PNNL-23994

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

# A New Approach to Space Situational Awareness using Small Ground-Based Telescopes

## Final Report

**December 2014**

NC Anheier
C Chen

U.S. DEPARTMENT OF ENERGY

## DISCLAIMER



PACIFIC NORTHWEST NATIONAL LABORATORY
*operated by*
BATTELLE
*for the*
UNITED STATES DEPARTMENT OF ENERGY
*under Contract DE-AC05-76RL01830*

**Printed in the United States of America**

**Available to DOE and DOE contractors from the**
**Office of Scientific and Technical Information,**
**P.O. Box 62, Oak Ridge, TN 37831-0062;**
**ph: (865) 576-8401**
**fax: (865) 576-5728**
**email: reports@adonis.osti.gov**

**Available to the public from the National Technical Information Service,**
**U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161**
**ph: (800) 553-6847**
**fax: (703) 605-6900**
**email: orders@ntis.fedworld.gov**
**online ordering: http://www.ntis.gov/ordering.htm**


This document was printed on recycled paper.
(9/2003)

PNNL-23994

# A New Approach to Space Situational Awareness using Small Ground-Based Telescopes

NC Anheier
C Chen

December 2014

Prepared for
the U.S. Department of State
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

# Executive Summary

In 1957, the United States became pressed to develop a space situational awareness (SSA) program after the launch of the Russian satellite, Sputnik. Today there are over 900 operational satellites and over 14,000 cataloged man-made debris objects greater than 5 centimeters (cm) in orbit. A U.S. National Research Council (1995) report portrays a grim future regarding the expanding space debris and collision risk to spacecraft and future space missions. A vast number of debris and natural (meteoroid) objects remain undetected between low earth orbit (LEO) and geosynchronous orbit (GEO). The SSA tracking burden increased by 60% after just two events: the 2007 Chinese anti-satellite test (FengYun 1C weather satellite) and the Cosmos 2251 and Iridium 33 satellite collision in 2009 created 100,000s of debris particles larger than 1 cm. The new fleet of micro-satellites (~10–30 cm) planned for LEO deployment further increases the SSA burden.

The United States' objectives to promote long-term sustainability of outer space activities are outlined in the National Space Policy and formalized through ratified treaties and other international agreements. National Space Policy describes the nation's commitment to promote safe and responsible operations in space, improve information collection and sharing for space object collision avoidance, protect critical space systems and supporting infrastructures, and strengthen measures to mitigate orbital debris. The United States has also ratified four space law treaties, including the Outer Space Treaty that bans nuclear weapons or any other weapons of mass destruction in outer space. The United States is also working with the European Union to develop the international "Code of Conduct for Outer Space Activities" agreement, which seeks to enhance the safety, security, and sustainability of outer space activities. These objectives will require new SSA technologies and techniques that protect satellite assets, detect and monitor orbiting debris fields, and provide treaty verification.

The fundamental requirement of SSA is to acquire full knowledge of all resident space objects (RSOs) in earth orbit. This is currently accomplished by two distinct activities, including new object detection and tracking known objects to periodically characterize their physical properties. RSO acquisition, tracking, and imagery data collection can be extremely challenging. Initial RSO acquisition is dependent on the orbital data and telescope pointing accuracy and the effectiveness of error correction (e.g., pointing, atmospheric correction). Once acquired, sustained tracking of RSO targets can also be difficult. Telescope tracking systems are typically designed to provide optimum tracking performance at sidereal rates (~15 arcsec/sec). However, LEO objects have extremely large angular velocities that can exceed sidereal rates by a factor of 10. Many telescopes can slew at much higher angular velocities, but they do not have the tracking accuracy or large field of view (FOV) needed to keep a fast moving LEO target within the FOV of the imaging camera. Many RSOs also have low radar cross sections that result in low apparent brightness. These factors make small LEO and distant GEO targets very difficult to acquire and detect against the background photon flux.

The United States Space Surveillance Network and NASA operate a worldwide SAA network of radar telescope systems and optical telescopes. Many of these efforts focus on new object detection using large, wide-field optical telescope systems that track large areas of the sky and automatically locate new RSOs by detecting the characteristic light streak against the fixed star-field. Other systems are designed to precisely track known RSOs and acquire photometry data. The Starfire Optical Range at Kirkland Air Force Base is equipped with state-of-the-art SSA optical telescopes, including a 3.5-m telescope featuring deformable optics and a laser guide star to help minimize atmospheric turbulence effects that distort the

image and shift the apparent brightness and position. The Starfire telescope, with atmospheric correction, can distinguish basketball-sized objects at an orbital distance of 1600 km. Large observatories, equipped with guide stars and deformable optics systems, are effective at detecting and resolving small objects; however, they are resource-intensive and difficult to extend broadly at other global observation sites.

This report discusses a new SSA approach evaluated by Pacific Northwest National Laboratory (PNNL) that may lead to highly scalable, small telescope observing stations designed to help manage the growing space surveillance burden. Using the methods and observing tools described in this report, **the team was able to acquire and track very faint satellites (near Pluto's apparent brightness). Photometric data was collected and used to correlate object orbital position as a function of atomic clock-derived time. Object apparent brightness was estimated by image analysis and nearby star calibration. The measurement performance was only limited by weather conditions, object brightness, and the sky glow at the observation site. In the future, these new SSA technologies and techniques may be utilized to protect satellite assets, detect and monitor orbiting debris fields, and support Outer Space Treaty monitoring and transparency.**

PNNL's approach capitalized on high-fidelity, commercial-off-the-shelf (COTS) products that were developed for advanced astronomical and science studies, including state-of-the-art electron multiplying charged coupled device (EMCCD) imaging cameras, small (<1 m) optical telescopes, servomotor telescope drive systems, and high-performance telescope control software. These tools were used along with extensive Internet resources (e.g., astronomical databases, orbital propagation tools, observational planning tools), and custom-developed software. The feasibility of this approach was demonstrated through a unique collaborative effort between PNNL and several regional academic institutions and observatory facilities.

Future work has been identified to further demonstrate and extend the techniques developed by this research team, including a need for the following tasks:

- Conduct further and more extensive satellite and space debris observational studies at collaborating regional observatories
- Demonstrate and develop further methods to acquire SSA data from very small targets moving at high angular velocities and faint near-earth objects
- Complete and demonstrate high-speed, arcsecond-precision telescope tracking system on the Hoch telescope
- Complete orbital propagation software code needed to generate target orbital data and collaborate with commercial vendors to improve the accuracy of the satellite tracking software
- Develop architecture design concepts for a fully automated SSA observation station
- Evaluate other imaging camera and other commercial technologies for SSA applications
- Explore options for technical exchanges between the U.S. and Russian SSA communities to discuss results of this project and similar efforts by Russia.

# Acknowledgments

The U.S. Department of State, Office of Verification and Transparency Technologies Key Verification Assets Fund (V Fund) sponsored the work described in this report. The authors gratefully acknowledge the collaboration and support from the following consortium members who participated in this R&D project: Dr. Andrea Dobson, Martin Scott, Emma Dahl, and Larry North (Whitman College), Michael Brady (Moore Observatory Director), Karl Olson, Michael Durst, Tony George (Retired), Levi Yencopal, Curtis Crawford, John Cole (Moore Observatory/CBC), and Roy Gephart, Richard Hoch, and Rob Parchen (AASTA). The authors also thank Steven Bisque (Software Bisque) and Dan Gray (Sidereal Technology) for their invaluable technical assistance on their observatory hardware and software products. Pacific Northwest National Laboratory is a multi-program national laboratory operated by Battelle for the U.S. Department of Energy. Last but certainly not least, the authors thank the Department of State/Bureau of Arms Control, Verification and Compliance V Fund for its support of our research. The views expressed herein do not represent the official positions or policies of the Department of State or any other entity of the United States Government.

# Acronyms and Abbreviations

| | |
|---|---|
| AASTA | Alliance for the Advancement of Science Through Astronomy |
| ALE | Arid Lands Ecology Reserve |
| ASCOM | Astronomy Common Object Model |
| CBC | Columbia Basin College |
| CCD | charged-coupled device |
| COTS | commercial-of-the-shelf |
| DEC | declination |
| EMCCD | electron multiplying charged coupled device |
| FOV | field of view |
| GEO | geosynchronous orbit |
| LEO | low earth orbit |
| NELM | naked eye limiting magnitude |
| PNNL | Pacific Northwest National Laboratory |
| PNRO | Pacific Northwest Regional Observatory |
| RA | right ascension |
| RSO | resident space object |
| SNR | signal-to-noise ratio |
| SSA | space situational awareness |
| TLE | two-line element |
| UTC | coordinated universal time |

# Contents

# Figures

# 1.0 Introduction

The goal of this space situational awareness (SSA) study was to evaluate the feasibility of using small (<1-m) optical telescopes to collect useful resident space object (RSO) data in support of future Outer Space Treaty and other SSA missions. Key to this goal was developing methods to collect time-resolved photometry measurements of RSO targets, without the overhead associated with large astronomical observatories. The electron multiplying charged coupled device (EMCCD) camera used in this study is an enabling technology, designed to operate at fast frame rates, while effectively eliminating a dominant noise source. The study was organized around evaluation of commercial hardware and software developed for the advanced amateur astronomy community, which has made significant contributions to professional astronomical research.

This SSA project also provided a unique opportunity for Pacific Northwest National Laboratory (PNNL) to collaborate with a regional consortium of public and private academic institutions and observatories, including the Alliance for the Advancement of Science Through Astronomy (AASTA), the Pacific Northwest Regional Observatory (PNRO), Whitman College, Columbia Basin College (CBC), and the Moore Observatory. This consortium utilizes their resources and expertise in support of community outreach, academic STEM programs, scientific research, and to foster the future generation of scientists, engineers, and policy makers now enrolled in school.

AASTA is a non-profit scientific and educational organization based in southeastern Washington State. AASTA operates PNRO, which includes a recently constructed observatory with a 7.3-meter Ash dome and the newly refurbished Hoch telescope. Whitman College, Department of Astronomy and CBC facilitated access to the Pacific Northwest Region and Moore Observatories, respectively. Whitman College is a private four-year, liberal arts and sciences college with enrollment of about 1600 students. Whitman College owns the Braden Ranch property where the PNRO is located. A memorandum of understanding exists between Whitman College and AASTA for the perpetual use of this property. Whitman's Department of Astronomy manages the PNRO and oversees ongoing operational expenses (e.g., power, internet, minor maintenance) using external funding. CBC is a public community college located in Pasco, Washington, that offers a wide range of workforce-related, 2-year associates degrees. The CBC campus is also home to the Moore Observatory. An astronomy program is offered that is tightly integrated with the Moore Observatory to provide science students hands-on experience in applying math and science skills in their learning process. The observatory is also used for elementary, middle, and high school programs, as well as for student research and community outreach. The observatory was developed through the generous donation of the late Robert Moore and his late wife, Elisabeth.

# 2.0 Approach

## 2.1 Pacific Northwest Regional Observatory and Moore Observatory

PNNL developed and evaluated SSA techniques at the Pacific Northwest Regional and the Moore observatories. The PNRO Hoch telescope is a custom research-grade 0.8-meter, f/11.9, classical Cassegrain reflecting telescope, mounted on a massive altitude-azimuth fork arm mount, as shown in

Figure 1. PNRO is located under dark skies within the expansive Braden Ranch, near 46° north latitude, 119° west longitude, at 490 m altitude. In early FY2014, the PNRO observatory and telescope were in the final commissioning stages, following its removal in 2009 from its original location on Rattlesnake Mountain on the U.S. Department of Energy, Fitzner/Eberhardt Arid Lands Ecology Reserve (ALE) in South Central Washington State.

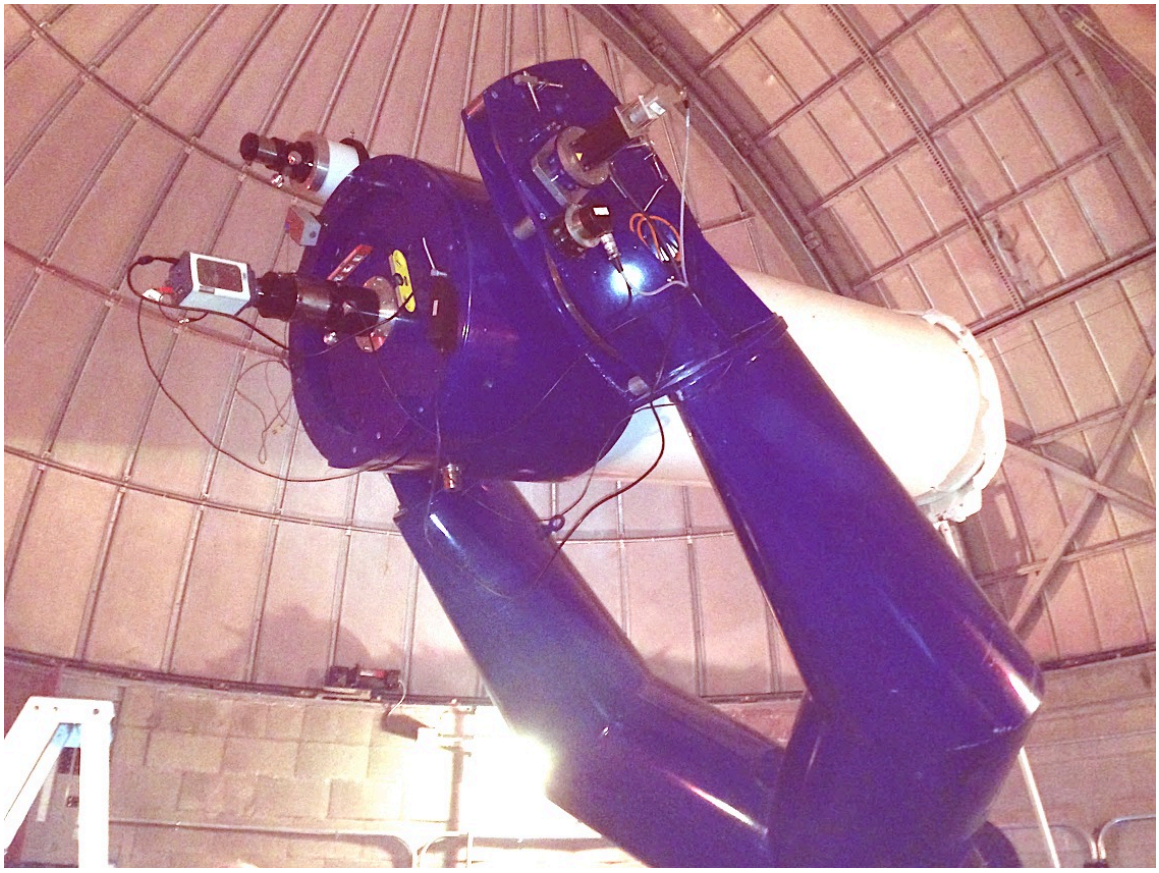


**Figure 1**. The 0.8-m Hoch Classical Cassegrain Telescope, shown Equipped with the EMCCD Camera Used for the SSA Study

The Moore Observatory is equipped with a Meade LX200 0.4-meter (16-in.) f/10 commercial-grade Schmidt-Cassegrain telescope, mounted on a German-equatorial mount (Paramount ME), as shown in Figure 2. The observatory is located on the campus of CBC at 46° 15' 8.4" north latitude and 119° 07' 36.5" west longitude, at an altitude of 124 m.

**Figure 2**. The 16-in. Moore Schmidt-Cassegrain Telescope, shown Equipped with the EMCCD Camera Used for the SSA Study

## 2.2 SSA Control Hardware and Software Architecture

The PNNL SSA study developed telescope and camera hardware and software architectures to provide accurate telescope pointing and tracking and effective RSO image data collection, as shown in Figure 3. To achieve these objectives, PNNL developed and evaluated designs based on commercial-off-the-shelf (COTS) hardware and software technologies, which were then combined with custom software solutions. The specific telescope control systems differ between PNRO and the Moore Observatory, but the fundamental control architectures share similar design features. The same camera control architecture was used at both observatories.

Because the PNRO was in the initial stages of commissioning, significant hardware and software upgrades and modifications were required to provide the target acquisition and tracking performance needed for the SSA study. Telescope control was provided by the RSO tracking and data collection architecture shown in Figure 3 (left). A Sidereal Technologies Servo Controller II hardware module and SiTechExe software was installed to control the telescope motion (http://siderealtechnology.com). The Hoch telescope drive system uses a dual-loop position-velocity servo feedback design. The servomotors are equipped with 10,000 pulses/rev incremental optical encoders that generate about 11 pulses for each arcsecond of telescope movement. The telescope is also equipped with incremental encoders on the final drive wheel that produce about $2^{27}$ pulses per full axis rotation. The servomotor encoders provide

feedback in the velocity PID servo loop, while the telescope encoders provide fine pointing error correction. A joystick is provided with the controller that allows manual control of the right ascension (RA) and declination (DEC) axes. The SiTechExe software provides a user interface that shows the current telescope pointing position overlaid onto the sky view. The application includes traditional features, such as *GO TO*, celestial object databases, Internet database queries, and a RSO database, with built-in satellite tracking support. A calibration routine, called *TPoint*, is included that quantifies and automatically compensates for systematic telescope pointing errors and telescope mount polar misalignment. This software modeling and compensation tool is a critical component needed to provide high-fidelity target acquisition and tracking performance. SiTechExe is an Astronomy Common Object Model (ASCOM)-compliant application that enables plug-and-play support between astronomy software and hardware (e.g., dome controllers, focusers, imaging cameras). The servo controller can also interface direct with SkyX (Software Bisque), which is a full-featured software tool for observatory and telescope control.

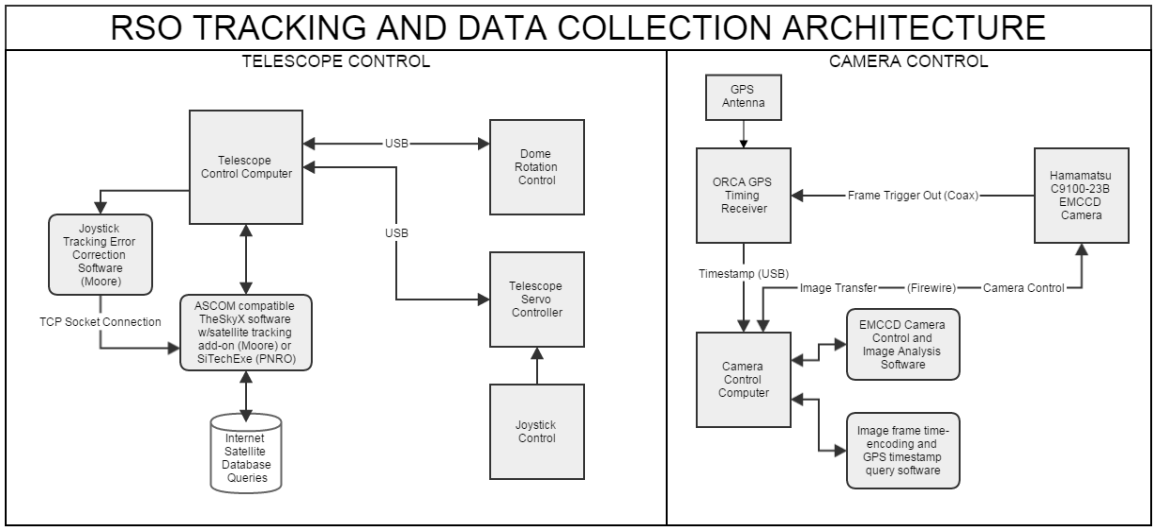



**Figure 3**. Telescope Control and Image Acquisition Architectures Developed for RSO Tracking

The Hoch telescope was equipped with Kollmorgen brushless DC servomotors (AKM43L) to drive the RA and DEC axes, as shown in Figure 4. These servomotors are capable of slewing the greater than 3,000 kg telescope at several degrees per second. The fork arm mount design features a 1440-to-1 gearless reduction drive, based on a roller/disk arrangement, which overcomes backlash and periodic error inaccuracies commonly associated with mesh and worm gear reducers (Knight 1979).

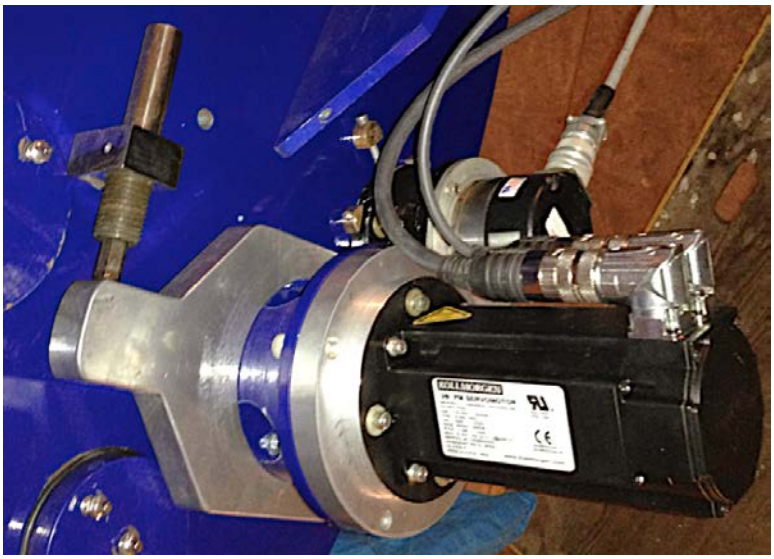


**Figure 4**. Kollmorgen Brushless DC Servomotors (AKM43L) Installed on the Hoch Telescope

The Moore Observatory architecture is very similar to PNRO's, except all the hardware and software components were procured as COTS products. The Paramount ME German equatorial mount is a proprietary design, but likely based on a closed-loop servomotor system. Large worm gears that provide less than 7 arcseconds peak-to-peak periodic error are used to provide the necessary gear-reduction on both axes. Software periodic error correction algorithms reduce this error even further. SkyX Professional, version 6, is used for observatory and telescope control. SkyX also provides an optional *TPoint* software add-on to model and compensate for telescope pointing errors. An ASCOM interface provides automatic dome rotation during telescope tracking to keep the dome opening aligned with the telescope aperture. SkyX has a basic satellite-tracking feature (only available when interfaced to the Paramount ME drive) that can automatically slew to satellites or other RSOs, and then continuously update the tracking rates for each axis so that the object remains centered on the camera.

An advanced satellite tracking add-on is under development at Software Bisque that has enhanced satellite orbital propagator code and telescope pointing and tracking accuracy. The add-on also includes a software-based joystick that can be used to center the RSO target within the telescope or camera field of view (FOV). Bisque Software provided this add-on to the Moore Observatory for evaluation purposes during the PNNL SSA study. Initially, this software-based joystick was cumbersome to use. Typically, an observer was at the wide-field finder scope during initial RSO target acquisition. If the target was not centered or within the finder FOV, the observer had to call out error correction commands to another staff member (at the desktop computer) to click on the joystick command buttons (e.g., up, down, right, left). Fortunately, SkyX can receive commands through a TCP/IP socket connection protocol. A Python program, called SkyX Bump.exe, was developed by PNNL to read keys 8, 5, 4, and 6 on a USB numeric keypad, and then send up, down, right, and left commands to the SkyX software joystick (Appendix A). The solution allowed the observer to conveniently correct pointing errors using the handheld keypad, while looking through the finder scope.

The image acquisition architecture is shown in Figure 3 (right). A Hamamatsu/Nikon EMCCD camera (ImagEM X2, C9100-23B) is used to collect RSO image data. The ImagEM X2 delivers

70 frames/sec at full frame (512 × 512 pixels) and provides exceptional quantitative ultralow light imaging at high speed. The unique charge multiplication capability allows signal amplification while still in the charge domain, before the charge is converted to voltage. This effectively eliminates a dominant noise source association with voltage amplifiers. The extremely low readout noise and user-selectable charge multiplication allows optimization of noise budget so that shot noise (theoretical limit) dominates a given measurement. The EMCCD camera can perform real-time image processing and has software for basic image post-processing. Further EMCCD performance specifications can be found in Appendix B.

Coordinated universal time (UTC) is acquired using a synchronized time code generator (ORCA, GS-101) with ±100 ns absolute time accuracy. This time code generator is equipped with a number of useful features, including serial communication and flexible input/output triggering. Further specifications can be found in Appendix C. Time synchronization was implemented by configuring the EMCCD camera frame output trigger (issued at the beginning of each image frame acquisition) to initiate a timestamp reading by the time code generator, which is then transmitted to the camera control computer for storage. The camera control computer also receives the image frame, as high-resolution TIFF data, using a high-speed IEEE 1394 serial bus. In subsequent software upgrades, the instantaneous telescope RA and DEC coordinates will be queried from SkyX (or SiTechExe) and saved with the timestamp and image frame.

A Windows-executable Python program (TSCapture.exe) was developed to configure and synchronize the image acquisition, time code generation, and data record generation software. This code is part of a larger suit of satellite tracking software, as shown in Figure 5. While the EMCCD control software (HCImage Live) captures and saves the image data to a folder on the camera control computer, TSCapture monitors the USB serial port for timestamp data which is also saved to a folder. After the image capture session is completed, TSCapture then overlays the timestamp data for each frame at the top, left-hand side of each image and also inserts the time stamp data into the image properties information. Finally, the processed images are combined into a video with the codec and frame rate specified by the user. A detailed description of this software, operation instructions, and the Python code is provided in Appendix D.

A preliminary observational planning Python program, called SatSearch.exe (Appendix E), was also developed to support PNNL SSA studies. Initial target studies required bright objects with relatively low angular velocities to gain experience with target acquisition and tracking and photometry measurements. Many Internet resources are available that feature extensive satellite databases and observing planners. The Python program collects data from the Internet satellite databases using scripted HTML commands. The program will search the satellite database maintained by the U.S. Strategic Command Joint Space Operations Center (space-track.org). Queries are currently based on object type and orbital angular velocity, but other search criteria can be easily added if required. The query data is saved to an Excel spreadsheet, while the two-line element (TLE) data for each object is saved as its own text file. The program then consults calsky.com to retrieve the visual magnitude estimates for the RSO list. The daily predictions for the 100 brightest satellites are also retrieved from heavens-above.com. The Space Track and 100 brightest satellites datasets are then cross-referenced to identify favorable RSO targets for a given night's observation. The NORAD Simplified General Propagator source code (SGP4) was acquired from Python's package index website (pypi.python.org), which provides a Python implementation of David Vallado's C++ SGP4 code. This is perhaps the most accepted codification of the SGP4 theory that is used to generate ephemeris data from TLEs (Vallado et al. 2006). Once the code has been modified to suit the study needs, the cross-referencing task will not be required.

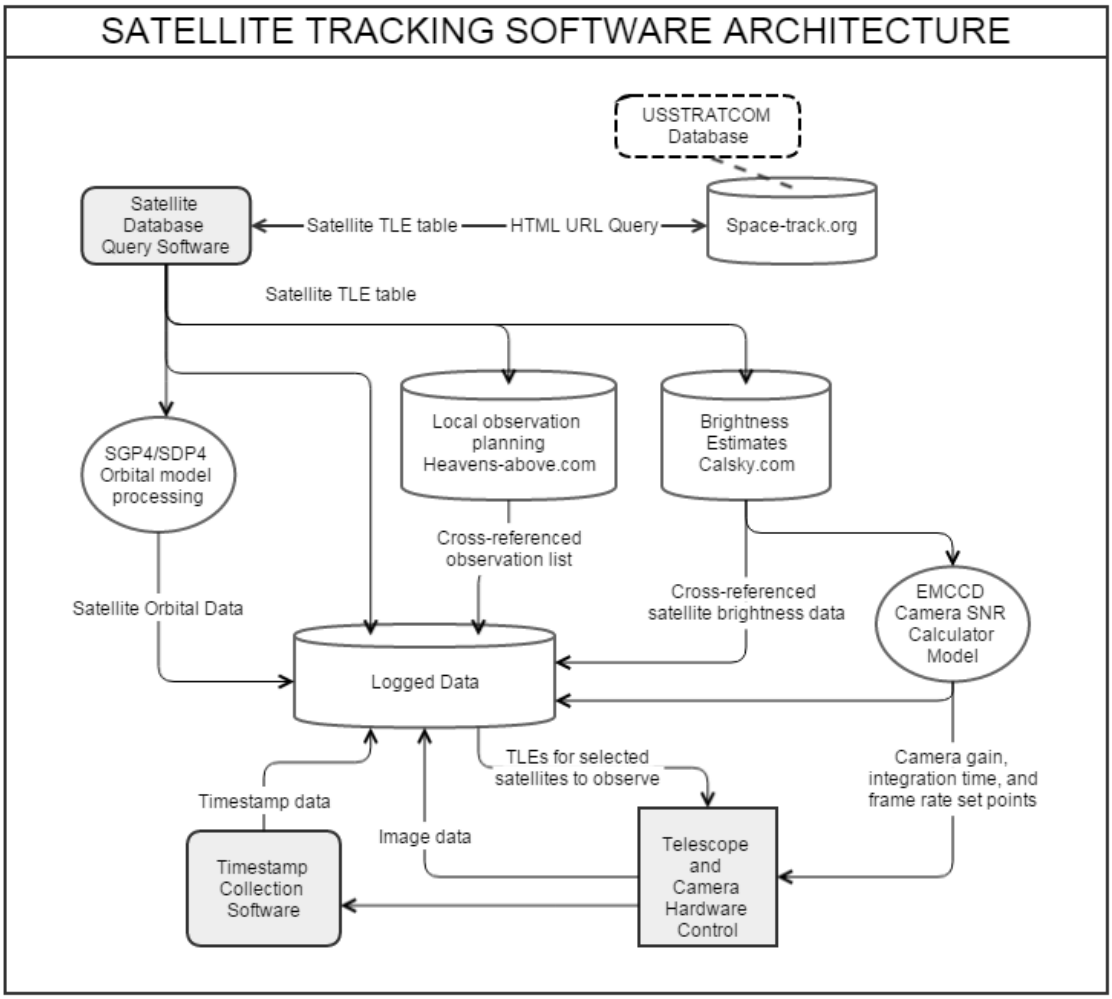



**Figure 5**. Telescope Control and Image Acquisition Architectures Developed for RSO Tracking

## 2.3 EMCCD Camera Optimization and Signal-to-Noise (SNR) Model

A goal of this SSA study was to evaluate EMCCD camera technology for RSO astronomical photometry measurements. The ultimate noise limit of a conventional charged-coupled device (CCD) imager is determined by the readout noise and dark current noise. Cooling the imaging array can minimize dark current noise; however, readout noise continues to increase as a function of pixel readout speed. Unlike CCD cameras, the EMCCD design provides user-selectable internal gain before the charge reaches the output amplifier. Gain from this low-noise charge amplifier can boost weak signals well above the readout noise floor to maintain EMCCD imaging performance (i.e., SNR), even at high frame rates. This unique feature is ideally suited for many RSO photometry measurements. Images can be collected at fast frame rates, and then post-processed to remove image frames taken during unsteady atmospheric conditions. A subset of the remaining frames can be summed to improve the photometry SNR. Image processing software is readily available to automate these steps.

PNNL also developed an EMCCD camera SNR model to evaluate the impact that local observing conditions and camera configurations have on the image acquisition noise budget. The radiometric calculations used to develop this modeling tool are summarized in this section and based on prior work (Ferrero et al. 2010; Shell 2010). Astronomical objects are specified in terms of their relative brightness on a visual magnitude scale. The object's visual magnitude, $M_s$, is first converted to irradiance using

$$I = (1.78 * 10^{-8}) * 10^{-.4M_s} \ \ \frac{Watts}{m^2}, \tag{1}$$

where the irradiance of zero-magnitude source is assumed to be $1.78 * 10^{-8}$ Watts/m. The photon flux (photons/sec) on the telescope aperture is given by

$$P_h = \frac{I*A*T_a}{E_p} \text{ photons/sec}, \tag{2}$$

Here $E_p$ is the energy per photon (at 625 nm), $A$ is the area of the telescope aperture, and $T_a$ is the atmospheric transmittance. The number of photoelectrons produced by the camera, due to the object's photon, is given by

$$P_e = P_h * Q_e * T_t * \tau \text{ photoelectrons}, \tag{3}$$

where $Q_e$ is the camera quantum efficiency, $T_t$ is the optical transmittance of the telescope, and $\tau$ is the exposure time in seconds. The total number of signal electrons produced by the charge-integrating amplifier is given by

$$S_e = P_e * G \text{ electrons}, \tag{4}$$

where $G$ gain given in units of signal electrons/photoelectrons.

The signal's shot noise is calculated as follows:

$$\sigma_s = G * \sqrt{F * P_e} \text{ electrons}, \tag{5}$$

where $F$ is the noise factor due to the EMCCD gain process. The noise factor ranges from 1 to 2. This calculation assumes a background signal subtraction, by collecting a second background image, to assure that electrons from each object pixel represent only the object signal. Noise added by the background subtraction is not included in this analysis.

### 2.3.1 Background Noise Considerations

The ability to see faint objects is limited by the local sky brightness due to light pollution and atmospheric conditions. Background light can produce the largest source of imaging signal noise. Observation sites are generally ranked by the empirically determined naked eye limiting magnitude (NELM), which refers to the faintest stars that can be observed by the unaided, dark-adapted eye. NELM limitations, due to light pollution, airglow, indirect sunlight scattering, and starlight scattering, varies greatly depending on the observing site location and time. The magnitude scale is logarithmic, with apparent brightness of an object decreasing by a factor of approximately 2.512 with increasing magnitude

integer values (Pogson 1856). For example, the bright star Sirius has an apparent magnitude of −1.47, while the faint planet Neptune has an apparent magnitude of about 8.

A light polluted inter-city location (i.e., NELM = 2) may have only about 50 stars visible, while a rural location (i.e., NELM = 6) may have more than 3000 stars visible. Typical PNRO site NELM is about magnitude 6.25. The Moore Observatory is situated on the campus of CBC in Pasco, Washington. Light pollution from the surrounding campus and city degrades the NELM to about magnitude 4. Background radiance conditions are often specified in terms of sky brightness (also known as apparent background surface brightness), with units of visual magnitude per arcsecond squared. Sky brightness can be estimating knowing NELM (Schaefer 1990). Neither NELM nor sky brightness units are useful radiometric terms; however, sky brightness can be converted into irradiance/steradian using

$$I_b = (5.6 * 10^{10}) * 10^{-.4M_b} * \left(\frac{180}{\pi}\right)^2 * 3600^2 \ \frac{photons}{sec*m^2*sr}. \quad (6)$$

The number of background photoelectrons produced by each camera pixel is

$$P_b = I_b * \Omega * x^2 * T_t * \tau * Q_e \text{ photoelectrons/pixel}, \quad (7)$$

where the telescope solid angle is $\Omega = \pi/(1 + 4\left(\frac{f^2}{d^2}\right))$ steradians, $d$ is the telescope diameter, $f$ is the telescope focal length, and $x$ is the pixel length (assuming square pixels).

The background noise in terms of signal electrons is given as

$$\sigma_b = G * F * \sqrt{P_b * N} \text{ electrons}, \quad (8)$$

where $N$ is the total number of pixels encompassing the object.

### 2.3.2 Camera Measurement Noise

The process of shifting charge carrier to the camera's analog-to-digital converter produces readout noise, $\sigma_r$. Readout noise was experimentally determined to have a maximum value of 20, as discussed in Appendix F. The total readout noise is given by

$$\sigma_T = \sqrt{N * {\sigma_r}^2} \text{ electrons}. \quad (9)$$

The camera measurement includes other sources of noise, including dark current and clock-induced-charge noise. However, these noise sources are generally negligible compared to the background and shot noise.

### 2.3.3 Signal to Noise Calculation

The final SNR value is given as the total number of signal electrons divided by the quadrature addition of all relevant noises,

$$SNR = \frac{S_e}{\sqrt{\sigma_s^2+\sigma_b^2+\sigma_T^2}}. \quad (10)$$

# 3.0 Observation Results

The GEO satellite, Galaxy 14 (USSPACECOM # 28790), was selected as a test case to study acquisition, tracking, and image collection at the Moore Observatory. Galaxy 14 is a C-band satellite built for PanAmSat Corporation to distribute entertainment and information to cable television systems. This satellite was launched in 2005 and placed in orbit at about 125° west longitude, at ~35,780 km orbital height. The satellite has a 20 $m^2$ radar cross section that provides an apparent visual magnitude typically ranged between 11 and 14 magnitude. In comparison, Pluto has an average apparent magnitude of about 15.

Observations were conducted on Galaxy 14 on October, 8, 2014, starting at about 8 PM PDT. Images were collected using the EMCCD camera, while synchronous UTC timestamps were being recorded from the time code generator. SkyX was used to select this satellite and then slew the telescope to this object to begin image collection. The satellite was not initially in the camera FOV, but was seen using the wide-field finder scope. It was possible that this was due to *TPoint* pointing error model limitations or the camera's massive weight (3.5 kg), which unbalanced the load on the Paramount ME mount. The SkyX software joystick was then used to center the satellite on the camera array. After centering, the satellite remained within the camera FOV during the 20-minute observation time. Future observing sessions will require careful camera/telescope load balance and an updated *TPoint* error model.

Figure 6 shows a sequential image mosaic of Galaxy 14, collected over a period of 38 seconds. These images were hand-selected out of a much larger collection (~2600) taken that night. The first image was collected at 8:31:09 PM local time and the following images are shown at about 2-second intervals. The last frame was taken at 8:31:23 PM. The first frame on the left shows the fixed location of Galaxy 14 and the red arrow shows the path of the star field as it drifts through the subsequent frames. Image post-processing was conducted by selecting a region-of-interest around Galaxy 14, then extracting an average signal value for a series of frames. Light curves were then generated, but no discernable light variations were detectable. Glint peaks have been detected from Galaxy 12 and 15, but the glints occurred infrequently and the peak widths were greater than 10 minutes (Hall and Kervin 2013).

A small subset of the 2600 image frames collected were selected and manually aligned to show the apparent track of Galaxy 14, as shown in Figure 7. The image sequence encompasses about 38 seconds elapsed time. The gray scale on each image was inverted using a color negative variant to improve the clarity of the faint objects (now black objects in the image). Next, all the images were aligned into a panoramic image by registering the local star-fields. The position of Galaxy 14 is highlighted in each frame by a dark blue dot and a track line (shown in cyan) applied by connecting the first and last satellite position. As can be seen from the image, the intermediate satellite locations appear above and below the track line. It is likely that registration error account for some of these offsets, but additional factors may be responsible, such as atmospheric turbulence.

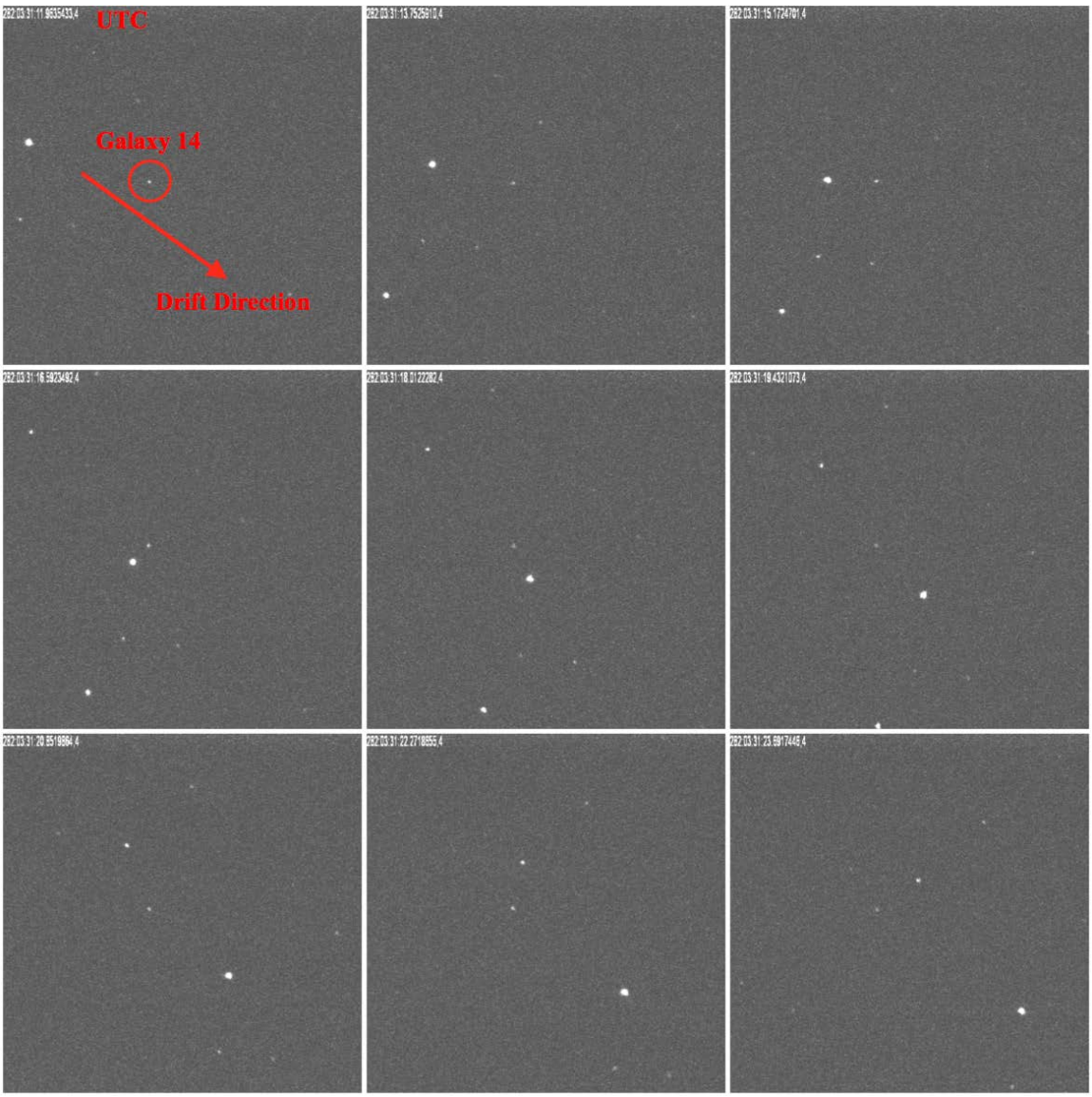



**Figure 6**. Sequential Image Frames Showing GEO Satellite, Galaxy 14, Near the Center of the Frame, with the Star Field Drifting Diagonally Left to Right

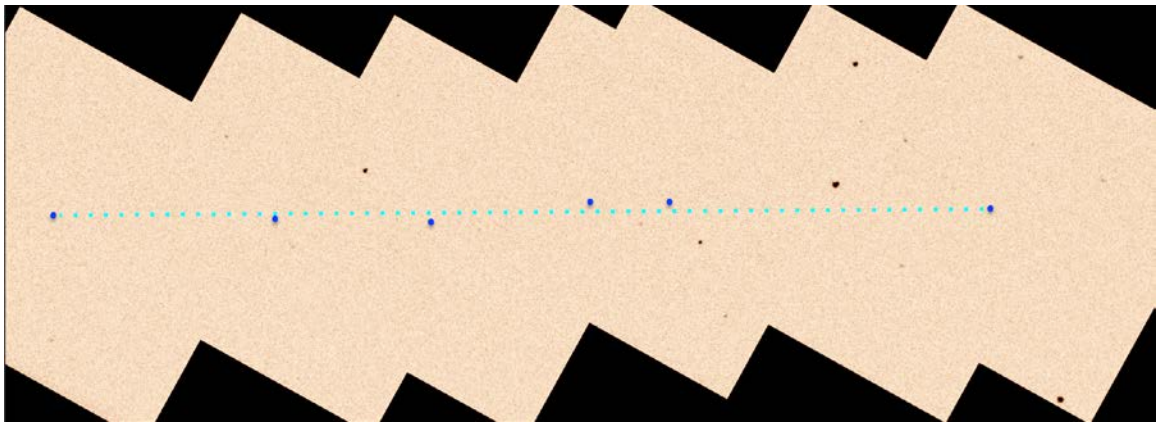


**Figure 7**. Sequential Image Frames Showing the Galaxy 14 Apparent Orbital Path (dotted line), with the Star Field Moving Left to Right. The satellite positions are shown as dark blue dots to improve the clarity.

Photometry measurements can be used to estimate RSO apparent brightness. The apparent brightness of Galaxy 14 was estimated by comparing the brightness of a nearby star. Star number GSC 5189:2425 appeared to have slightly lower relative image brightness on the same processed image frame. Published average brightness for this star is 12$^{th}$ magnitude. Calsky estimated Galaxy 14's apparent brightness during this observation time at 13.8 magnitude. Two unresolved stars (UCAC3 169.291215 and 169.291196) are shown to the left of Galaxy 14 in Figure 8. These stars have published 11.5 and 10.9 magnitude values. The combined apparent brightness is 10.4 magnitude, suggesting that these binaries were about 4 times brighter than GSC 5189:2425 and 23 times brighter than Galaxy 14.

Because pixel count is linearly related to stellar brightness, the known magnitude and signal count of a reference star can be used to estimate the magnitudes of Galaxy 14 and GSC 5189:2425. Image processing software was used to select a region of interest around each object to extract total signal count. Within the same image frame, a similar region of interest without stars was selected to determine the background signal count. The background was subtracted from the object count to provide a background-corrected signal count for each object. The relative photometric values indicated that Galaxy 14 produced 1.5 times greater signal compared to GSC 5189:2425. The binaries were found to have 34 and 52 times greater signal compared to Galaxy 14 and GSC 5189:2425, respectively. This suggests that Galaxy 14 and GSC 5189:2425 had apparent magnitudes 14.3 and 14.7, respectively. While this analysis underestimated the reported magnitudes (likely because of low measurement SNR and poor atmospheric conditions), these photometry measurements could be optimized and automated to provide accurate and precise RSO apparent magnitude estimates.

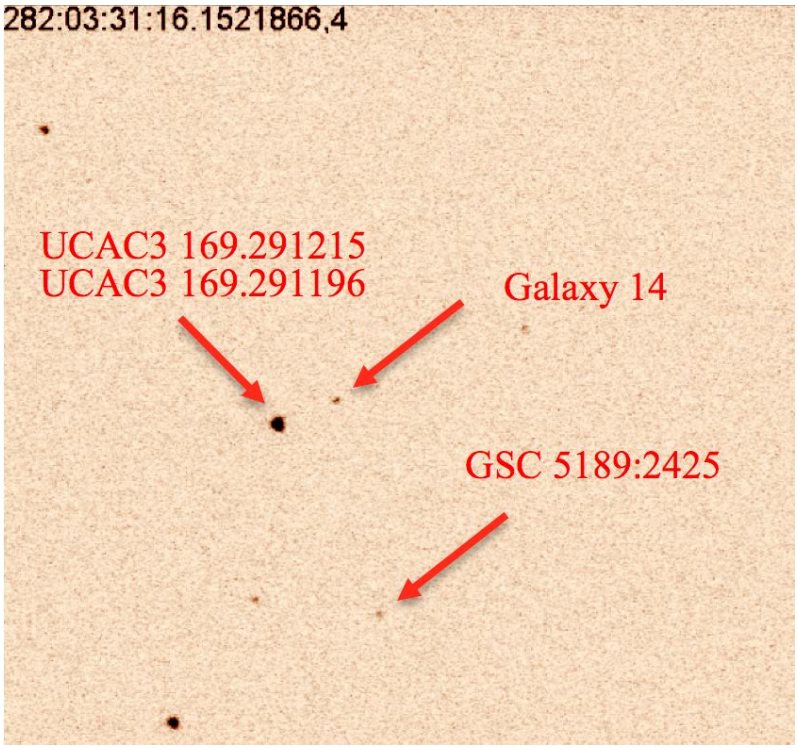



**Figure 8**. Galaxy 14 Apparent Brightest Estimated using a Reference Star and a Nearby Binary Star

# 4.0 Summary

**This SSA study demonstrated the feasibility of using small (<1 m) optical telescopes to collect time-resolved RSO measurements that could be used in the future to protect satellite assets, detect and monitor orbiting debris fields, and support Outer Space Treaty monitoring and transparency.** The EMCCD camera used in this study offers the potential to collect high frame rate images of RSOs, while effectively eliminating a dominant noise source that plagues conventional CCD technology. The study also capitalized on commercial hardware and software used by the advanced amateur astronomy community. A telescope and camera hardware and software architecture was developed to provide accurate telescope pointing and tracking and RSO image data collection. This SSA study also developed extensive custom software to plan RSO observations, model the measurement noise budget, and support photometry data collection and processing. A very faint satellite (Galaxy 14) was successfully acquired and tracked using the 16-in. telescope at the Moore Observatory. Photometric data was collected and used to correlate object orbital position as a function of atomic clock-derived time. Object apparent brightness was estimated by image analysis and nearby star calibration. The measurement performance was only limited by weather conditions, object brightness, and the sky glow at the observation site. These observations were enabled through a unique collaborative effort between PNNL and several regional academic institutions and observatory facilities.

Future work has been identified to further demonstrate and extend the techniques developed by this research team, including a need for the following tasks:

- Conduct further and more extensive satellite and space debris observational studies at collaborating regional observatories
- Demonstrate and develop further methods to acquire SSA data from very small targets moving at high angular velocities
- Complete and demonstrate high-speed, arcsecond-precision telescope tracking system on the Hoch telescope
- Complete orbital propagation software code needed to generate target orbital data and plan observation studies
- Develop architecture design concepts for a fully automated SSA observation station
- Collaborate with commercial vendors to improve the accuracy of the satellite tracking software
- Evaluate other imaging camera and other commercial technologies for SSA applications
- Explore options for technical exchange between the U.S. and Russian SSA communities to discuss results of this project and similar efforts by Russia.

# 5.0 References


Ferrero A, R Felletti, L Hanlon, J Campos and A Pons. 2010. "Electron-multiplying CCD Astronomical Photometry." In *Proceedings of SPIE 7536, Sensors, Cameras, and Systems for Industrial/Scientific Applications XI*, pp. 75360Q-75360Q-11. January 17, 2010, San Jose, California. The International Society for Optical Engineering, Bellingham, Washington.

Hall D and P Kervin. 2013. "Analysis of Faint Glints from Stabilized GEO Satellites." In *2013 Advanced Maui Optical and Space Surveillance (AMOS) Technical Conference*. September 10-13, 2013, Kihei, Maui, Hawaii. The Maui Economic Development Board, Maui, Hawaii. Boeing report number #377ABW-2013-0638, OPS-13-5157. Available at www.amostech.com/TechnicalPapers/2013/NROC/HALL.pdf.

Knight B. 1979. "An Efficient Gearless Telescope Drive." *Journal of the British Astronomical Association* 90:45.

National Research Council. 1995. *Orbital Debris: A Technical Assessment*. National Academy Press, Washington, D.C.

Pogson N. 1856. "Magnitudes of Thirty-six of the Minor Planets for the First Day of Each Month of the Year 1857." *Monthly Notices of the Royal Astronomical Society* 17:12-15.

Schaefer BE. 1990. "Telescopic Limiting Magnitudes." *Publications of the Astronomical Society of the Pacific* 102(648):212-229.

Shell J. 2010. "Optimizing Orbital Debris Monitoring with Optical Telescopes." In *2010 Advanced Maui Optical and Space Surveillance (AMOS) Technical Conference*, p. E42. September 14-17, 2010, Wailea, Maui, Hawaii. The Maui Economic Development Board, Maui, Hawaii.

Vallado DA, P Crawford, R Hujsak and TS Kelso. 2006. "Revisiting Spacetrack Report #3." In *AIAA/AAS Astrodynamics Specialist Conference*, pp. 1984-2071. August 21-August 24, 2006, Keystone, Colorado. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia.

# Appendix A

# SkyX Bump.exe Program

# Appendix A

# SkyX Bump.exe Program

## A.1 Description

An advanced SkyX satellite tracking add-on is under development at Software Bisque that has enhanced satellite orbital propagator code and telescope pointing and tracking accuracy. The add-on also includes a software-based joystick that can be used to center the RSO target within the telescope or camera field of view (FOV). PNNL developed a hardware joystick solution using a USB numerical keypad and a Python program to send SkyX bump commands through a TCP/IP socket connection protocol. The SkyX Bump.exe Python GUI reads keys 8, 5, 4, and 6 on a USB numeric keypad, then sends Javascript to SkyX for the up, down, right, and left bump commands using a TCP/IP socket connection protocol. One advantage of this approach is that this program does not need to run on the same SkyX computer, because it uses a TCP socket connection.

## A.2 How to Use

1. First turn on SkyX's TCP server by going to Tools>TCP Server. A new dialog window should open up and make sure to click on the checkbox labeled, "Listening for connections"; you can close the dialog window once the checkbox has been marked.
2. Next run the BumpControls.exe program located in the BumpControls/dist folder. This will initiate SkyX Bump.exe. Once this GUI is open, type in the port number used to connect to SkyX's TCP server. The port number is shown in SkyX's TCP server dialog window. Once connected, the indicator should change to green and the textbox should indicate a positive connection status as well as the version number of the Raven3 object. Raven3 is the software object used to control the bump offsets. The TCP server dialog window should also indicate the connection as well. A "ReferenceError. Cannot find variable: Raven3" message indicates that the advance satellite tracking add-on is not installed with SkyX.
3. SkyX Bump.exe is now ready to receive key commands from the USB keypad. The SkyX bump up, down, left, and right commands are bound to the 8, 5, 4, and 6 keys, respectively, on the numeric keypad. Pressing any of these keys during satellite tracking will command SkyX to bump the telescope in the appropriate direction by a given amount. The size of the bump offset can be set in either the SkyX Track Satellite window or by adjusting the slider in the client GUI.
4. To disconnect, press the disconnect button and the client should disconnect from the server. Once disconnected, the USB keypad controls will be unbound from the SkyX application.

NOTE: When SkyX and SkyX Bump.exe run on the same computer, make sure that the SkyX client window is the active window when using the bump controls. If the client window is not the active window, the GUI will no longer associate the key presses with appropriate commands. Also SkyX Bump.exe is unable to differentiate key presses between the computer keyboard and the USB keypad.

```
'''----------------------------------------------------------------------
BumpControls.py
Written by Cliff Chen, PNNL

This program creates a simple TCP client to talk to the TCP server in SkyX and send Javascript
commands to control the bump controls during satellite tracking.  The commands are bound to keys on a
numeric keypad and are activated only when the client is connected.
----------------------------------------------------------------------'''

import socket, os
from Tkinter import *

class ClientWindow(Frame):
    def __init__(self,parent=None):
        self.root = parent
        self.connected = False

        Frame.__init__(self,parent)
        self.createWidgets()
        self.pack()

    def createWidgets(self):
        #create connection indicator
        self.connection_indicator = Canvas(self,bg='red',height=50,width=50)
        self.connection_indicator.grid(row=0,column=0,rowspan=2,sticky=E)

        #create the entry boxes for the server IP and port number
        self.iplabel = Label(self,text='Server IP:')
        self.iplabel.grid(row=0,column=1,sticky=E,pady=5)
        self.ipentry = Entry(self)
        self.ipentry.insert(0,socket.gethostbyname(socket.gethostname()))
        self.ipentry.grid(row=0,column=2,sticky=W,pady=5)
        self.portlabel = Label(self,text='Port:')
        self.portlabel.grid(row=1,column=1,sticky=E,pady=5)
        self.portentry = Entry(self)
        self.portentry.grid(row=1,column=2,sticky=W,pady=5)
        self.portentry.focus()

        #create the scale for specifying the size of the bump
        self.scalelabel = Label(self,text='Bump Size in Arcseconds')
        self.scalelabel.grid(row=3,column=0,columnspan=2,sticky=E+S,pady=5)
        self.scale = Scale(self,orient=HORIZONTAL,length=300,to=360)
        self.scale.grid(row=3,column=2,columnspan=2,sticky=W,pady=5)

        #create text box to receives outputs from the TCP server
        self.outputlabel = Label(self,text='Server Response:')
        self.outputlabel.grid(row=4,column=0,sticky=W)
        self.outputentry = Text(self)
        self.outputentry.config(height=5,width=55)
        self.outputentry.grid(row=5,column=0,columnspan=5,sticky=W+E)
```

```
        #create connect and disconnect buttons
        self.connectbutton = Button(self,text='Connect',command=self.connect2TCPServer)
        self.connectbutton.grid(row=0,column=3,sticky=W+E)
        self.disconnectbutton = Button(self,text='Disconnect',command=self.disconnectFromTCPServer)
        self.disconnectbutton.grid(row=1,column=3,sticky=W+E)

    #the key bindings are set up on the numeric keypad and organized to mimic the regular
up,down,left,right
    #arrow keys on a keyboard
    def bindControls(self):
        self.bind_all('<8>',self.moveUp)
        self.bind_all('5',self.moveDown)
        self.bind_all('4',self.moveLeft)
        self.bind_all('<6>',self.moveRight)
        self.scale.bind('<ButtonRelease-1>',self.readScale)

    def unbindControls(self):
        self.unbind_all('<8>')
        self.unbind_all('5')
        self.unbind_all('<6>')
        self.unbind_all('4')
        self.scale.unbind('<ButtonRelease-1>')

    #wrapper function to check connection status
    def checkConnection(func,*args,**kwargs):
        def check(self,*args,**kwargs):
            if self.connected:
                func(self,*args,**kwargs)
            else:
                self.printResponse('Not connected to SkyX TCP server.')
        return check

    #wrapper function to check for socket errors
    def check4SocketError(func,*args,**kwargs):
        def check (self,*args,**kwargs):
            try:
                func(self,*args,**kwargs)
            except socket.error as e:
                self.printResponse(os.strerror(e.errno))
                self.changeIndicator('red')
        return check

    def changeIndicator(self,color):
        self.connection_indicator.config(bg=color)

    #callback function to read the scale and update the bump size in skyx
    def readScale(self,event):
        bump_step = self.scale.get()
        self.sock.sendall("/* Java Script */ Raven3.bumpSizeArcSecs=%d; Out='Bump
size='+String(Raven3.bumpSizeArcSecs);"%(bump_step))
        self.recvCommandResponse()
```

```
    def clearOutputEntry(self):
        self.outputentry.delete("%d.%d" % (1, 0),END)

    def printResponse(self,message):
        self.clearOutputEntry()
        self.outputentry.insert("%d.%d" % (1,0),message)

    def connect2TCPServer(self):
        try:
            port = int(self.portentry.get())
            hostname = self.ipentry.get()
            self.sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            self.sock.settimeout(5)
            self.sock.connect((hostname,port))
            self.sock.sendall("/* Java Script */ var Out; Out = 'Connected. Raven3 Version:
'+String(Raven3.trackLEOVersion);")
            self.printResponse(self.sock.recv(2048))
        except ValueError:
            self.printResponse('Not a valid port number.')
        except socket.error as e:
            self.printResponse(os.strerror(e.errno))
        except socket.timeout:
            self.printResponse('Socket has timed out.')
        else:
            self.connected = True
            #bind controls once the client is connected
            self.bindControls()
            self.changeIndicator('green')
            self.focus()

    @checkConnection
    def disconnectFromTCPServer(self):
        try:
            self.sock.close()
        except socket.error as e:
            self.printResponse(os.strerror(e.errno))
        else:
            self.printResponse('Disconnected from SkyX TCP server.')
            self.connected = False
            #unbind the controls when the client disconnects
            self.unbindControls()
            self.changeIndicator('red')

    @check4SocketError
    def recvCommandResponse(self):
        chunks = []
        delimiter_detected = False
        while not delimiter_detected:
            chunk = self.sock.recv(2048)
            if chunk == b'':
```

```
                    #if the socket receives no more bytes, then connection has been terminated so indicate
that it has
                    self.printResponse('Disconnected from SkyX TCP server.')
                    self.changeIndicator('red')
                    return
                #using the | character as the delimiter in the messages received from the SkyX's TCP server
                elif '|' in chunk:
                    chunks.append(chunk)
                    if not chunk.endswith('.'):
                            chunks.append(self.sock.recv(2048))
                    delimiter_detected = True
                else:
                    chunks.append(chunk)

            self.printResponse(b''.join(chunks))

        #list of commands to be bound
        @check4SocketError
        def moveUp(self,event):
            if event.keycode == 104:
                self.sock.sendall("/* Java Script */ Raven3.trackLEOBumpUp();")
                self.recvCommandResponse()

        @check4SocketError
        def moveDown(self,event):
            if event.keycode == 101:
                self.sock.sendall("/* Java Script */ Raven3.trackLEOBumpDown();")
                self.recvCommandResponse()

        @check4SocketError
        def moveRight(self,event):
            if event.keycode == 102:
                self.sock.sendall("/* Java Script */ Raven3.trackLEOBumpRight();")
                self.recvCommandResponse()

        @check4SocketError
        def moveLeft(self,event):
            if event.keycode == 100:
                self.sock.sendall("/* Java Script */ Raven3.trackLEOBumpLeft();")
                self.recvCommandResponse()

#script to run and create the client window
if __name__ == '__main__':
    root = Tk()
    root.title('Bump Controls')
    front = ClientWindow(root)
    front.mainloop()
```

```
'''----------------------------------------

BumpControlssetup.py

py2exe setup file for building the BumpControls GUI

----------------------------------------'''

from distutils.core import setup
import py2exe

setup(windows=['BumpControls.py'])
```

# Appendix B

# Hamamatsu C9100-23b ImagEM X2 Camera Specifications

# Appendix B

# Hamamatsu C9100-23b ImagEM X2 Camera Specifications

| | | | | |
|---|---|---|---|---|
| **Type number** | | | | C9100-23B(ImagEM X2) |
| **Window** | | | | Anti-reflection (AR) coatings on both sides, single window |
| **Imaging device** | | | | Electron Multiplying Back-Thinned Frame Transfer CCD |
| **Effective number of pixels** | | | | 512 (H) x 512 (V) |
| **Cell size** | | | | 16um (H) x 16um (V) |
| **Effective Area** | | | | 8.19mm (H) x 8.19mm (V) |
| **Pixel Clock rate** | **EM-CCD Readout** | | | 22MHz, 11MHz, 0.6875 MHz |
| | **Normal-CCD Readout** | | | 0.6875MHz |
| **EM gain (typ)** | | | | 4x to 1200x |
| **Ultra-low light detection** | | | | Photon Imaging Mode 1,2,3 |
| **Fastest readout speed** | | | | 70.4 frame/s to 1076 frame/s |
| **Readout noise (rms) (typ)** | **EM-CCD Readout** | | **EM gain 4x** | 36 electrons at 22MHz |
| | | | | 25 electrons at 11MHz |
| | | | | 8 electrons at 0.6875MHz |
| | | | **EM gain 1200x** | 1 electron max at 22MHz |
| | | | | 1 electron max at 11MHz |
| | | | | 1 electron max at 0.6875MHz |
| | **Normal-CCD Readout** | | | 8 electrons at 0.6875MHz |
| **Full well capacity (typ)** | **EM-CCD mode** | | | 370000 electrons (CIC serious consideration) |
| | **Normal-CCD mode** | | | 140000 electrons (CIC serious consideration) |
| **Analog gain** | **EM-CCD Readout** | **22MHz** | | 1x |
| | | **11MHz/0.6875 MHz** | | 0.5x, 1x |
| | **Normal-CCD Readout** | | | 1x, 2x, 3x, 4x, 5x |
| **Cooling method/temperature** | **Forced- air cooled** | **at temperature control** | | -65C stabilized (0C to +30C) |
| | | **at slow scan** | | -75C (Room temperature: Stable at +20C, No binning) |
| | | **at maximum cooling typ.** | | -80C stabilized (Water temperature +20C) |
| | **Water cooled** | **at temperature control** | | -80C stabilized (Water temperature +20C) |
| | | **at maximum cooling typ.** | | -100C stabilized (Water temperature lower than +10C) |
| **Temperature stability (typ)** | | | | +/- 0.01C |
| **Dark current (typ)** | **Forced- air cooled (-65C)** | | | 0.005 electron/pixel/s |
| | **Water cooled (-80C)** | | | 0.0005 electron/pixel/s |
| **Clock induced charge (typ)** | | | | 0.0015 events/pixel/frame |
| **Exposure time** | **Internal synchronous mode** | | | 13.9ms to 1s (22 MHz) |
| | | | | 27.2ms to 2hours (11MHz) |
| | | | | 421.5ms to 2hours (0.6875MHz) |
| **Exposure time** | **External trigger mode** | | | 10us to 1s (22MHz) |
| | | | | 10us to 2hours (11MHz, 0.6875MHz) |
| **A/D converter** | | | | 16bit |
| **Output Signal/ External control** | | | | IEEE 1394b (Firewire) |
| **Binning** | | | | 2x2,4x4,8x8,16x16 |
| **External trigger mode** | | | | Edge trigger, level trigger, start trigger synchronous trigger |
| **Trigger output** | | | | Exposure timing output, programmable timing output (delay and pulse length are variable), trigger ready output |
| **Imaging processing features (real-time)** | | | | Background subtraction, shading correction, recursive filter, frame averaging, spot noise reducer |
| **EM gain protection** | | | | EM warning mode, EM protection mode |
| **Ambient Operating temperature** | | | | 0C to +40C |
| **Performance guaranteed temperature** | | | | 0C to +30C |

# Appendix C

# ORCA Technologies Synchronized Time Code Generator Model GS-101

# Appendix C

# ORCA Technologies Synchronized Time Code Generator Model GS-101

| | | | |
|---|---|---|---|
| **General Specs** | Position Accuracy | | <10 to 20 meters SEP (SA off) |
| | Timing Accuracy | | +/- 100 nanoseconds to UTC (GPS) |
| | GPS Input | | 1.575 GHz L1 C/A Code |
| | GPS Receiver | | 12 parallel channels |
| | Internal Oscillator | | TCXO <5X10^-9 (disciplined to GPS) |
| | Antenna | | L1 GPS with 5-meter SMA cable |
| **Fixed Inputs** | IRIG-B AM Serial Time Code Input | Format | IRIG-B 122 |
| | | Amplitude | 1 Vrms into 50 Ohms |
| | | Ratio | factory set to 3:1 |
| | | Connector | SMA labeled CODE INPUT |
| | | Termination | 50/10k Ohms, switch selectable |
| | IRIG-B DC Level Shift (DCLS) Serial Time Code Input | Specification | IRIG-B 002 |
| | | Amplitude | TTL levels |
| | | Connector | DB-15 multi-pin |
| **Fixed Outputs** | AM Serial Time Code Output | Format | IRIG-B 122, IRIGB 123 and IEEE-1344 |
| | | Amplitude | 1 Vrms into 50 Ohms |
| | | Ratio | factory set to 3:1 |
| | | Connector | SMA labeled CODE OUTPUT |
| | DLC Level Shift (DCLS) Serial Time Code Output | Specification | IRIG-B 002, IRIG-B 003, and IEE-1344 |
| | | Amplitude | TTL levels |
| | | Connector | DB-15 multi-pin |
| | 1PPS Output | Accuracy | <100 nanoseconds |
| | | Logic level | TTL into 50 Ohms |
| | | Timing | Positive edge on time |
| | | Duty cycle | 50% |
| | | Connector | DB-15 multi-pin |
| | Programmable Pulse Output | Logic level | TTL |
| | | Timing | Positive edge on time |
| | | Duty cycle | TBD |
| | | Connector | DB-15 multi-pin |
| | RS-232 I/O Port | Baud Rate | 9600-115200 |
| | | Output | Time, position, status and current settings |
| | | Input | Operating mode and setup parameters |
| | | Connector | DB-15 multi-pin |
| | 2nd RS-232 I/O Port | Outputs NMEA 0183 messages containing navigation and tracking information. Port will accept differential GPS real-time pseudo-range correction data in RTCM SC-104 format | |
| | USB Port | Output data: Time, position and status | |
| | Manual Control | Setup functions for operating mode, time, local and daylight savings time and programmable pulse | |
| | DC Power | 50 to V0 VDC <500 mWatts (can be powered through USB) | |

# Appendix D

# TSCapture.exe Program

# Appendix D

# TSCapture.exe Program

## D.1 Description

TSCapture.exe is a Windows-executable Python script that is used in conjunction with the Hamamatsu C9100-23B EM CCD camera and Hamamatsu HCImage Live program to capture timestamps from the ORCA GS-101 Timing Receiver module. The timestamps can be saved into a txt file and overlay onto corresponding images acquired from the HCImage program as well as saved into the images' properties information. The original images will be archived and the images that contain the timestamp overlay will be saved to a new directory. The program is capable of also compiling the images into a video with the codec and frame rate specified by the user.

## D.2 Installation

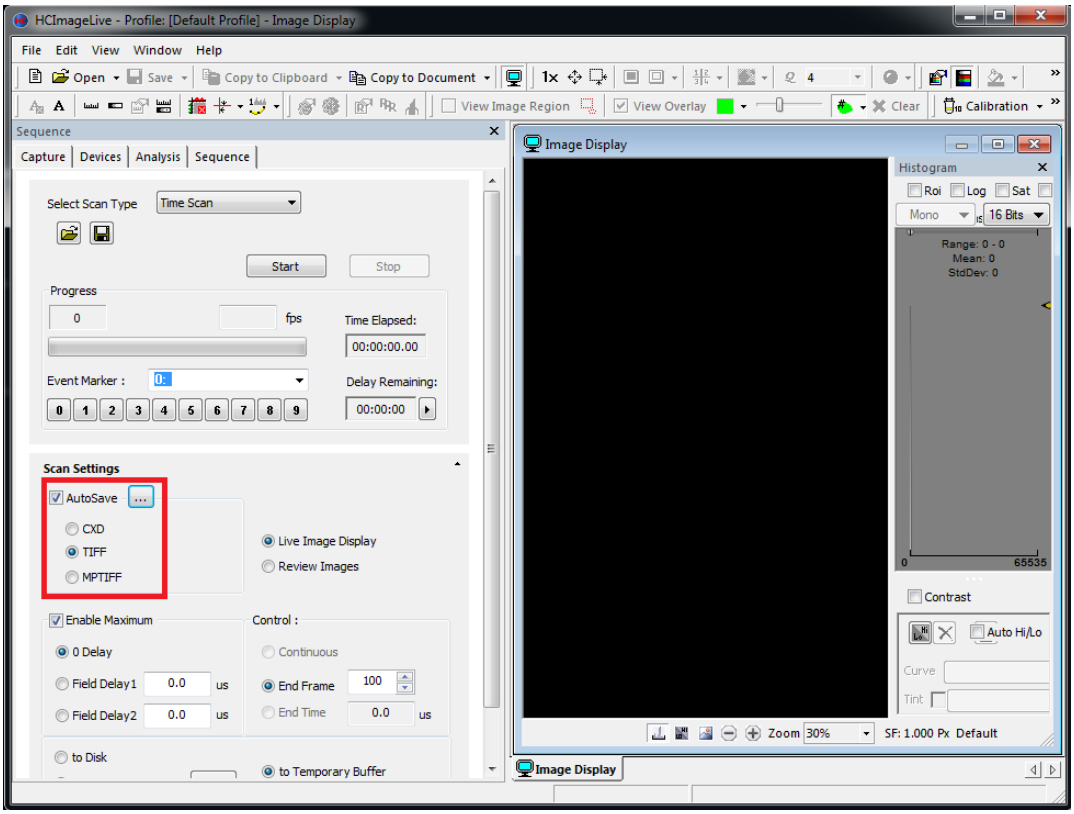
To install, copy the TSCapture directory into any location on the computer. Next, add the 'TSCapture/dist/ffmpeg' directory to system PATH variable. This defines the file path for the library of codecs use when compiling a video. If directory is not added to the PATH variable, then the program will not have video processing functionality and will only produce empty video files if executed. To run TSCapture.exe, navigate to the TSCapture/dist directory and launch the program.

## D.3 Capturing Timestamps

### D.3.1 Before Capturing: Configuring the HCImage Software

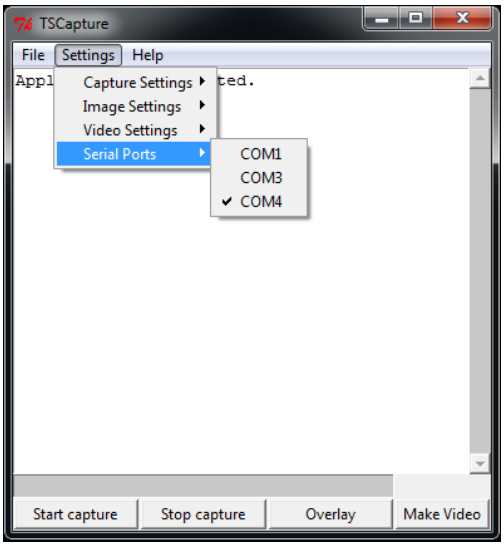
Within the HCImage application, configure the following:

1. **TRIGGERING:** Configure the trigger settings on the camera appropriately before running to the program.
2. **IMAGE FORMAT:** If overlaid timestamps are need, make sure that all the images are saved as TIFF images and not CXD or MPTIFF.
3. **FILENAME:** Make sure that the filenames of the images follow the format: (name)(delimiter)(image#)(extension) if processing overlaid timestamps. It is important that only the image's number within the set is between the delimiter and the file extension. Furthermore, it is important that only one instance of the delimiter appears in the filename; otherwise, the program will not sort the images properly. The HCImage program will automatically name the image according to this format with the delimiter being an underscore.
4. **SAVE LOCATION:** Make sure that all the appropriate images from one set are saved to their own directory if you want to overlay timestamps. The program will automatically grab all images with the indicated extension from the directory, so any extraneous images with the same extension will also be grabbed also, which will result in timestamps becoming out of sequence this their respective images.



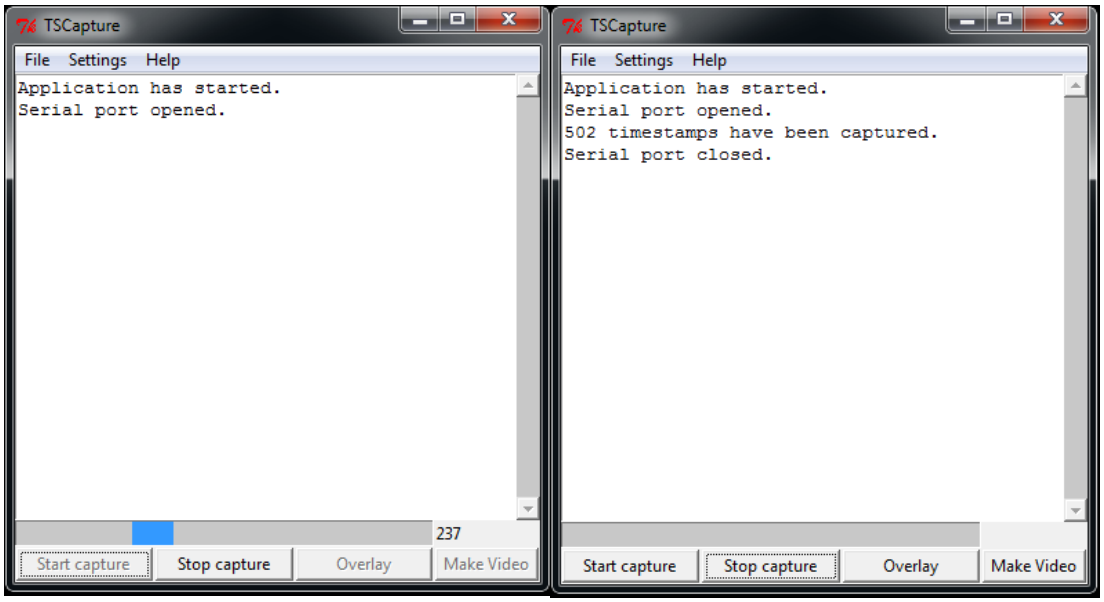

## D.3.2 Capturing

To capture, first select the serial port from the Settings>Serial Ports menu.




Once the serial port has been set, press the 'Start capture' button on the right in the application window to begin. The program will continuously read the serial port every 0.25 seconds to detect new timestamps when they are transmitted from the time code generator. When a timestamp is detected, it will be appended to an array to store the timestamps until saved by the user. The number of timestamps

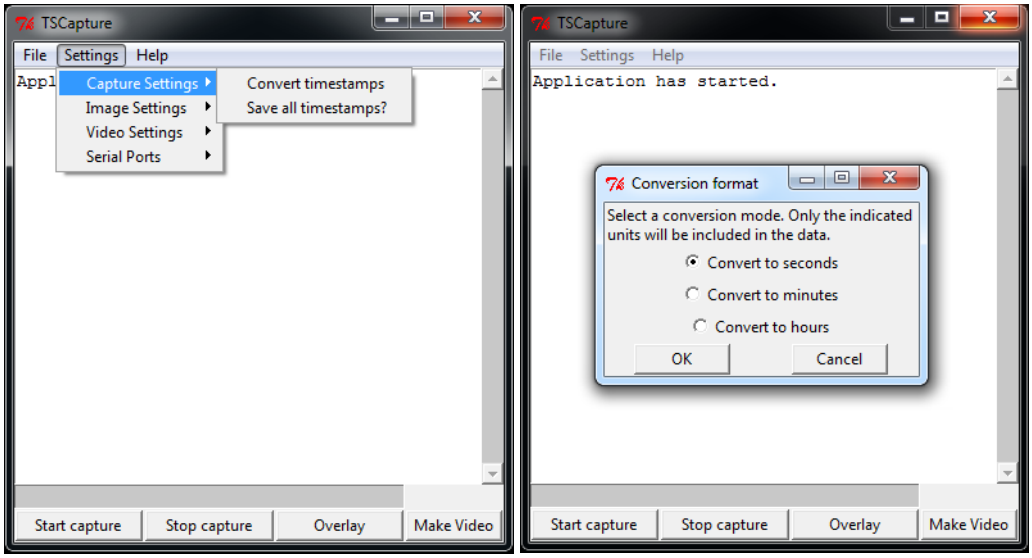
currently captured will be displayed in the lower right hand corner. To stop the capture, press the 'Stop capture' button to stop the capture loop and close the serial port.




**NOTE:** The program will almost always capture more timestamps than the number of images recorded because the EMCCD camera continues to send frame trigger pulses after in imaging sequence is completed, due to a timing delay in the HCImage application software.

### D.3.3 Capture Settings

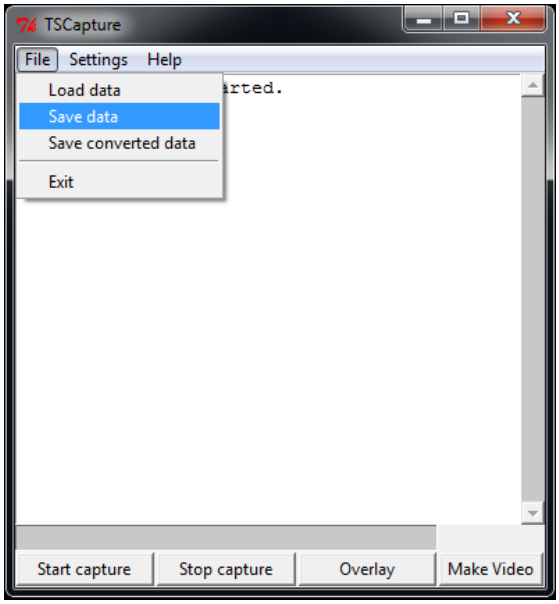
There is a list of options that the user can choose to control how the program captures and uses timestamps in the Settings>Capture Settings menu:



1. **CONVERT TIMESTAMPS:** Selecting this option will open up a dialog box that will ask to choose how to convert the timestamps. The conversion will only store the data in the units indicated and ignore any data with units larger than the indicated units. For example, if the minutes option is chosen, then only the minutes and seconds data will be retained (with the seconds data converted to fractions of a minute) and the remaining hour and day data will be not be used. The converted timestamps are stored separately from the raw data.
2. **SAVE ALL TIMESTAMPS:** Selecting this option will save all the collected including, any extra ones (see note above).

## D.4 Saving and Loading Timestamp Data

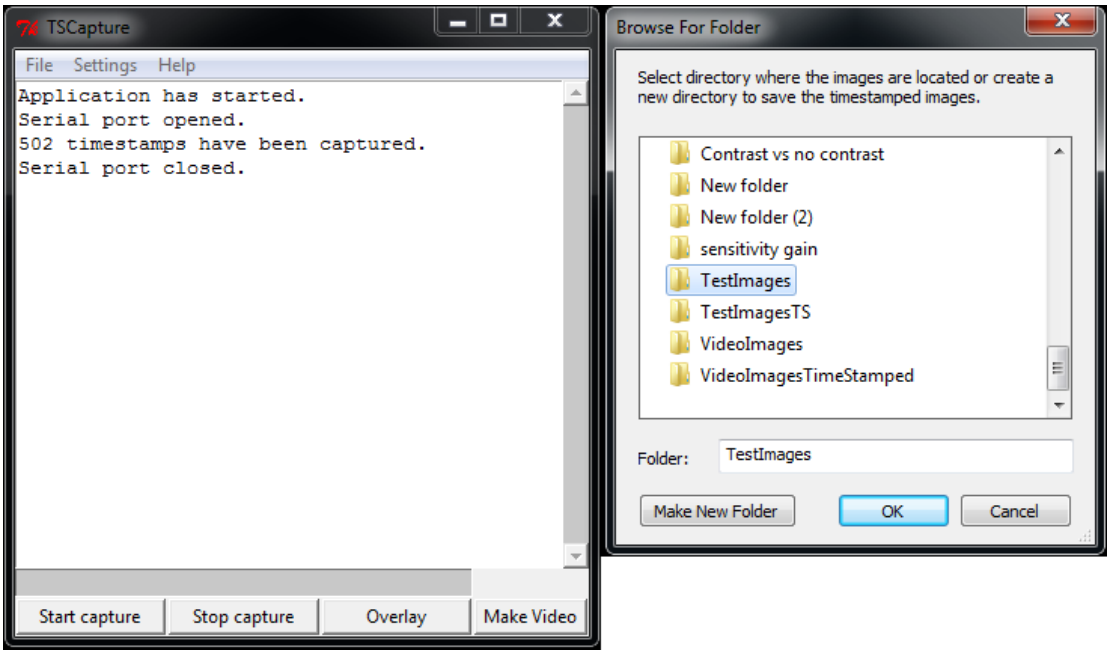
To save the original timestamp data, navigate to 'File>Save data' to open up a dialog box to select the file storage path. The file will be saved a .txt file. The number of timestamps saved will equal the number of image if, the user had previously overlaid timestamps on the images. Otherwise the program will save all the timestamps from the last image capture session. There is also an option to always save all the timestamps located in 'Settings>Capture Settings'. Selecting this option will always save all the timestamps collected. Selecting the 'Save converted data' will save any timestamp data that has been converted into the appropriate format of seconds, minutes or hours. Make sure to indicate the units in the filename to save confusion later.



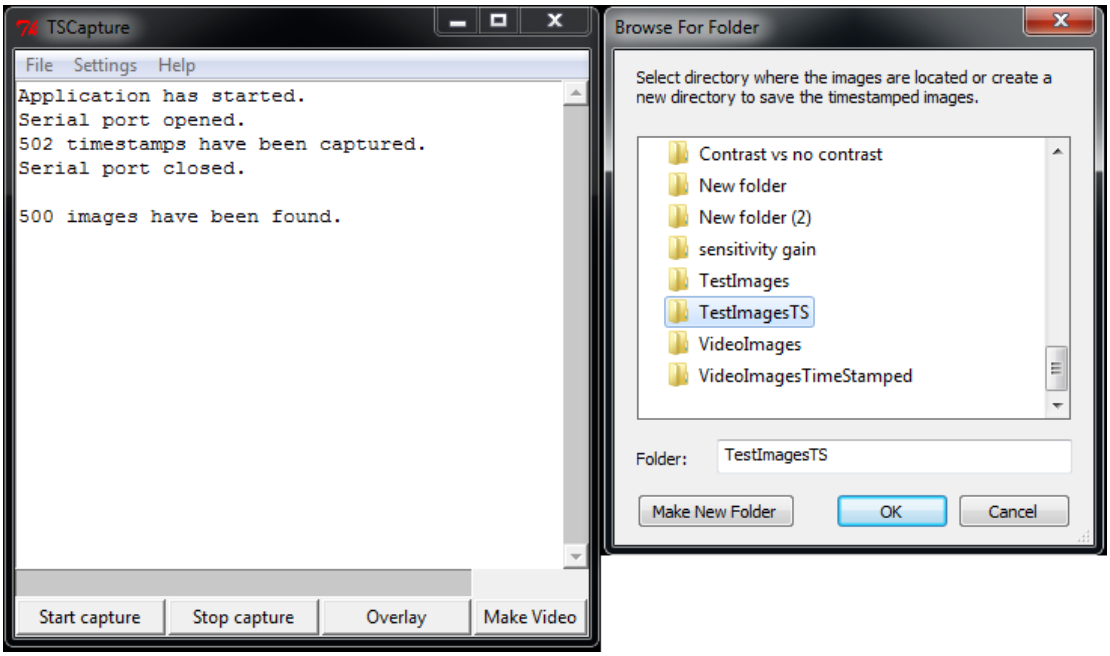

To load timestamps into program from a data file, navigate to 'File>Load data'. The program will read in the all timestamps in the data file and provide the number read in. The program may not read in the data correctly if the file data is not formatted the same way as it was originally created and you will not be able to use the ‘Convert timestamps’ option located in the Settings>Capture Settings menu.

## D.5 Overlaying Timestamps onto Images

After capturing or loading a set of timestamps into the application, TSCapture.exe can overlay the timestamps onto a set of images. This can be done by pressing the Overlay button. The program will then prompt the user to select the directory where the images are stored.

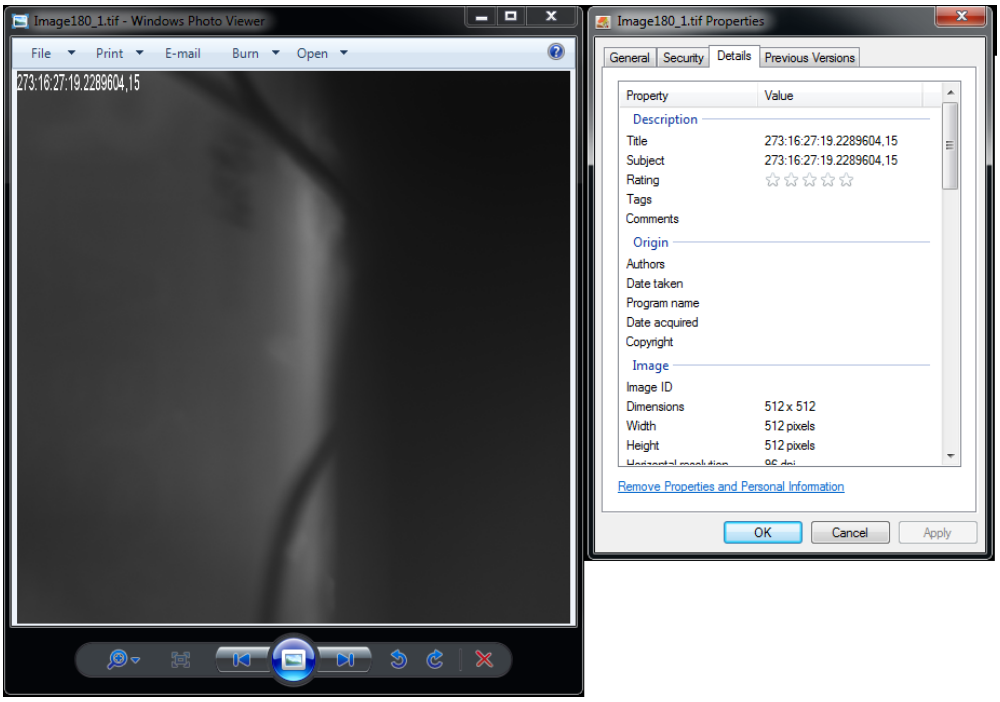



Once the user picks the directory, another dialog box will appear to ask the user which directory he or she wants to save the timestamped images into. It is important that the user select or create a new directory as the images will be saved with the original filename so any previous images may be overwritten.



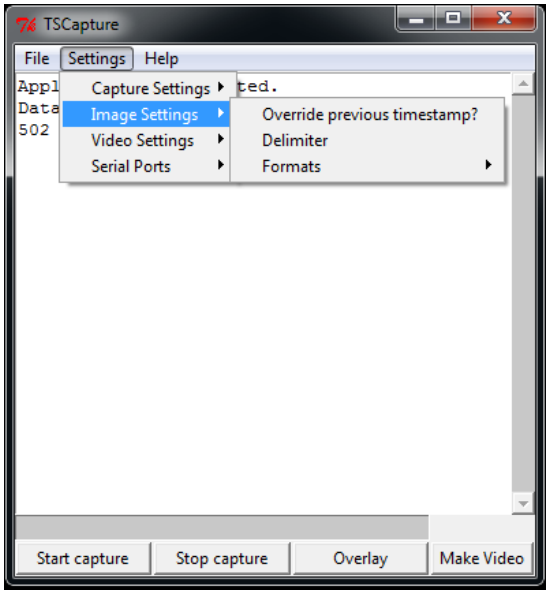

After the selections are made, the program will then go to the first directory to grab all the images with the appropriate extension and sort them numerically based on the image number given in their filenames. The program will then overlay the timestamp in the upper-left corner of the image before saving the timestamped images in another directory specified by the user. The number of successfully timestamped images will also be displayed to the user. The timestamp information will also be placed into the image's TIFF field for description if possible. The program will also automatically use the number of the images obtained to determine the number of timestamps to save when saving the timestamp data as a text file. Any extra timestamps are then automatically removed.

**NOTE:** Due to the fact that the image writing process is opened up as a separate process apart from the Tkinter GUI thread, the progress bar may not actually update at the same speed as the image writing. In other words, the program may finish overlaying the timestamps before the progress bar fills up completely. If this does occur, it should not have any effect on the rest of the program's usability, with the exception that if another process is ran before the progress bar is finished, the progress bar may update erratically and may generate an error.




## D.5.1 Image Options

There is a list of options that the user can choose to control how the program overlays the timestamps in the Settings>Image Settings menu:

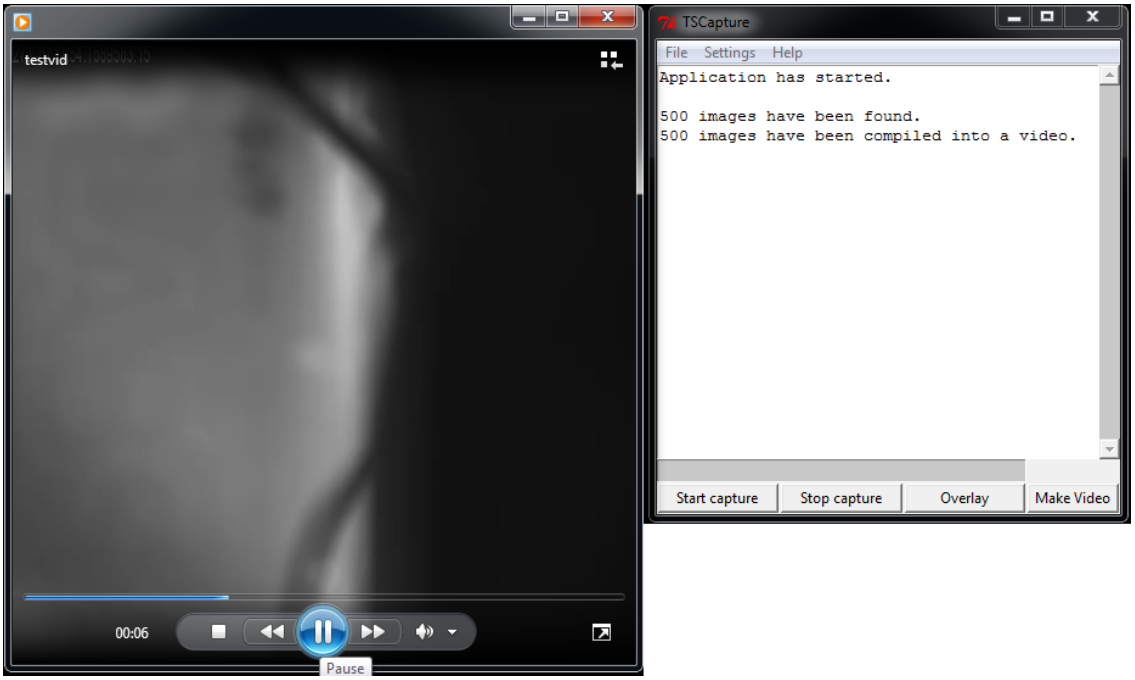



1. **OVERRIDE:** This option allows the user to override the previous timestamp on an image by covering it up with a black rectangle before placing the correct timestamp in its place. Use this option if the timestamp placed on an image is incorrect. This setting is turned off by default.

2. **DELIMITER:** This option will pop open a small dialog box that will prompt the user to input a delimiter to use when the program sorts the filenames numerically. The default value is an underscore.
3. **IMAGE FORMAT:** This option allows the user to choose what extension to use when the program grabs the images from the indicated save directory. The default value is .tif for TIFF images, which is the format that the user will be saving the images in when capturing images using the HCImage software.

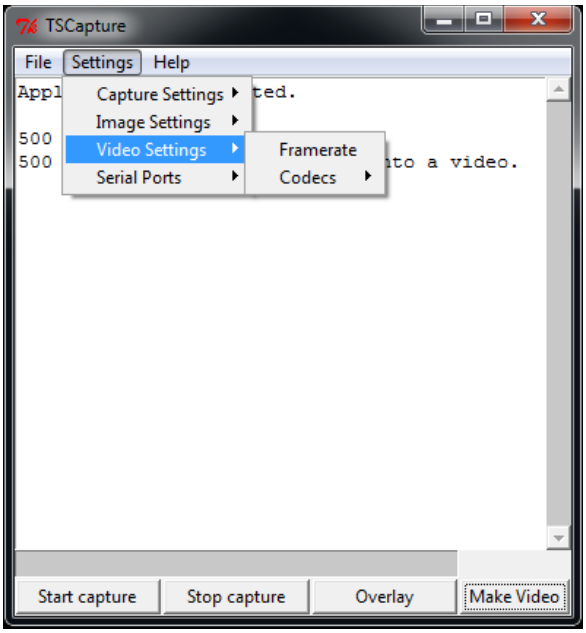
## D.6 Compiling Images into a Video

The program can combine a series of images into a video file. To compile a video, click the Make Video button. The program will first ask the user for the images location. The options for delimiter and file extension are set through the Settings>Image Settings menu. After the images are obtained and sorted numerically, the program will then query the user for the save location of the video file. The video file will automatically be saved using the .avi container, since this is a recognized Windows video format and will pose less problems when saving the video on a Windows machine. The program will then proceed to make the video and indicate the number of image successfully written to the video.




### D.6.1 Video Options

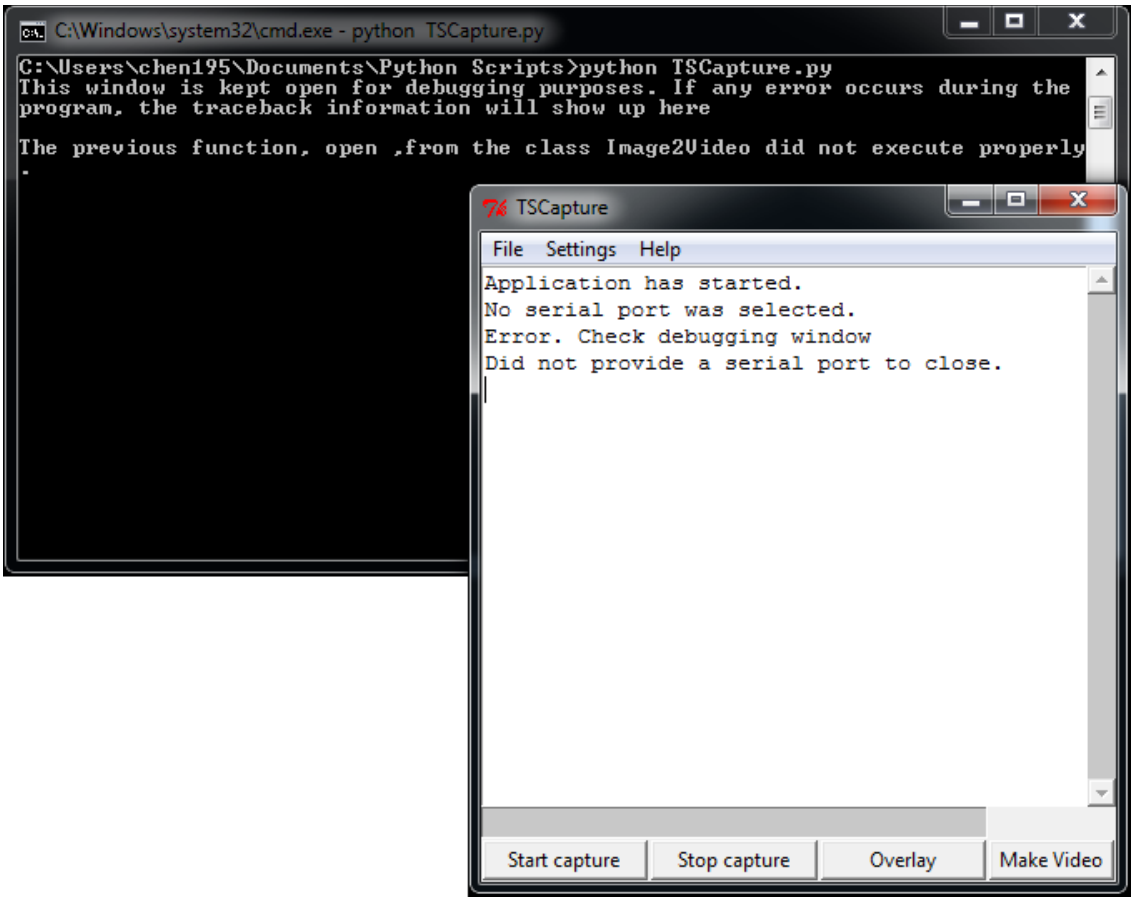
There are also two options the user can specify to control how the video is made in the Settings>Video Settings menu:




1. **FRAMERATE:** This option will pop open a dialog box that will ask the user the frames per second to use for the video. The default value is set at 24 frames per second.
2. **CODEC:** This option will list out the available codecs to use for compression when making the video. The codecs listed are the ones supported by the OpenCV VideoWriter class. Not all the codecs the class supports are listed. The default value is the DIVX codec.

## D.7 Error Handling

A terminal window will remain open while the program is running to display any traceback information about errors that occur during the program's execution. The source code for the program can be found in the TSCapture/source directory.




```
'''------------------------------------------------------------
setup.py
Writtten by Cliff Chen, PNNL

This is the setup file for building the GUI application using py2exe.  The setup file includes several image files and the html file used for program's reference.
------------------------------------------------------------'''

from distutils.core import setup
import py2exe, numpy

#images used in the reference manual
image_files =
['start_capture.png','convert_ts.png','finish_capture.png','debugging_window.png','hcimage.png','image_info.png','image_settings.png',

    'savenload.png','savets.png','serial_ports.png','tiffdir.png','video.png','video_settings.png','exposure_time.png',
                'capture_settings.png']

for i,file in enumerate(image_files):
    image_files[i] = 'ref_image/'+file
```

```
reference_files = [('reference',['TSCapture_reference.html']),
                   ('reference/ref_image',image_files)]

setup(console=['TSCapture.py'], data_files=reference_files)



'''------------------------------------------------------------

TSCapture.py

This is the driver module of the program. It starts the GUI and enters into the mainloop.

------------------------------------------------------------'''
from Tkinter import *
from TS import *
#need to explicitly include import numpy in the script as well in the setup.py script
#py2exe seems to have trouble importing some of the dlls on its own.
import numpy

print 'This window is kept open for debugging purposes. If any error occurs during the program, the
traceback information will show up here\n'
if __name__ == '__main__':
    root = Tk()
    root.title('TSCapture')
    front = FrontPanel(parent=root)
    root.mainloop()



'''------------------------------------------------------------------

TS.py

This is the 'main' module that contains all the classes used the program.

PROGRAM NOTES:
- Need to use 'import PIL.Image' instead of 'from PIL import Image' to avoid name space issues.
Otherwise, the program will import only
Image class and not the entire module
- OpenCV has issues with loading the ffmpeg library if the computer is running without ffmpeg support.
 To fix, rename the opencv_ffmpeg.dll file to opencv_ffmpeg(current_version).dll in the directory:
opencv\sources\3rdparty\ffmpeg
 and add the directory to the path variable. In this case, the library has been renamed to
opencv_ffmpeg249.dll
------------------------------------------------------------------------'''

import traceback, tkFileDialog, tkMessageBox, os, serial, glob, json, cv2, webbrowser, Queue, functools
from PIL import ImageFont, ImageDraw
import PIL.Image
```

```
from Tkinter import *
from ttk import *
from parse import *
from time import sleep
from multiprocessing.pool import ThreadPool


'''
Wrapper class that wraps another object's functions and checks if the output of a function is True, False or None.
If the output is False or None, then prevent any more functions from executing until we reset the error checker.
The class also accesses the wrapped object's list of messages to display to the textbox and posts which function
failed to execute properly in a sequence of commands in the debugging window.

NOTE: This wrapper class requires all functions in the wrapped class, public and private, to return a value when complete,
even if the function does not compute a value like a set function. If no value is returned, the wrapper class will catch
the null return and prevent any further functions in the wrapped class from executing until the error checker is reset.
'''
class TFNoneWrapper(object):
    def __init__(self,wrapped_class):
        self.wrapped_class = wrapped_class()
        #create the flag that prevents future functions from executing if a past one failed
        self.prevfunc_passed = True
        self.prevfunc_name = ''

    def __getattr__(self,attr):
        orig_attr = self.wrapped_class.__getattribute__(attr)
        if callable(orig_attr):
            name = orig_attr.__name__
            #wrapper function that checks if previous function passed
            def checker(*args,**kwargs):
                if self.prevfunc_passed:
                    #if passed set previous function name to current function name
                    self.prevfunc_name = name
                    #now execute and set result
                    result = orig_attr(*args,**kwargs)
                    try:
                        if result == False:
                            self.prevfunc_passed = False
                            return self.wrapped_class.messages[name]['false']
                        elif result == None:
                            self.prevfunc_passed = False
                            return self.wrapped_class.messages[name]['none']
                        elif result == True:
                            return self.wrapped_class.messages[name]['true']
                        else:
```

```
                                        return result
                                except KeyError:
                                        #return an empty string if no statement is found in the messages
dictionary
                                        return ''
                        else:
                                #if previous function did not pass, indicate that it did not
                                print 'The previous function, '+self.prevfunc_name+' ,from the class
'+self.wrapped_class.__name__+' did not execute properly.'
                                return 'Error. Check debugging window'

                return checker
        else:
                return orig_attr

'''
Base class that is capable of opening communication with the ORCA and obtaining timestamps
and saving them as well as loading them.
'''
class TimeStamp(object):
    def __init__(self):
        self.__name__ = 'TimeStamp'
        self.timestamps = []
        self.converted_timestamps = []
        self.numbercaptured = 0
        #dictionary of messages depending if the function returned a true,false,or none
        self.messages = {'open':{'true':'Serial port opened.','false':'Error in opening serial port.','none':'No
serial port was selected.'}
                                ,'close':{'true':'Serial port closed.','false':'Error in closing serial
port.','none':'Did not provide a serial port to close.'}
                                ,'save':{'true':'Data file saved.','false':'Error in saving data file.','none':'No
savepath was specified.'}
                                ,'load':{'true':'Data file loaded.','false':'Error in loading data
file.','none':'No file was specified.'}
                                ,'parseTimestamps':{'true':'Timestamps parsed into selected
format.','false':'Error occured during parsing timestamps.'}
                                ,'capture':{'false':'Error in reading serial port.'}
                                ,'convertTimestamps':{'none':'No timestamps to convert.'}}

    '''
    function opens up serial port connection to the specified comport and sends the auto-output enable
commands for the ORCA
    '''
    def open(self,comport):
        #commands to enable auto-output on the ORCA
        cmd = 'AOEN,1'.encode(encoding='ascii')
        cmd2 = 'AOMD,1'.encode(encoding='ascii')
        if comport == None:
            return None
        else:
            try:
```

```
                self.ser = serial.Serial(port=comport,timeout=10)
                self.ser.write(cmd)
                self.ser.write(cmd2)
            except (OSError,serial.SerialException):
                print 'Error in opening serial port to ',comport
                print traceback.print_exc()
                return False
            else:
                sleep(.25)
                self.ser.read(self.ser.inWaiting())
                return True

    '''
    function closes serial port connection
    '''
    def close(self,comport):
        if comport == None:
            return None
        else:
            try:
                self.ser.close()
            except (OSError,serial.SerialException):
                print 'Error in closing serial port ',comport
                print traceback.print_exc()
                return False
            except AttributeError:
                print 'No serial port was previously opened.'
                return False
            else:
                return True

    '''
    function captures timestamps output from the ORCA
    '''
    def capture(self,progress_queue,stop_queue):
        #clear the timestamp list for the new set of data
        self.timestamps = []
        self.numbercaptured = 0
        capturedThisLoop = 0
        timeStampTemp = []

        #the loop repeatedly reads the buffer to check if there is any new data.
        #if so, append the new data to a list. also, check the stop queue to see if
        #the user has indicated for the program to stop.
        while True:
            capturedThisLoop = 0
            try:
                timeStampTemp = self.ser.read(self.ser.inWaiting()).split()
                flag = stop_queue.get_nowait()
                if flag:
                        break
```

```
            except serial.SerialException:
                print "Error in reading serial port."
                print traceback.print_exc()
                return False
            except Queue.Empty:
                pass

            capturedThisLoop = len(timeStampTemp)
            self.numbercaptured += capturedThisLoop
            if capturedThisLoop > 0:
                self.timestamps.extend(timeStampTemp)
                progress_queue.put(self.numbercaptured)

            #now wait .25 seconds until the next buffer read
            sleep(.25)

        progress_queue.put(None)
        return self.numbercaptured

    '''
    parse the timestamps into the proper format
    '''
    def parseTimestamps(self):
        success = 0

        for n,i in enumerate(self.timestamps):
            try:
                self.timestamps[n] = parse('\x01{UTC}',i)['UTC']
            except KeyError:
                print "Error in parsing timestamp. Program will keep timestamp as is"
                #pass the value back into itself just in case the error affected the data
                self.timestamps[n] = self.timestamps[n]
            except TypeError:
                return success
            else:
                success += 1

        return success
    '''
    convert the timestamps to a particular unit: second, minute or hour
    The remaining portion of the timestamp that is in units larger than the one indicated
    will not show up as part of the data. The converted data will be saved separately from the
    raw data
    '''
    def convertTimestamps(self,unit='second'):
        if len(self.timestamps) == 0:
            return None
        #clear the previous values
        self.converted_timestamps = []
        success = 0
        parse_func = self.convert_mode(mode=unit)
```

```
        for n,ts in enumerate(self.timestamps):
            try:
                self.converted_timestamps.append(parse_func(ts))
            except KeyError:
                print "Error in parsing timestamp. Program will keep timestamp as is"
                #pass the value back into itself just in case the error affected the data
                self.converted_timestamps[n] = self.timestamps[n]
            except TypeError:
                traceback.print_exc()
                return success
            else:
                success += 1

        return success

    '''
    returns one of three functions to use for conversion depending on the mode specified
    '''
    def convert_mode(self,mode='second'):
        if mode == 'second':
            def convert_second(timestamp):
                result = parse('{day}:{hour}:{minute}:{second},{TFOM}',timestamp)
                return float(result['second'])
            return convert_second
        elif mode == 'minute':
            def convert_minute(timestamp):
                result = parse('{day}:{hour}:{minute}:{second},{TFOM}',timestamp)
                return float(result['minute'])+(float(result['second']))/60
            return convert_minute
        elif mode == 'hour':
            def convert_hour(timestamp):
                result = parse('{day}:{hour}:{minute}:{second},{TFOM}',timestamp)
                return float(result['hour'])+(float(result['minute']))/60+(float(result['second']))/3600
            return convert_hour


    '''
    load a data file containing timestamps
    '''
    def load(self,loadpath):
        if loadpath == None:
            return None
        else:
            try:
                with open(loadpath,'r+') as file:
                        self.timestamps = json.load(file)
            except IOError:
                print "Error with loading data"
                print traceback.print_exc()
                return False
            else:
```

```
                return True

    '''
    save the timestamp data into a text file. function returns None if no savepath was specified, false if an
error occurred
    and true if everything passed
    '''
    def save(self,savepath,length,mode='raw'):
        if savepath == None:
            return None
        else:
            try:
                with open(savepath,'w') as file:
                        if length == 0:
                                if mode == 'raw':
                                        json.dump(self.timestamps,file)
                                else:
                                        json.dump(self.converted_timestamps,file)
                        else:
                                if mode == 'raw':
                                        json.dump(self.timestamps[:length],file)
                                else:
                                        json.dump(self.converted_timestamps[:length],file)

            except IOError:
                print "Error with saving data"
                print traceback.print_exc()
                return False
            else:
                return True


'''
Class that inherits the TimeStamp class and adds on functionality to apply timestamps to images
'''
class TS2Image(TimeStamp):
    def __init__(self):
        TimeStamp.__init__(self)
        #image formats that PIL supports
        self.imformats = ('.bmp','.dib','.dcx','.eps','.ps','.gif','.im','.jpg','.jpeg','jpe','.pcd'

    ,'.pcx','.pdf','.png','.pbm','.pgm','.ppm','.psd','.tif','.tiff','.xbm','.xpm')

        self.imagelist = []
        self.override = False
        self.imformat = '.tif'
        self.delimiter = '_'
        self.__name__ = 'TS2Image'
        self.messages['grabAllFromDir'] = {'true':'Filenames obtained.','false':'Error in reading image
filenames.','none':'No directory was specified.'}
        self.messages['overlayTimestamps'] = {'none':'No save directory was specified.'}
```

```
    def setImFormat(self,format):
        self.imformat = format
        return True

    def setDelimiter(self,delimiter):
        self.delimiter = delimiter
        return True

    def setOverride(self,state):
        #explicitly state that we want a boolean value since the BooleanVar class in the Tkinter module
returns integers instead of boolean values
        self.override = bool(state)
        return True

    '''
    sort function to sort the filenames in numerical order.
    sort function only works if there is only one instance of the delimiter in the file name
    and if the only value between the delimiter and the file extension is an integer
    '''
    def sortImagelist(self,list):
        return sorted(list,key=lambda item:
int(item[item.index(self.delimiter)+1:item.index(self.imformat)]))

    '''
    function goes into a directory and grabs all the files with the corresponding extension
    and then sorts them
    '''
    def grabFromDir(self,dirpath,extension):
        if dirpath == None:
            return None
        current_dir = os.getcwd()
        try:
            #change to the indicated directory
            os.chdir(dirpath)
            self.imagelist = glob.glob('*'+extension)
        except OSError:
            print 'Error in grabbing images from directory'
            print traceback.print_exc()
            return False
        else:
            self.imagelist = self.sortImagelist(self.imagelist)
            #change the filenames to their absolute paths
            for i,file in enumerate(self.imagelist):
                self.imagelist[i] = os.path.abspath(file)
        finally:
            #go back to the original directory
            os.chdir(current_dir)
            return True
```

```
    '''
    function opens up an image, overlays the corresponding timestamp into the upper-left corner and save the image
    with the same filename in a new directory specified by the user
    function also provides an argument for a queue to send updates to a progressbar
    '''
    def overlayTimestamps(self,newdir,queue):
        if newdir == None:
            return

        #if for some reason the number of timestamps is less than the number of images, only overlay
        #images up to the number of timestamps. otherwise use the number of images found as the number to overlay
        if len(self.timestamps) < len(self.imagelist):
            length = len(self.timestamps)
        else:
            length = len(self.imagelist)

        #using this font type and size. Imagefont automatically checks the Windows font files if it can't find the file in the indicated directory.
        f = ImageFont.truetype('arial.ttf',20)
        success = 0

        for m,file in enumerate(self.imagelist[:length]):
            currentTime = self.timestamps[m]
            try:
                #open up the image in the current directory
                #program uses PIL to verlay the timestamp onto the image
                with open(file,'r+b') as i:
                        im = PIL.Image.open(i)

                        #access the TIFF image's description using the corresponding numerical TIFF tag value 270
                        #hammatsu stores the camera settings and what not for the image into this field
                        #to keep it, we need to read it out and manually re-save it back into the image
                        #currently not used
                        '''
                        if 270 in im.tag:
                                info = im.tag[270]
                        else:
                                info = ''
                        '''

                        #crop the region where the timestamp will be overlayed
                        region = im.crop((0,0,250,20))
                        draw = ImageDraw.Draw(region)

                        #if the override mode is true, erase the previous timestamp by covering it with a black rectangle
                        if self.override:
                                draw.rectangle([0,0,250,20],fill=0)
```

```
                        else:
                                pass

                        #now overlay the text containing the timestamp into the upper left corner of the
crop
                        draw.text((0,0),str(currentTime),fill=255,font=f)
                        im.paste(region,(0,0,250,20))

                        #save the image with the timestamp as the description
                        #into the new directory the user created
                        with open(os.path.join(newdir,os.path.basename(file)),'w+b') as newim:
                                im.save(newim,mode="L",description=str(currentTime))

                        im.close()
                        del draw

                except IOError:
                    print "Error with opening/saving ",k
                    print traceback.print_exc()
                    continue
                else:
                    success += 1
                    queue.put(success)

        queue.put(None)
        return success


'''
class that inherits the TS2Image class and adds on functionality to convert images to video
'''
class Image2Video(TS2Image):
    def __init__(self):

        TS2Image.__init__(self)
        self.__name__ = 'Image2Video'

        #list of available codecs supported by the OpenCV VideoWriter class. Not all the codecs the class
supports is listed here
        self.codeclist = ('divx','xvid','mp43','mp4s','fmp4','div1','mp4v','ump4','div5','mpg3','div6'

    ,'div1','h263','i263','h261','jpgl','ljpg','mjpg','flv1','i420','yuy2','y422'
                                        ,'yv12','uyvy','hdyc','iv32','y800')
        self.codec = 'divx'

        #list of available image formats that the OpenCV cv2 class supports for reading. Overwrites the
original imformats list in TS2Image
        self.imformats = ('.bmp','.dib','.jpg','.jpeg','.jpe','.png','.pbm','.pgm','.tiff','.tif')

        #add to the list of messages
        self.messages['writeVideo'] = {'none':'No save location was specified'}
```

```
        #create the video writer object
        self.videowriter = cv2.VideoWriter()

    def getAvailableCodecs(self):
        return self.codeclist

    def setCodec(self,codec):
        self.codec = codec
        return True

    '''
    writes a series of images to a video file. the only trusted option for video containers currently is .avi
    function also provides an option to include a queue to send updates to the progressbar
    '''
    def writeVideo(self,savepath,progress_queue,fps=24.0):
        if savepath == None:
            return None

        length = len(self.imagelist)

        fourcc = cv2.cv.CV_FOURCC(*self.codec)
        self.videowriter.open(savepath,fourcc,fps,(512,512),False)
        success = 0
        #try to write the images to the video file
        for image in self.imagelist:
            try:
                self.videowriter.write(cv2.imread(image,0))
            except (cv2.error, IOError):
                print 'Error in writing image to video file'
                traceback.print_exc()
            else:
                success += 1
                progress_queue.put(success)

        self.videowriter.release()
        cv2.destroyAllWindows()
        progress_queue.put(None)
        return success

'''
main class that handles the GUI and contains an instance of the Image2Video class
'''
class FrontPanel(Frame):
    def __init__(self,parent=None):
        self.root = parent

        #create and wrap the Image2Video object
        self.vid = TFNoneWrapper(Image2Video)

        #initialize all the values to empty values
```

```
        self.info = {'tstiffdir':None,'comport':None,'fps':24.0,'saveall':False,'delimiter': self.vid.delimiter,
                     'imagesgrabbed':0}

        #create the tracking variables for the menu options
        self.vars =
{'port':StringVar(),'codec':StringVar(),'override':BooleanVar(),'saveall':BooleanVar(),'imformat':StringVa
r(),'progresslabel':StringVar()}
        self.vars['port'].set(self.info['comport'])
        self.vars['codec'].set(self.vid.codec)
        self.vars['override'].set(self.vid.override)
        self.vars['imformat'].set(self.vid.imformat)
        self.vars['saveall'].set(self.info['saveall'])
        self.vars['progresslabel'].set('')

        #create the queue object to hold values produced by another thread for the progressbar
        self.progress_queue = Queue.Queue()
        #create the queue that controls when the capture sequence stops
        self.stop_queue =  Queue.Queue()


        #initialize the frame and the widgets
        Frame.__init__(self,parent)
        self.pack()
        self.createWidgets()

    '''
    this function creates all the widgets and GUI controls in the application
    '''
    def createWidgets(self):

        #create the file menu
        self.menubar = Menu(self)
        self.filemenu = Menu(self.menubar,tearoff=0)
        self.filemenu.add_command(label='Load data',command=self.loadTimestamps)
        self.filemenu.add_command(label='Save
data',command=lambda:self.saveTimestamps(mode='raw'))
        self.filemenu.add_command(label='Save converted
data',command=lambda:self.saveTimestamps(mode='converted'))
        self.filemenu.add_separator()
        self.filemenu.add_command(label='Exit',command=self.leaveApplication)

        #create the settings menu
        self.settingsmenu = Menu(self.menubar,tearoff=0)

        #create the capture settings menu
        self.capturemenu = Menu(self.settingsmenu,tearoff=0)
        self.capturemenu.add_command(label='Convert
timestamps',command=self.convertTimestampsDialog)
        self.capturemenu.add_checkbutton(label='Save all
timestamps?',variable=self.vars['saveall'],onvalue=True,offvalue=False,command=lambda:
self.setSaveall(self.vars['saveall'].get()))
```

```
        self.settingsmenu.add_cascade(label='Capture Settings',menu=self.capturemenu)

        #create the image settings menu
        self.imagemenu = Menu(self.settingsmenu,tearoff=0)
        self.imagemenu.add_checkbutton(label='Override previous
timestamp?',variable=self.vars['override'],onvalue=True,offvalue=False,command=lambda:
self.vid.setOverride(self.vars['override'].get()))
        self.imagemenu.add_command(label='Delimiter',command=self.setDelimiter)
        self.imformatmenu = Menu(self.imagemenu,tearoff=0)
        self.showImformats()
        self.imagemenu.add_cascade(label='Formats',menu=self.imformatmenu)
        self.settingsmenu.add_cascade(label='Image Settings',menu=self.imagemenu)

        #create the video settings menu
        self.videomenu = Menu(self.settingsmenu,tearoff=0)
        self.videomenu.add_command(label='Framerate',command=self.setFPS)
        self.codecmenu = Menu(self.videomenu,tearoff=0)
        self.showCodecs()
        self.videomenu.add_cascade(label='Codecs',menu=self.codecmenu)
        self.settingsmenu.add_cascade(label='Video Settings',menu=self.videomenu)

        #create the serial port menu
        self.portmenu = Menu(self.settingsmenu,tearoff=0)
        self.findCOMPorts()
        self.settingsmenu.add_cascade(label='Serial Ports',menu=self.portmenu)

        #create the help menu
        self.helpmenu = Menu(self.menubar,tearoff=0)
        self.helpmenu.add_command(label='Lookup satellite',command=lambda:
webbrowser.open_new_tab('https://www.calsky.com/cs.cgi/Satellites?obs=94196120633660'))
        self.helpmenu.add_command(label='Reference',command=lambda:
webbrowser.open_new_tab(os.path.join(os.path.relpath('reference'),'TSCapture_reference.html')))

        #add the menubar to the frame to display it
        self.menubar.add_cascade(label='File',menu=self.filemenu)
        self.menubar.add_cascade(label='Settings',menu=self.settingsmenu)
        self.menubar.add_cascade(label='Help',menu=self.helpmenu)
        self.root.config(menu=self.menubar)

        #create a message box and scrollbar
        self.scrollbar = Scrollbar(self)
        self.scrollbar.grid(row=0,column=4,sticky=N+S)
        self.status = Text(self)
        self.status.config(height=20,width=45,yscrollcommand=self.scrollbar.set)
        self.status.insert(END,'Application has started.\n')
        self.status.grid(row=0,column=0,columnspan=4)
        self.scrollbar.config(command=self.status.yview)

        #create the progressbar and label that runs when writing images
        self.progresslabel = Label(self,textvariable=self.vars['progresslabel'])
        self.progress = Progressbar(self,mode='determinate')
```

```
        self.progress.grid(row=1,column=0,columnspan=3,sticky=N+S+E+W)
        self.progresslabel.grid(row=1,column=3,sticky=N+S+E+W)

        #create the action buttons
        self.capture_start = Button(self,text="Start capture",command=self.start_capture)
        self.capture_start.grid(row=2,column=0,sticky=N+S+W+E)
        self.capture_stop = Button(self,text="Stop capture",command=self.stop_capture)
        self.capture_stop.grid(row=2,column=1,sticky=N+S+W+E)
        self.apply = Button(self,text='Overlay',command=self.applyTimestamps)
        self.apply.grid(row=2,column=2,sticky=N+S+W+E)
        self.video = Button(self,text='Make Video',command=self.writeImages2Video)
        self.video.grid(row=2,column=3,columnspan=2,sticky=N+S+W+E)

    '''
    this function reads values from a queue and updates the progressbar every 50ms
    '''
    def update_progress(self,length):
        try:
            while True:
                status = self.progress_queue.get_nowait()
                #check if the task is finished. if so, stop updating
                if status == None:
                        self.progress['value'] = 0
                        self.vars['progresslabel'].set('')
                        self.progress.update_idletasks()
                        return
                self.progress['value'] = status
                self.vars['progresslabel'].set('%d/%d'%(status,length))
                self.progress.update_idletasks()
        except Queue.Empty:
            pass
        #after 50ms, recursively execute the function again
        self.after(50,lambda: self.update_progress(length))

    '''
    update only the label. this is used when capturing timestamps from the orca
    '''
    def update_label(self):
        try:
            status = self.progress_queue.get_nowait()
            #check if the task is finished. if so, stop updating
            if status == None:
                self.vars['progresslabel'].set('')
                #reset the progressbar
                self.progress.stop()
                self.progress.configure(mode='determinate')
                self.progress['value'] = 0
                return
            self.vars['progresslabel'].set('%d'%(status))
        except Queue.Empty:
            pass
```

```
        self.after(50,self.update_label)

    '''
    this function adds a phrase to a new line in the textbox
    '''
    def insertNewline(self,phrase):
        if phrase == None:
            return
        else:
            self.status.insert(END,phrase)
            self.status.insert(END,'\n')
            self.status.see(END)

    '''
    function used to exit the application
    '''
    def leaveApplication(self):
        self.quit()
        self.root.destroy()

    '''
    function used to set a value into the dictionary at the index
    '''
    def toggle(self,index,value):
        self.info[index]=value

    '''
    function populates the menu list with available image extensions to use when the program is looking
for image filenames
    '''
    def showImformats(self):
        for format in self.vid.imformats:

    self.imformatmenu.add_checkbutton(label=format,variable=self.vars['imformat'],onvalue=format,off
value=None,command=lambda: self.vid.setImFormat(self.vars['imformat'].get()))

    '''
    function populates menu with available codecs to use when creating the video
    '''
    def showCodecs(self):
        for codec in self.vid.getAvailableCodecs():

    self.codecmenu.add_checkbutton(label=codec.upper(),variable=self.vars['codec'],onvalue=codec,offv
alue=None,command=lambda: self.vid.setCodec(self.vars['codec'].get()))

    '''
    function opens a dialog box to ask for path information. the kind of box depends on the mode.
    if no selection is made, then the function returns a None for the path information
    '''
    def getPath(self,mode = 'savetext',startdir = None):
        while True:
```

```
            if mode == 'savetext':
                path =
tkFileDialog.asksaveasfilename(parent=self.root,initialdir=startdir,defaultextension='.txt', title="Save text
file as")
            elif mode == 'savevid':
                path =
tkFileDialog.asksaveasfilename(parent=self.root,initialdir=startdir,defaultextension='.avi', title="Save
video as")
            elif mode == 'open':
                path = tkFileDialog.askopenfilename(parent=self.root,initialdir=startdir,title="Select data
file to load")
            elif mode == 'opens':
                path = tkFileDialog.askopenfilenames(parent=self.root,initialdir=startdir,title="Select
images")
            elif mode == 'dir':
                path = tkFileDialog.askdirectory(parent=self.root,title="Select directory where the
images are located or create a new directory to save the timestamped images.")

            #check if the path is empty or not
            if len(path) == 0:
                if tkMessageBox.askretrycancel(title='No selection', message='No selection was made.
Retry?'):
                        continue
                else:
                        return None
            else:
                return os.path.abspath(path)

    '''
    function finds available serial ports depending on the computer platform and adds them to the menu
list
    '''
    def findCOMPorts(self):
        ports = []
        #windows
        if sys.platform.startswith('win'):
            ports = ['COM'+str(i+1) for i in range(20)]
        #mac
        elif sys.platform.startswith('darwin'):
            ports = glob.glob('/dev/tty.*')
        #linux or computer running cygwin
        elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
            # this is to exclude your current terminal "/dev/tty"
            ports = glob.glob('/dev/tty[A-Za-z]*')
        else:
            raise EnvironmentError('Cannot find serial ports. Unsupported platform.')

        #now test which ports are available, add to menu if it is
        for port in ports:
            try:
                s = serial.Serial(port)
```

```
                s.close()
            except (OSError, serial.SerialException):
                continue
            else:

    self.portmenu.add_checkbutton(label=port,variable=self.vars['port'],onvalue=port,offvalue=None,co
mmand=lambda: self.toggle('comport',self.vars['port'].get()))

    '''
    Dialog box wrapper for set functions that opens up a small window to ask for a value. returns an error
dialog if the input is invalid.
    '''
    def askValue(*args,**kwargs):
        def returnDialog(func):
            def openDialog(self):
                query_dialog = Toplevel(self)
                query_dialog.minsize(210,25)
                query_dialog.title(kwargs['title'])
                label = Label(query_dialog, text=kwargs['label'])
                label.pack(side='left')
                entry = Entry(query_dialog,width=10)
                entry.insert(0,str(self.info[kwargs['initvalue']]))
                entry.pack(side='left')

                def closeDialog():
                        try:func(self,entry.get())
                        except ValueError:
                                tkMessageBox.showerror(title='Invalid Value', message='Invalid input.
Try again.')
                                query_dialog.deiconify()
                        else: query_dialog.destroy()

                accept = Button(query_dialog,text="OK",command=closeDialog)
                accept.pack(side='right')

            return openDialog
        return returnDialog

    '''
    asks the user for the fps to use when building the movie
    '''
    @askValue(title='Input FPS',label='Indicate FPS',initvalue='fps')
    def setFPS(self,value):
        self.info['fps'] = float(value)

    '''
    asks the user for the delimiter to use when looking for the image filenames
    '''
    @askValue(title='Input Delimiter',label='Input filename delimiter',initvalue='delimiter')
    def setDelimiter(self,value):
        self.info['delimiter'] = str(value)
```

```
            self.vid.setDelimiter(str(value))

        '''
        set the state on whether the application saves all timestamps it has
        '''
        def setSaveall(self,state):
            #coerce to boolean value since boolean is actually a subclass of integer in python
            #and BoolVar returns an integer and not actually a boolean
            self.info['saveall'] = bool(state)


        '''
        decorator function that resets the error checker in the TFNoneWrapper class after the wrapped
function finishes executing
        so that another independent function is not affected by the previous function call
        '''
        def resetErrorchecker(func,*args,**kwargs):
            def reset(self,*args,**kwargs):
                func(self,*args,**kwargs)
                self.vid.prevfunc_passed = True
            return reset

        '''
        command sequence used to capture timestamps from the ORCA
        '''
        def start_capture(self):
            #change the mode of the progressbar
            self.progress.configure(mode='indeterminate')
            #open up the serial port
            self.insertNewline(self.vid.open(self.info['comport']))
            #disable the buttons to avoid opening up any new threads or putting in any more tasks into the
tkinter main thread
            self.capture_start['state'] = 'disabled'
            self.apply['state'] = 'disabled'
            self.video['state'] = 'disabled'
            #open the capture loop in a separate process to avoid blocking the tkinter main thread
            pool = ThreadPool(processes=1)

        pool.apply_async(self.vid.capture,args=(self.progress_queue,self.stop_queue),callback=self.finish_ca
pture)
            #start the progressbar and label
            self.progress.start()
            self.update_label()

        '''
        function to stop the capture loop
        '''
        def stop_capture(self):
            self.stop_queue.put(True)

        '''
```

```
callback function that finishes up the capture sequence
'''
@resetErrorchecker
def finish_capture(self,captured):
    #print result
    if isinstance(captured,int):
        self.insertNewline('%d timestamps have been captured.'%(captured))
    else:
        self.insertNewline(captured)

    #stop the update_label method if a previous function did not pass
    if self.vid.prevfunc_passed == False:
        self.progress_queue.put(None)

    #re-enable the buttons
    self.capture_start['state'] = 'enabled'
    self.apply['state'] = 'enabled'
    self.video['state'] = 'enabled'

    #finish by parsing the timestamps and closing the serial port
    number_parsed = self.vid.parseTimestamps()
    if isinstance(number_parsed,int):
        self.insertNewline('%d timestamps have been parsed.'%(number_parsed))

    self.insertNewline(self.vid.close(self.info['comport']))

'''
command sequence used to overlay timestamps to a series of images
'''
@resetErrorchecker
def applyTimestamps(self):
    if len(self.vid.timestamps) <= 0:
        self.insertNewline('No timestamps to overlay.')
    else:
        self.insertNewline(self.vid.grabFromDir(self.getPath(mode='dir'),self.vid.imformat))
        self.info['imagesgrabbed'] = len(self.vid.imagelist)
        self.insertNewline('%d images have been found.'%(self.info['imagesgrabbed']))

        #if the number of images grabbed is not zero proceed
        if self.info['imagesgrabbed'] > 0:
            self.progress.configure(maximum=self.info['imagesgrabbed'])
            newdir = self.getPath(mode='dir')

            #define temporary callback function to print number of images successfully written
            def print_result(written):
                if isinstance(written,int):
                    self.insertNewline('%d images been timstamped.'%(written))
                else:
                    self.insertNewline(written)
```

```
                #perform the image writing process in another process so as to avoid blocking the tkinter
main thread
                pool = ThreadPool(processes=1)

    pool.apply_async(self.vid.overlayTimestamps,args=(newdir,self.progress_queue),callback=print_resu
lt)
                #start the progress bar update
                self.update_progress(self.info['imagesgrabbed'])
                #set the starting directory to specified save directory
                self.info['tstiffdir'] = newdir

    '''
    command sequence used to write a series of images to a video file
    '''
    @resetErrorchecker
    def writeImages2Video(self):

    self.insertNewline(self.vid.grabFromDir(self.getPath(mode='dir',startdir=self.info['tstiffdir']),self.vid.i
mformat))
        imagesgrabbed = len(self.vid.imagelist)
        self.info['imagesgrabbed'] = imagesgrabbed
        self.insertNewline('%d images have been found.'%(self.info['imagesgrabbed']))
        if imagesgrabbed > 0:
            self.progress.configure(maximum=imagesgrabbed)
            newdir = self.getPath(mode='savevid',startdir=self.info['tstiffdir'])
            #define callback function to print result to screen
            def print_result(written):
                if isinstance(written,int):
                        self.insertNewline('%d images have been compiled into a video.'%(written))
                else:
                        self.insertNewline(written)

            #perform image writing to video procedure in another process to avoid blocking the tkinter
main thread
            pool = ThreadPool(processes=1)

    pool.apply_async(self.vid.writeVideo,args=(newdir,self.progress_queue),kwds=dict(fps=self.info['fps
']),callback=print_result)
            #start the progress bar update
            self.update_progress(imagesgrabbed)

    '''
    command sequence used to save the currently captured timestamps to a text file
    '''
    @resetErrorchecker
    def saveTimestamps(self,mode='raw'):
        if len(self.vid.timestamps) == 0:
            self.insertNewline('No timestamps to save.')
            return

        savepath = self.getPath(mode='savetext',startdir=self.info['tstiffdir'])
```

```
            if mode == 'raw':
                if self.info['saveall'] or (self.info['imagesgrabbed'] > len(self.vid.timestamps)):
                    self.insertNewline(self.vid.save(savepath,0,mode))
                else:
                    self.insertNewline(self.vid.save(savepath,self.info['imagesgrabbed'],mode))
            else:
                if self.info['saveall'] or (self.info['imagesgrabbed'] > len(self.vid.converted_timestamps)):
                    self.insertNewline(self.vid.save(savepath,0,mode))
                else:
                    self.insertNewline(self.vid.save(savepath,self.info['imagesgrabbed'],mode))


        '''
        command sequence used to load a data file containing timestamps into the program
        '''
        @resetErrorchecker
        def loadTimestamps(self):
            loadpath = self.getPath(mode='open')
            self.insertNewline(self.vid.load(loadpath))
            if loadpath != None:
                self.insertNewline('%d timestamps found.'%(len(self.vid.timestamps)))

        @resetErrorchecker
        def convertTimestampsDialog(self):
            dialog = Toplevel()
            dialog.title('Conversion format')
            #create message
            message = Label(dialog,text='Select a conversion mode. Only the indicated \nunits will be
included in the data.')
            message.grid(column=0,row=0,columnspan=2)
            #create radiobuttons
            radio_tracker = StringVar()
            radio_tracker.set('second')
            second_button = Radiobutton(dialog,text='Convert to
seconds',variable=radio_tracker,value='second')
            second_button.grid(column=0,row=1,columnspan=2)
            minute_button = Radiobutton(dialog,text='Convert to
minutes',variable=radio_tracker,value='minute')
            minute_button.grid(column=0,row=2,columnspan=2)
            hour_button = Radiobutton(dialog,text='Convert to hours',variable=radio_tracker,value='hour')
            hour_button.grid(column=0,row=3,columnspan=2)
            #create ok and cancel buttons
            def do_conversion():
                result = self.vid.convertTimestamps(unit=radio_tracker.get())
                if isinstance(result,int):
                    self.insertNewline('%d timestamps have been converted.'%(result))
                    dialog.destroy()
                else:
                    tkMessageBox.showerror(title='Error', message=result)
                    dialog.deiconify()
```

```
                #manually reset the error checker since the wrapper function cannot do so due to the fact
that
                #the Toplevel window does not block so the wrapper function will reset the error checker
as soon as the
                #dialog window is created
                self.vid.prevfunc_passed = True

        accept = Button(dialog,text="OK",command=do_conversion)
        accept.grid(column=0,row=4)
        cancel = Button(dialog,text='Cancel',command=dialog.destroy)
        cancel.grid(column=1,row=4)
```

# Appendix E

# SatSearch.exe Program

# Appendix E

# SatSearch.exe Program

## E.1 Description

A preliminary observational planning Python program, called SatSearch.exe is described in this section. The SatSearch.exe program is a Python GUI written to search the satellite database maintained by the U.S. Strategic Command Joint Space Operations Center (space-track.org). Queries are currently based on object type and orbital angular velocity, but other search criteria can be easily added if required. The query data is saved to an Excel spreadsheet, while the TLE data for each object is saved as its own text file. The program then consults calsky.com to retrieve the visual magnitude estimates for the RSO list. The daily predictions for the 100 brightest satellites are also retrieved from heavens-above.com. The Space Track and 100 brightest satellites datasets are then cross-referenced to identify favorable RSO targets for a given night's observation. Further functionality will be added in the future to obtain estimates of the satellites' ephemeris data.

## E.2 How to Use

1. Click on the SatSearch.exe file in the dist folder to begin the program. Once the GUI is started up, enter in your login and password to space-track.org in order to access their database.
2. Once logged-in, the search criteria can be selected. The only current criteria available are the satellite databases, the object type, and the angular speed. The program automatically searches for the most recent satellite catalog data, but older catalog data can be queried by unchecking the 'Most current record' checkbox. Press the 'Search for sat' button to search.
3. The satellite TLE data will be saved as an Excel file, with one column listing the satellite NORAD id number, satellite name, and its corresponding angular speed. After that come the brightness data, rise, culmination, and set time estimates obtained from calsky.com.
4. The data returned from the space-track.org query can be cross-referenced with the satellite database at heavens-above.com. If this choice is selected, then the program will create a second spreadsheet in the same Excel file to save the heavens-above.com data and a third sheet to contain the cross-referenced data. You can also indicate the observing location to use when accessing heavens-above.com by indicating the longitude, latitude, altitude, city, and time zone.
5. To download the latest satellites TLEs that meet the search criteria as specified in the SatSearch program, simply click on the Download TLEs button. The TLE search can also be configure to search TLE that are accurate at a previous date. The program will prompt for a save location for the TLEs. The TLEs will be saved as text files with the satellite name as the filename. TLEs for a single object can be downloaded by specifying the NORAD ID number (more precise search parameter) or its satellite name, then clicking on the 'Download TLE for this sat' button.

6. Note: If the satellite name is used in the SatSearch.exe, the program will query the space-track.org database to search for any satellites that have a name that includes the name specify. So if 'ISS' is specified for the query, the program will download the TLE for the 'ISS' as well as for the 'AISSAT' satellites. Also if no data is found, the program will save an empty Excel file, but no TLE file will be created.

```
'''------------------------------------
SatSearch.py
Written by Cliff Chen, PNNL

This program provides a simple GUI application to search the space-track.org database. Further work
needs to be made to optimize the code.
----------------------------------------'''
from distutils.core import setup
import py2exe

setup(windows=['SatSearch.py'],data_files=['cacert.pem'])

'''------------------------------------

SatSearch.py

This program provides a simple GUI application to search the space-track.org database
Further work needs to be made to optimize the code and add on further functionality
'''

import requests, sys, os, xlsxwriter, tkFileDialog, re, traceback, Queue, multiprocessing, itertools
from Tkinter import *
from bs4 import BeautifulSoup
from time import sleep
from multiprocessing.managers import BaseManager

#create a manager to control sharing a requests Session object between processes
#the 'original' Session object is unpicklable so using a Manager to create a proxy object is necessary
#in order to use multiprocessing
class SessionManager(BaseManager):
    pass
#register the object with the manager
SessionManager.register('Session',requests.Session)

class SearchWindow(Frame):
    def __init__(self,parent=None):
        self.root = parent
        Frame.__init__(self,parent)
        self.pack()
        self.createWidgets()
        #hide the main window until the user logs in
        self.root.withdraw()
```

```
        self.tle_dir = ''
        self.excel_dir = ''

        #start the manager
        self.mgr = SessionManager()
        self.mgr.start()

        #display the log in window
        self.loginWindow()

    def createWidgets(self):

        #create frame to organize location fields
        self.loc_frame = Frame(self,borderwidth=10)
        self.loc_frame.config(highlightbackground='red')
        self.loc_frame.grid(row=0,column=0,rowspan=11,sticky=N)

        self.position_description = Label(self.loc_frame,text='Observation site coordinates')
        self.position_description.pack(anchor='w')

        #create input boxes for location
        self.long_label = Label(self.loc_frame,text='Longitude (deg)')
        self.long_label.pack(anchor='w')
        self.long_entry = Entry(self.loc_frame,width=10)
        self.long_entry.pack(anchor='w')
        self.long_entry.insert(0,'-119.1006')

        self.lat_label = Label(self.loc_frame,text='Latitude (deg)')
        self.lat_label.pack(anchor='w')
        self.lat_entry = Entry(self.loc_frame,width=10)
        self.lat_entry.pack(anchor='w')
        self.lat_entry.insert(0,'46.2396')

        self.alt_label = Label(self.loc_frame,text='Altitude (m)')
        self.alt_label.pack(anchor='w')
        self.alt_entry = Entry(self.loc_frame,width=10)
        self.alt_entry.pack(anchor='w')
        self.alt_entry.insert(0,'118')

        self.city_label = Label(self.loc_frame,text='City')
        self.city_label.pack(anchor='w')
        self.city_entry = Entry(self.loc_frame)
        self.city_entry.pack(anchor='w')
        self.city_entry.insert(0,'Pasco')

        self.timezone_label = Label(self.loc_frame,text='Timezone')
        self.timezone_label.pack(anchor='w')
        self.timezone_entry = Entry(self.loc_frame,width=10)
        self.timezone_entry.pack(anchor='w')
        self.timezone_entry.insert(0,'PST')
```

```
        #create space-track frame to organize fields
        self.st_frame = Frame(self,borderwidth=10)
        self.st_frame.grid(row=0,column=1,rowspan=15,sticky=N)

        self.login_description = Label(self.st_frame,text='Space-track.org search criteria')
        self.login_description.pack(anchor='w')

        #create controller list
        self.controller_label = Label(self.st_frame,text='Database')
        self.controller_label.pack(anchor='w')
        self.controller_list = Listbox(self.st_frame,selectmode='single',height=2,exportselection=0)
        self.controller_list.insert(1,'basicspacedata')
        self.controller_list.insert(2,'expandedspacedata')
        self.controller_list.pack(anchor='w')
        self.controller_list.select_set(0)

        #create object type list
        self.object_label = Label(self.st_frame,text='Object type')
        self.object_label.pack(anchor='w')
        self.object_list = Listbox(self.st_frame,selectmode='single',height=3,exportselection=0)
        self.object_list.insert(1,'PAYLOAD')
        self.object_list.insert(2,'ROCKET_BODY')
        self.object_list.insert(3,'DEBRIS')
        self.object_list.pack(anchor='w')
        self.object_list.select_set(0)

        #create angular speed range entry boxes
        self.maxspeed_label = Label(self.st_frame,text='Max Arcsec/s')
        self.maxspeed_label.pack(anchor='w')
        self.maxspeed_box = Entry(self.st_frame,width=10)
        self.maxspeed_box.pack(anchor='w')
        self.minspeed_label = Label(self.st_frame,text='Min Arcsec/s')
        self.minspeed_label.pack(anchor='w')
        self.minspeed_box = Entry(self.st_frame,width=10)
        self.minspeed_box.pack(anchor='w')

        #create entry box for indicating how far into the past to search for TLEs
        self.days_label = Label(self.st_frame,text='Search TLEs in\nthe past x days')
        self.days_label.pack(anchor='w')
        self.days_box = Entry(self.st_frame,width=10)
        self.days_box.pack(anchor='w')
        self.days_box.insert(0,'7')

        #create the checkbox to sort by most current catalog record
        self.current_tracker = StringVar()
        self.current_checkbox = Checkbutton(self.st_frame,text='Most current
record',variable=self.current_tracker,onvalue='true',offvalue='false')
        self.current_checkbox.pack(anchor='w')
        self.current_checkbox.select()

        #create checkbox for cross-referencing with heavens-above
```

```
        self.cross_tracker = StringVar()
        self.cross_checkbox = Checkbutton(self.st_frame,text='Cross ref with heavens-
above.com',variable=self.cross_tracker,onvalue='true',offvalue='false')
        self.cross_checkbox.pack(anchor='w')
        self.cross_checkbox.select()

        #create entry box for satellite name
        self.sat_id_label = Label(self.st_frame,text='NORAD ID or\n satellite name')
        self.sat_id_label.pack(anchor='w')
        self.sat_id_box = Entry(self.st_frame,width=15)
        self.sat_id_box.pack(anchor='w')

        #create frame to organize buttons
        self.button_frame = Frame(self,borderwidth=10)
        self.button_frame.grid(row=0,column=2,sticky=N)

        #create submit buttons
        self.search = Button(self.button_frame,text='Search for Sat',command=self.startSearch)
        self.search.pack(side='left',padx=5)
        self.download = Button(self.button_frame,text='Download TLEs',command=lambda:
self.downloadTLEs('multiple'))
        self.download.pack(side='left',padx=5)
        self.download_single = Button(self.button_frame,text='Download single TLE',command=lambda:
self.downloadTLEs('single'))
        self.download_single.pack(side='left',padx=5)

        #create output box
        self.output_frame = Frame(self)
        self.output_frame.grid(row=1,column=2,rowspan=8,sticky=N)
        self.output_box_label = Label(self.output_frame,text='Status')
        self.output_box_label.pack(anchor='nw')
        self.output_box = Text(self.output_frame,height=10,width=40)
        self.output_box.pack(anchor='nw',pady=5)

    #create a login window at the start of the program
    def loginWindow(self):
        logged = False
        #create login entry boxes
        tl = Toplevel()
        f1 = Frame(tl,borderwidth=10)
        f1.pack()
        description = Label(f1,text='space-track.org login')
        description.pack(anchor='w')
        username_label = Label(f1,text='Username')
        username_label.pack(anchor='w')
        username = Entry(f1)
        username.pack(anchor='w')
        username.focus()
        password_label = Label(f1,text='Password')
        password_label.pack(anchor='w')
        password = Entry(f1,show='*')
```

```
        password.pack(anchor='w')

        #redefine the closing protocol on the pop-up window
        #so that it if the user presses the 'x' button the program
        #will completely exit instead of closing only the pop-up window
        #and leave the rest of the program running the background
        def closeWindow():
            self.root.destroy()
        tl.protocol('WM_DELETE_WINDOW',closeWindow)

        #define login process
        def login():
            self.login2SpaceTrack(username.get(),password.get())
            #show the main window
            self.root.deiconify()
            #destroy only the pop-up window
            tl.destroy()

        #function to bind login to an event
        def bind_login(event):
            login()

        #bind return key event with login process
        password.bind('<Return>',bind_login)

        f2 = Frame(tl,borderwidth=10)
        f2.pack()
        #create login/logout buttons
        login_button = Button(f2,text='Login',width=10,command=login)
        login_button.pack(side='left',anchor='w',padx=5)
        #end the application if the user chooses not to login
        cancel_button = Button(f2,text='Cancel',width=10,command=self.root.destroy)
        cancel_button.pack(side='left',anchor='w',padx=5)

    def outputResponse(self,message):
        self.output_box.delete("%d.%d" % (1, 0),END)
        self.output_box.insert('%d.%d'%(1,0),message)

    #wrapper function to check for connection or timeout problems
    def checkNetworkConnection(func,*args,**kwargs):
        def check(self,*args,**kwargs):
            try:
                return func(self,*args,**kwargs)
            except requests.exceptions.ConnectionError as e:
                self.outputResponse(e)
            except requests.exceptions.Timeout as e:
                self.outputResponse(e)
        return check

    @checkNetworkConnection
    def login2SpaceTrack(self,username,password):
```

```
            self.focus()
            #create the proxy session object using the manager
            self.session = self.mgr.Session()
            response = self.session.post('https://www.space-
track.org/ajaxauth/login',data={'identity':username,'password':password},timeout=30)
            if response.status_code == 200:
                if response.text[1:-1] == '':
                    self.outputResponse('Logged onto space-track.org')
                else:
                    self.outputResponse(response.text)
            else:
                self.outputResponse('Could not log onto space-track.org Error code =
%d'%response.status_code)

        #not currently being used
        @checkNetworkConnection
        def logout(self):
            try:
                response = self.session.get('https://www.space-track.org/ajaxauth/logout',timeout=30)
            except AttributeError:
                self.outputResponse('Not logged into space-track.org')
            else:
                if response.status_code == 200:
                    self.outputResponse(response.text)
                    self.session.close()
                    del self.session
                else:
                    self.outputResponse('Problem logging off space-track.org Error code =
%d'%response.status_code)

        #build the query url for space-track.org
        def buildURL(self,**kwargs):
            parameters = []
            controller_selection = self.controller_list.curselection()
            if len(controller_selection) == 0:
                self.outputResponse('No database was selected.')
                return None
            else:
                parameters.append(self.controller_list.get(controller_selection))

            #add parameter to indicate query
            parameters.append('query')
            #add parameter to designate class
            parameters.append('class')
            object_type = self.object = self.object_list.curselection()

            #depending on which database we are querying the program will build the url differently
            if kwargs['data_type'] == 'satcat':
                #append parameter to indicate the satellite catalog databaes
                parameters.append('satcat')
                #append parameter to indicate object type
```

```
        parameters.append('OBJECT_TYPE')
        parameters.append(self.object_list.get(object_type))

        #append parameter to indicate period
        parameters.append('PERIOD')
        try:
            min_speed = float(self.minspeed_box.get())
            max_speed = float(self.maxspeed_box.get())
        except ValueError:
            self.outputResponse('Min and max speeds are not valid.')
            return None
        #convert arcsec/s to its corresponding period
        max_period = str(2*180*3600/(60*min_speed))
        min_period = str(2*180*3600/(60*max_speed))
        parameters.append(min_period+'--'+max_period)

        #indicate if we want only the most current record
        if self.current_tracker.get() == 'true':
            parameters.append('CURRENT')
            parameters.append('Y')

        #order by descending period
        parameters.append('orderby')
        parameters.append('PERIOD%20desc')
        #remove any duplicate rows
        parameters.append('distinct')
        parameters.append('true')

    elif kwargs['data_type'] == 'multiple':
        #grab the most recent tles
        parameters.append('tle_latest')
        #show only the most recent tle
        parameters.append('ORDINAL')
        parameters.append('1')

        #indicate object type
        parameters.append('OBJECT_TYPE')
        parameters.append(self.object_list.get(object_type))

        #indicate how far into the past to look for the most recent tle
        parameters.append('EPOCH')
        try:
            days = int(self.days_box.get())
        except ValueError:
            days = 7

        parameters.append('>now-'+str(days))

        #specify the mean-motion based on the arcsec/s
        parameters.append('MEAN_MOTION')
        try:
```

```
                min_speed = float(self.minspeed_box.get())
                max_speed = float(self.maxspeed_box.get())
            except ValueError:
                self.outputResponse('Min and max speeds are not valid.')
                return None
            max_mean_motion = str(24*float(max_speed)/360)
            min_mean_motion = str(24*float(min_speed)/360)
            parameters.append(min_mean_motion+'--'+max_mean_motion)

            #remove duplicate rows
            parameters.append('distinct')
            parameters.append('true')

        else:
            parameters.append('tle_latest')
            #show only the most recent tle
            parameters.append('ORDINAL')
            parameters.append('1')

            #indicate how far into the past to look for the latest tle
            parameters.append('EPOCH')
            try:
                days = int(self.days_box.get())
            except ValueError:
                #if there is an error, just use a default number of 7 days
                days = 7

            #read in the input from entry box and determine if the person gave an id#
            #or a satellite name to search for
            parameters.append('>now-'+str(days))
            try:
                sat_id = int(self.sat_id_box.get())
            except ValueError:
                sat_name = self.sat_id_box.get()
                if len(sat_name) == 0:
                        self.outputResponse('No NORAD ID# or satellite name was given.')
                        return None
                else:
                        parameters.append('OBJECT_NAME')
                        parameters.append('~~'+sat_name)
            else:
                parameters.append('NORAD_CAT_ID')
                parameters.append(str(sat_id))

        #use the json format for easier computer processing
        parameters.append('format')
        parameters.append('json')

        return 'https://www.space-track.org/'+'/'.join(parameters)

    #return a dictionary containing the observing location information
```

```
    def buildLocation(self):
        return
{'lat':self.lat_entry.get(),'lng':self.long_entry.get(),'loc':self.city_entry.get(),'alt':self.alt_entry.get(),'tz':self.
timezone_entry.get()}

    #method to update the status text box while a search or action is being completed
    def loadingSequence(self,stop_queue):
        try:
            #use get_nowait to prevent blocking the tkinter thread
            status = stop_queue.get_nowait()
        except Queue.Empty:
            pass
        else:
            if status == 'stop':
                return
            self.outputResponse(status)

        #reschedule another queue check after a 100ms wait
        self.after(100,lambda: self.loadingSequence(stop_queue))

    #method to start a search
    def startSearch(self):
        url = self.buildURL(data_type='satcat')
        filename = tkFileDialog.asksaveasfilename(defaultextension='.xlsx',title='Save data
as',initialdir=self.excel_dir)
        if url is not None and len(filename) != 0:
            try:
                self.excel_dir = os.path.dirname(filename)
                #create communication queue between child process and main tkinter thread
                queue = multiprocessing.Queue()
                search_criteria =
{'session':self.session,'url':url,'filename':filename,'location':self.buildLocation(),'textbox_queue':queue,'cro
ss_ref':self.cross_tracker.get()}
                search = multiprocessing.Process(target=checkDatabases,kwargs=search_criteria)
                search.start()
                self.loadingSequence(queue)
            except AttributeError:
                self.outputResponse('Not logged into space-track.org')

    #method to start a tle download
    def downloadTLEs(self,type):
        self.focus()
        url = self.buildURL(data_type=type)
        directory = tkFileDialog.askdirectory(title='Indicate directory to save the
TLEs',initialdir=self.tle_dir)
        if url is not None and len(directory) != 0:
            try:
                self.tle_dir = directory
                queue = multiprocessing.Queue()
                queue.put('...')
```

```
                    dl_process =
multiprocessing.Process(target=grabSpaceTrackTLE,args=(self.session,url,directory,queue))
                    dl_process.start()
                    self.loadingSequence(queue)
                except AttributeError:
                    self.outputResponse('Not logged into space-track.org')

#slave class that continually reads a queue and performs the task received from the queue
#results are returned in another queue and the slave stops when it receives a None
class Slave(multiprocessing.Process):
    def __init__(self,task_queue,result_queue):
        multiprocessing.Process.__init__(self)
        self.task_queue = task_queue
        self.result_queue = result_queue

    def run(self):
        while True:
            #grab a task. this will block until something is available in the queue
            task = self.task_queue.get()
            #check if there are no more tasks to receive
            if task == None:
                self.task_queue.task_done()
                return
            result = task[0](task[1])
            self.task_queue.task_done()
            self.result_queue.put(result)

        return

#these functions are defined at the top level of the module, outside of any classes
#since the multiprocessing module has trouble pickling class methods
def checkDatabases(**kwargs):
    #create task queue and results queue
    #results are returned as a tuple in the format:
    #(task-name,task-data,task-error-info)
    tasks = multiprocessing.JoinableQueue()
    results = multiprocessing.Queue()

    #create search slaves
    num_slaves = [Slave(tasks,results) for i in xrange(multiprocessing.cpu_count())]

    #start the slaves
    for s in num_slaves:
        s.start()

    #build output message
    textbox_queue = kwargs['textbox_queue']
    message = 'Searching:\n'

    #enqueue the searches to perform
    tasks.put((wrapped_getSpaceTrackData,{'session':kwargs['session'],'url':kwargs['url']}))
```

```
message = message + 'space-track.org\n'

if kwargs['cross_ref'] == 'true':
    tasks.put((wrapped_getDataFromHeavensAbove,kwargs['location']))
    message = message + 'heavens-above.com\n'

textbox_queue.put(message)

#wait for all tasks to be completed before continuing

tasks.join()

#grab the search results from the results queue checking if there were any errors
#load any errors into the error dictionary
result_dict = {}
error_dict = {}
num_results = results.qsize()
for i in xrange(num_results):
    result = results.get()
    if result[2] is None:
        result_dict[result[0]] = result[1]
    else:
        error_dict[result[0]] = result[1]+str(result[2])

#check if the primary space-track search went through with no connection errors
#does not necessarily mean that data was received
if 'space-track' not in error_dict:
    textbox_queue.put('Getting brightness estimates from calsky.com')

    #extract the list of satellite ids
    id_list = []
    for sat in result_dict['space-track']:
        id_list.append(str(sat['NORAD_CAT_ID']))

    #for each id in the list, enqueue a new search task to find the satellite brightness
    for id in id_list:
        tasks.put((wrapped_getMagnitudeEstimates,id))

    #wait for the tasks to be completed before continuing
    tasks.join()

    #get the brightness estimate results
    num_results = results.qsize()
    mag_list = {}
    for k in xrange(num_results):
        result = results.get()
        if result[2] is None:
            mag_list[result[0]] = result[1]
        else:
            error_dict[result[0]] = result[1]+str(result[2])
```

```
            #open up the excel file for writing
            book = xlsxwriter.Workbook(kwargs['filename'])

            #write the data to the excel file
            textbox_queue.put('Writing data to spreadsheet.')
            for key in result_dict:
                if key == 'space-track':
                    writeSpaceTrackData2Excel(book,result_dict['space-track'],mag_list)
                else:
                    writeHeavensAboveData2Excel(book,result_dict[key])

            #cross reference if there were no errors
            if kwargs['cross_ref'] == 'true':
                if 'heavens-above' not in error_dict:
                    compareSatellites(book,result_dict['space-track'],result_dict['heavens-above'])

            #close the excel file
            book.close()

        #enqueue a 'kill' indicator for each slave to end them
        for i in xrange(len(num_slaves)):
            tasks.put(None)

        #output the error message if there were any errors
        if len(error_dict) != 0 :
            err_message = 'Errors occured in the following searches:\n'
            for i in error_dict:
                err_message = err_message+'%s:%s\n'%(i,error_dict[i])
            textbox_queue.put(err_message)
        else:
            textbox_queue.put('Search finished with no errors.')

        textbox_queue.put('stop')

#cross reference the satellites obtained from space-track and heavens-above
def compareSatellites(book,space_track,heavens_above):
    worksheet = book.add_worksheet('st ha cross ref')
    createCrossRefHeaders(worksheet)
    k=1
    for n,row in enumerate(heavens_above):
        #match satellite names using regular expressions and produce a list of yes/no matches
        matches = ((re.match(row[0]+'\b',sat['SATNAME'],re.IGNORECASE),sat['PERIOD']) for sat in
space_track)

        #filter out the 'no' matches and replace with only the matched sat's period
        matches = (match[1] for match in matches if match[0] is not None)

        #for each match add the data to the spreadsheet
        for match in matches:
            for m,value in enumerate(row):
                worksheet.write(k,m,value)
```

```
                worksheet.write('L%d'%(k),2*180*3600/(60*float(match)))
                k+=1

#create the headers for cross-referencing
def createCrossRefHeaders(worksheet):
    worksheet.write('A1','SAT NAME')
    worksheet.write('B1','MAG')
    worksheet.write('C1','START TIME')
    worksheet.write('D1','START ALT')
    worksheet.write('E1','START AZI')
    worksheet.write('F1','HIGHEST TIME')
    worksheet.write('G1','HIGHEST ALT')
    worksheet.write('H1','HIGHEST AZI')
    worksheet.write('I1','END TIME')
    worksheet.write('J1','END ALT')
    worksheet.write('K1','END AZI')
    worksheet.write('L1','ARCSEC/S')

#a rehash of the wrapper function above
def checkConnection(name,func,*args,**kwargs):
    def check(*args,**kwargs):
        try:
            return func(*args,**kwargs)
        except requests.exceptions.ConnectionError as e:
            return (name,'Connection error.',e)
        except requests.exceptions.Timeout as e:
            return (name,'Page request timed out.',e)
    return check

#get the data from space-track.org. the function does not use true **kwargs arguments
def getSpaceTrackData(kwargs):
    session = kwargs['session']
    response = session.get(kwargs['url'])
    if response.status_code == 401:
        return ('space-track','Unauthorized access.',401)
    else:
        return ('space-track',response.json(),None)

#due to the fact that multiprocessing can only pickle top level functions, the decorator syntax no longer
works since it can't
#find the wrapped function so we need to wrap it manually
def wrapped_getSpaceTrackData(kwargs):
    wrapped = checkConnection('space-track',getSpaceTrackData)
    return wrapped(kwargs)

#write the space track data to an excel file
def writeSpaceTrackData2Excel(book,data,mag_list):
    worksheet = book.add_worksheet('space-track catalog')
    createSpaceTrackHeaders(worksheet)
    createCalSkyHeaders(worksheet)
    for n,sat in enumerate(data):
```

```
        id = sat['NORAD_CAT_ID']
        worksheet.write('A%d'%(n+2),id)
        worksheet.write('B%d'%(n+2),sat['SATNAME'])
        worksheet.write('C%d'%(n+2),6*3600/(float(sat['PERIOD'])))

        #check if the dictionary has the id as a key
        #the id may not be present if there were connection problems
        #when accessing calsky.com
        try:
            calsky_info = mag_list[id]
        except KeyError:
            pass
        else:
            for k,list in enumerate(calsky_info):
                for m,item in enumerate(list):
                        worksheet.write(n+1,3+4*m+k,item)

def wrapped_getMagnitudeEstimates(id):
    wrapped = checkConnection(id,getCalSkyMagEstimates)
    return wrapped(id)

#this function accepts a satellite's NORAD id and accesses calsky.com to get the listed brightness data
#the function uses BeautifulSoup as the html scraper and regular expressions to parse out the data from
the
#text. the data is returned as a tuple containing the 4 lists with the information
def getCalSkyMagEstimates(id):
    keys = {'object':'Satellite','number':'3','sat':id}
    response = requests.get('http://www.calsky.com/csrender.cgi?',params=keys)
    soup = BeautifulSoup(response.text,'lxml')

    #return a list of all h3 tags that contain the text 'Topocentric:\n'
    all_h3 = soup('h3',text='Topocentric:\n')

    #create the data lists
    mag_list = []
    azi_list = []
    alt_list = []
    times_list = []

    #this for loop will only run at most once
    for h3 in all_h3:
        #get the next tag that is at the same level in the tag tree
        pre = h3.next_sibling

        #search the text within the tag for any strings that contain the word 'Magnitude:'
        mag_text = pre(text=re.compile('Magnitude:'))
        for item in itertools.islice(mag_text,4):
            #for each line grab the data between the words 'Magnitude:' and 'mag'
            m = re.search('(?<=Magnitude: ).+(?=mag)',item)
            if m is not None:
                mag_list.append(float(m.group(0)))
```

```
            else:
                mag_list.append('')

        #search for strings that have 'Altazimuth:' in them
        altazi_text = pre(text=re.compile('Altazimuth:'))
        for i in itertools.islice(altazi_text,4):
            #get the string data that comes after 'Az' and 'Alt'
            n = re.search('(?<=Az    = )\s*\d+(\.\d*)\xb0\s(.{1,3})',i,re.UNICODE)
            k = re.search('(?<=Alt   = )\s*.\d+(\.\d*)\xb0',i,re.UNICODE)
            if n is not None:
                azi_list.append(n.group(0))
            else:
                azi_list.append('')
            if k is not None:
                alt_list.append(k.group(0))
            else:
                alt_list.append('')

        #search for strings that have the indicated time format
        times_text = pre(text=re.compile('\s*\d+h([0-5][0-9])m([0-5][0-9])s',re.UNICODE))
        for d in itertools.islice(times_text,4):
            t = re.search('\s*\d+h([0-5][0-9])m([0-5][0-9])s',d,re.UNICODE)
            if t is not None:
                times_list.append(t.group(0))
            else:
                times_list.append('')

    return (id,(mag_list,azi_list,alt_list,times_list),None)

def createSpaceTrackHeaders(worksheet):
    worksheet.write('A1','NORAD ID')
    worksheet.write('B1','SAT NAME')
    worksheet.write('C1','ARCSEC/S')

def createCalSkyHeaders(worksheet):
    worksheet.write('D1','TOPOCENTRIC MAG')
    worksheet.write('E1','TOPOCENTRIC AZI')
    worksheet.write('F1','TOPOCENTRIC ALT')
    worksheet.write('G1','TOPOCENTRIC RA')
    worksheet.write('H1','RISE MAG')
    worksheet.write('I1','RISE AZI')
    worksheet.write('J1','RISE ALT')
    worksheet.write('K1','RISE TIME')
    worksheet.write('L1','CULMINATE MAG')
    worksheet.write('M1','CULMINATE AZI')
    worksheet.write('N1','CULMINATE ALT')
    worksheet.write('O1','CULMINATE TIME')
    worksheet.write('P1','SET MAG')
    worksheet.write('Q1','SET AZI')
    worksheet.write('R1','SET ALT')
    worksheet.write('S1','SET TIME')
```

```
#grab the data from heavens-above, parse it through BeautifulSoup and spit out an easier to work with
data list
def getDataFromHeavensAbove(location):
    response = requests.get('http://www.heavens-above.com/AllSats.aspx',params=location)
    soup = BeautifulSoup(response.text)
    all_rows = soup.find_all('tr',class_='clickableRow')
    data = []
    for n,row in enumerate(all_rows):
        this_row = []
        this_row.append(str(row.td.contents[0]))

        for m,sibling in enumerate(row.td.find_next_siblings('td')):
            #if statement is necessary since one of the data elements is contained within a link tag
            if (m+1) == 5:
                this_row.append(sibling.a.contents[0].encode('ascii','ignore'))
            else:
                this_row.append(sibling.contents[0].encode('ascii','ignore'))

        data.append(this_row)

    return ('heavens-above',data,None)

#once again, we need to manually wrap the function
def wrapped_getDataFromHeavensAbove(location):
    wrapped = checkConnection('heavens-above',getDataFromHeavensAbove)
    return wrapped(location)

#write data from heavens-above to an excel file
def writeHeavensAboveData2Excel(book,data):
    worksheet = book.add_worksheet('heavens-above catalog')
    createHeavensAboveHeaders(worksheet)
    for r,row in enumerate(data):
        for c,item in enumerate(row):
            worksheet.write(r+1,c,item)

#create the headers for the heavens-above data
def createHeavensAboveHeaders(worksheet):
    worksheet.write('A1','SAT NAME')
    worksheet.write('B1','MAG')
    worksheet.write('C1','START TIME')
    worksheet.write('D1','START ALT')
    worksheet.write('E1','START AZI')
    worksheet.write('F1','HIGHEST TIME')
    worksheet.write('G1','HIGHEST ALT')
    worksheet.write('H1','HIGHEST AZI')
    worksheet.write('I1','END TIME')
    worksheet.write('J1','END ALT')
    worksheet.write('K1','END AZI')

#grab the tles from space-track
```

```
def grabSpaceTrackTLE(session,url,directory,stop_queue):
    stop_queue.put('Gathering TLEs...')
    response = session.get(url)

    if response.status_code == 401:
        stop_queue.put('Unauthorized access.')
    else:
        for sat in response.json():
            stop_queue.put('Writing TLE for:'+sat['OBJECT_NAME']+'...')
            path = os.path.join(directory,re.sub('[/]',' ',sat['OBJECT_NAME'])+'.txt')
            index = 1
            newpath = path
            #while loop to keep checking if a filename exists. if it does, append a number to the
            #end of the filename and recheck until the filename does not exist
            while os.path.isfile(newpath):
                newpath = path[:path.index('.txt')]+'(%d)'%index+'.txt'
                index += 1

            with open(newpath,'w') as f:
                f.write(sat['OBJECT_NAME']+'\n')
                f.write(sat['TLE_LINE1']+'\n')
                f.write(sat['TLE_LINE2'])
                f.close()

        stop_queue.put('Finished downloading TLEs.')

    stop_queue.put('stop')

if __name__ == '__main__':
    #point to the file containing list of trusted ssl certificates since the py2exe module
    #does not bundle the ca file with the rest of the program when building the executable
    os.environ['REQUESTS_CA_BUNDLE'] = os.path.join(os.getcwd(),'cacert.pem')
    root = Tk()
    root.title('Satellite Search')
    search = SearchWindow(root)
    root.mainloop()
```

# Appendix F

# EMCCD Noise Characterization

# Appendix F

# EMCCD Noise Characterization

## F.1 Introduction

Readout noise is generated during the electronic process that converts pixel photoelectrons to digitized count. During this process, charge accumulated by the pixels is shifted across the detector to the output charge amplifier. Noise generated during this process increases significantly as a function of readout speed or frame rate. Readout noise is independent of photon shot noise and can still be observed under completely dark conditions. The readout noise for a camera is usually given in terms of its RMS value in units of electrons.

## F.2 Measurement Procedure

Readout noise was characterized as a function of EMCCD gain and exposure time, as described in this appendix. Sets of 20 images were captured in a dark room with a lens cap placed over the camera lens. Each image set was collected at different exposure time and gain. Fixed pattern noise was removed by image subtraction using the last two frames of the set of 20. The readout noise in each image was determined by the standard deviation of the pixel counts across the image, divided by $\sqrt{2}$ (because we are considering the noise twice). The readout noise measurements also include dark noise and clock-induced charge noise components. However, these noise components were not independently characterized, because they are generally small relative to the readout noise.

The readout noise measurement results are shown in Figure F.1. The effective readout noise (i.e., readout noise/EM gain) is less than 20 electrons at a gain of 4 and decreases with increasing gain. Two of the plots were collected using a field delay, which refers to the amount of time between subsequent exposures. A 30-ms exposure with a 100-ms field delay implies a 130-ms period between each frame. While Figure F.1 shows data at sub-electron noise levels, readout noise for gains $\geq$ 400 are actually at the 1 electron level.

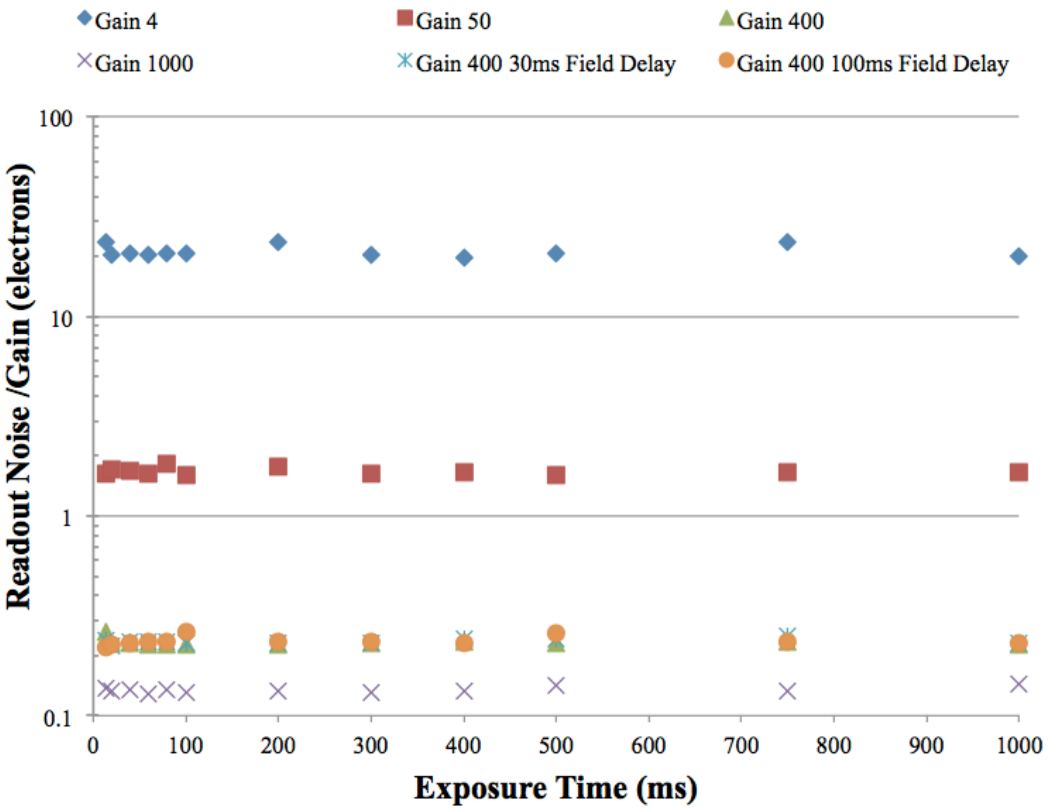



**Figure F.1**. Effective Readout for Several Exposure Times and EM Gains

www.pnnl.gov

Pacific Northwest
NATIONAL LABORATORY

*Proudly Operated by* **Battelle** *Since 1965*

U.S. DEPARTMENT OF
ENERGY

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
**1-888-375-PNNL** (7665)