNode="127.0.0.1"
sync
```

10) Save and exit the file (CTRL+x, confirm saving and filename with "y").

11) Set that file to be executable:

```
>> sudo chmod a+x /usr/local/bin/start_rtai
```

12) Verify that the modules load correctly by running that file as root:
>> /usr/local/bin/start_rtai
13) Set that loader script to be called when the system boots by adding the following line to /etc/rc.local before the "exit 0" line:
/usr/local/bin/start_rtai

## Appendix B: Configuration of I2C-dev

The I2C-dev module is included with the Linux kernel by default, but the header file to include in C programs when compiling them to use it must be downloaded manually.

1) Install the package that contains the file >> sudo apt-get install I2C-tools libI2C-dev.

2) Set the module to be loaded at boot time by adding "I2C-dev" to the end of the file /etc/modules the same way the start_rtai file was edited previously.

## Appendix C: Using the S-Functions in a Simulink Model

This appendix and all the following are done from the Linux desktop and not the PC/104. It also assumes that MATLAB, Simulink, and the Simulink Coder have been installed and configured with the GNU Complier collection available for use with the "mex" command. More information about this is available from MathWorks.

The C S-functions must be compiled into executables that MATLAB can natively call ("mexed") before they can be included in a Simulink model. This is a simple procedure for the CYE8 reader block; once the files are copied to the toolbox folder, navigate to the directory containing the .c files from within MATLAB and run "mex <filename>.c" for it, then put the resulting mex file into a folder in MATLAB's list of include paths so it can be used.

The PhaseSpace block requires the addition of extra shared libraries for communication with the server, which are available from PhaseSpace, Inc. To being using this block, a few things must be changed to match your system.

1) Open sfun_PhaseSpaceC.c and change the paths on the two #include lines that point to "owl.h" and "owl_math.h" to point to where they are on your system.

2) Define the coordinates of the LEDs on your specific test spacecraft in the variable "float RIGID_BODY[][]" just inside mainFunction().

3) mex the .c file along with the shared library from within MATLAB with "mex sfun_PhaseSpaceC.c libowlsock.so".

When using the blocks in a model that is to be run on the PC/104, some more options must be configured.

1) With the model open, open the Configuration Parameters dialog (in the "Simulation" menu).

2) Select the "Real-Time Workshop" tab in the left panel.

3) In the "System target file" box, click the "Browse" button and select "rtai.tlc", and make sure the language in the option just below it is set to C.

4) At the bottom of the same panel, check the option for "Generate Code Only".

5) The rtai.tmf file needs to be modified to handle using a shared library for the PhaseSpace block by commenting out the "-static" option in the LDFLAGS variable (line 71) by changing it to "#-static".

6) In the same file, in the bottom of the "Additional Libraries" section, "/usr/local/lib/libowlsock.so" should be added to the end of last line starting with "LIBS +=".

After this, it is possible to generate the program to be run on the PC/104 from the Simulink model.

1) Now back in the Configuration Parameters dialog, press the "Generate code" button to set up the model for compiling.

2) Look at the output in the main MATLAB window for any errors, and when it completes successfully, the last part of the output will show which directory the code has been generated in.

3) Open a terminal window and navigate to this directory, which will be in the same location as the model itself in the folder "<model_name>_rtai": >> make -f <model_name>.mk.

4) If the compilation fails with "file not found" errors for the S-functions or required libraries, copy the sfun_<block_name>.c into that directory and recompile until it works.

## Appendix D Connecting to the PC/104 and Running the Models in Real Time

1) Access a terminal window on the desktop computer and navigate to the directory containing the Simulink model (which also contains the executable after it has been compiled, with the same name as the model but no extension). Connect to the PC/104 and upload the file to the home directory on it:
>> sftp <username>@<PC/104 IP address>
>> put <executable_name>
>> exit

2) Remotely log on to the PC/104 to run the model:
>> ssh <username>@<PC/104 IP address>
>> sudo ./<executable_name> -v –f 5

3) The preceding command runs the model in verbose mode to show all the output at the command line (-v), with a total run time of 5 s (-f 5). Omitting "-v" will make it run with no output, changing the value after "-f" changes the time for the simulation to run, and omitting "-f #" makes the simulation run indefinitely. More information about the available options can be found from MathWorks in the Simulink Coder help.

## Acknowledgment

This research was supported by the NASA/New York State Space Grant Consortium, grant J40259.

## References

[1] Thakker, P., and Shiroma, W., "Emergence of Pico- and Nanosatellites for Atmospheric Research and Technology Testing," *Progress in Astronautics and Aeronautics*, Vol. 234, AIAA, Reston, VA, 15 Sept. 2010.

[2] Chin, A., Coelho, R., Brooks, L., Nugent, R., and Puig-Suari, J., "Standardization Promotes Flexibility: A Review of CubeSats' Success," *AIAA/6th Responsive Space Conference*, AIAA Paper RS6-2008-4006, 2008.

[3] Konstantinidis, K., "CubeSats: a Review," Online Semester Project, Space Research Laboratory, Democritus Univ. of Thrace, Xanthi, Greece, 2010, http://www.thesciencecollective.com/ctide/wp-content/uploads/2011/11/Cubesats_a_review_KKonstant.pdf [retrieved 10 Oct. 2012].
[4] Bowen, J. A., "On-board Orbit Determination and 3-Axis Attitude Determination For Picosatellite Applications," M.S. Thesis, Dept. of Aerospace Engineering, California Polytechnic State Univ., San Luis Obispo, CA, July 2009.
[5] Mantegazza, P., Dozio, E. L., and Papacharalambous, S., "RTAI: Real Time Application Interface," *Linux Journal*, Vol. 2000, No. 72, April 2000.
[6] Bevilacqua, R., Hall, J., Horning, J., and Romano, M., "Ad Hoc Wireless Networking and Shared Computation for Autonomous Multirobot Systems," *Journal of Aerospace Computing, Information, and Communication*, Vol. 6, No. 5, 2009, pp. 328–353. doi:10.2514/1.40734
[7] Gallardo, D., Bevilacqua, R., and Rasmussen, R. E., "Advances on a 6 Degrees of Freedom Testbed for Autonomous Satellites Operations," *AIAA Guidance, Dynamics and Control Conference*, AIAA Paper 2011-6591, 2011.
[8] Kenyon, S., Bridges, C. P., Liddle, D., Dyer, R., Parsons, J., Feltham, D., Taylor, R., Mellor, D., Schofield, A., and Linehan, R., "STRaND-1: Use of a $500 Smartphone as the Central Avionics of a Nanosatellite," *62nd International Astronautical Congress 2011*, Cape Town, South Africa, Paper IAC-11-B4.6B.8, Oct. 2011.
[9] Srinivasan, M. V., "An Image-Interpolation Technique for the Computation of Optic Flow and Egomotion," *Biological Cybernetics*, Vol. 71, No. 5, 1994, pp. 401–415. doi:10.1007/BF00198917
[10] Barrows, G. L., "Mixed-Mode VLSI Optic Flow Sensors for Micro Air Vehicles," Ph.D. Dissertation, Dept. of Electrical Engineering, Univ. of Maryland at College Park, College Park, MD, Dec. 1999.
[11] Kitchen, L., and Rosenfeld, A., "Gray Level Corner Detection," *Pattern Recognition Letters*, Vol. 1, No. 2, 1982, pp. 95–102. doi:10.1016/0167-8655(82)90020-4
[12] Davis, J., Doebbler, J., and Vavrina, M., "Characterization and Calibrating the Novel PhaseSpace Camera System," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2011-6582, Aug. 2011.
[13] Wie, B., *Space Vehicle Dynamics and Control*, AIAA Education Series, AIAA, Reston, VA, 1998, pp. 326–327.
[14] Shuster, M. D., "Approximate Algorithms for Fast Optimal Attitude Computation," *AIAA Guidance and Control Conference*, AIAA Paper 1978-1249, Aug. 1978.
[15] Shuster, M. D., and Natanson, G. A., "Quaternion Computation from a Geometric Point of View," *Journal of Guidance and Control*, Vol. 4, No. 1, 1981, pp. 70–77. doi:10.2514/3.19717
[16] Markley, F. L., and Mortari, M., "Quaternion Attitude Estimation Using Vector Measurements," *Journal of the Astronautical Sciences*, Vol. 48, Nos. 2–3, April–Sept. 2000, pp. 359–380.

E. Atkins
*Associate Editor*