Optical Navigation using Near Celestial Bodies for Spacecraft Autonomy

by

Thibaud F. Teil

B.S., Aerospace Engineering, ISAE-Supaero

Toulouse, France, 2014

M.S., Aerospace Engineering, ISAE-Supaero

Toulouse, France, 2016

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Aerospace Engineering Sciences

2020

This thesis entitled:
Optical Navigation using Near Celestial Bodies for Spacecraft Autonomy
written by Thibaud F. Teil
has been approved for the Department of Aerospace Engineering Sciences

Prof. Hanspeter Schaub

Prof. Nisar Ahmed

Dr. Islam Hussein

Dr. Daniel Kubitschek

Prof. Jay McMahon

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Teil, Thibaud F. (Ph.D., Aerospace Engineering Sciences)

Optical Navigation using Near Celestial Bodies for Spacecraft Autonomy

Thesis directed by Prof. Hanspeter Schaub

Spacecraft have been exploring the celestial bodies of our solar system for more than half a century. Despite the distances they have crossed, spacecraft remain in great part tethered to Earth for navigation and control purposes. As humankind continues to explore the solar system, the need for autonomous operations with minimal ground contact grows. The communication delays can be much larger than the required spacecraft response time such as during an orbit insertion maneuver or during entry decent and landing phase. Additionally, enhancing autonomy in navigation will continue to lighten the load on mission operations. When possible, performing navigation functions onboard circumvents light-time and human related delays, reduces the load on ground stations, and enables certain mission designs that are intractable without on-board decision-making. Optical Navigation in astrodynamics refers to the use of images taken by an onboard camera in order to determine the spacecraft's position. The images contain solar system bodies and therefore provide relative position, velocity and attitude information. The work in this dissertation focuses on the guidance, navigation, and control algorithms that allow for probes to travel the solar system. It revolves primarily around the autonomous navigation capabilities that optical navigation provides. By directly using the local environment, optical measurements can aid in spacecraft orientation and orbit determination and guidance, as well as science. The work in this thesis presents advances in heading determination filters, robust orbit determination methods using space imagery, fault detection cases, and machine learning for astrodynamics. The research is enabled by the initial development of an open-source software package, and presents research interests on its own. Applications of this thesis include mission design, Monte-Carlo analysis, and spacecraft autonomy.

À Papa

Navigateur depuis ta naissance,

même si tu confondais parfois ta droite de ta gauche.

Nous lirons ces pages en pensant à toi.

Acknowledgements

I would first like to thank my PhD advisor, Dr. Hanspeter Schaub. I joined the Autonomous Vehicle Systems Lab in 2016 and Dr. Schaub has funded me throughout my PhD working on the EMM Mars mission with LASP. He supported my research interests from the start and gave me all the freedom to explore fields of interest. The experience and knowledge that he has provided me is invaluable and I am very thankful to have worked with him. Dr. Schaub has been deeply dedicated to my success, and has shown nothing but passion for his field and students. I'd also like to thank my committee members Dr. Nisar Ahmed, Dr. Islam Hussein, Dr. Daniel Kubitschek and Dr. Jay McMahon for their time and support. They have all provided insight into my research as well as countless avenues to explore in the years to come.

I came into the AVS Lab under the wing of Patrick Kenneally, Cody Allard, John Alcorn. I am so glad to have been able to learn and grow around such a smart and compassionate group of individuals. Many thanks to them for their friendship in and outside of the lab. Alongside this crew, working with the LASP team on EMM has been incredibly enriching. Scott Piggott is a brilliant technical mind who readily gives his limited time to support individual efforts (including CSS filter fun!). Dan Kubitschek is a generous, deeply knowledgeable leader and mentor in every context. Without the emotional and technical support of these two, much of the work presented here would not have been possible. I am very thankful to have been working with this team for years, and most grateful for the bonds we have created.

A big thanks to Sam Bateman, who provided great insight in neural network design and software development. Support from Lucas Webb and other DLA students has also been very

valuable and has helped achieve all the goals I was aiming for.

A shout out to current members of the AVS Lab and CCAR, I lack the room to properly acknowledge everyone's influence. Ben Bercovici, Jordan Maxwell, Andrew Harris, Mar Cols, Chandrakanth Venigalla: you have all been there for the tough and good times. Thanks for the fond memories, and the many more to come. To all my friends in France for their love and support: famille. Halie, you have always been supportive of my efforts, and you have constantly given your time and energy. Hard to imagine what and where I would be without all your love and I am deeply grateful.

So few words could never describe the gratitude and love that I have for my whole family. Especially Mom, and Dad; Margaux and Romain; Grand-Maman, Grand-Papa, François and Claire; Christophe, Fabienne, Laurie and Morgane. What a lottery I have won just by landing on Earth (of all places) with you all to call family. What was that quote regarding standing on the shoulders of giants? Mom and Dad have literally and figuratively carried me throughout my life and are real-life giants that no acknowledgment could do justice to. Today, its hard to express the sadness of not being able to share this accomplishment with some, but then again, we have always been each others proudest achievements.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

1.1 Background and Motivation

Spacecraft have been exploring the celestial bodies of the Solar System for more than half a century. Space missions have taught humankind much about the world and has enriched fields from planetary science to medicine and philosophy. Deep space exploration tugs on the limits of human knowledge all the while propelling it. Given the wealth of natural objects in the solar system, the variety in robotic and crewed missions have only been limited by available technology and appropriated resources.

For any mission goal to be achieved, many engineering challenges must be overcome in order to bring the craft safely to its target. Expeditions have included flybys, orbit insertion, landings, and sample-returns, all in support of a broad spectrum of science goals. One of the fundamental challenges resides simply in knowing the position of the spacecraft throughout its mission (orbit determination) and controlling its trajectory and attitude according to objectives. The determination of a spacecraft's states is traditionally done on earth using standard radiometric tracking data — two-way Doppler, two-way range, and Delta-DOR — with the occasional use of onboard optical data (images taken by the spacecraft camera and sent to Earth for analysis).[17]

As spacecraft continue to explore the solar system, the need for autonomous operations with minimal Earth contact continues to grow.[195] The communication delays can be much larger than the required spacecraft response time such as during an orbit insertion or during entry decent and landing (EDL). Additionally, enhancing autonomy in navigation will continue to lighten the load

on mission operations. Figure 1.1 shows some the many orbits that the Cassini spacecraft flew in the Saturn system. If missions of this complexity are to become more frequent and more daring, they must become more autonomous.

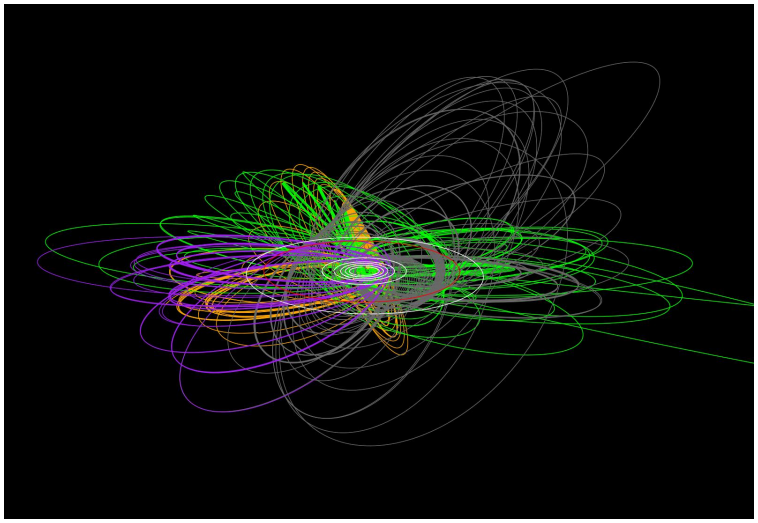


Figure 1.1: Cassini's "Ball of Yarn" Credit: NASA/JPL-Caltech

When possible, performing navigation functions onboard circumvents light-time and human related delays, reduces the load on ground stations, and enables certain mission designs that are intractable without on-board decision-making. Optical Navigation (OpNav) in astrodynamics refers to the use of images taken by an onboard camera in order to determine the spacecraft's position.[167] The images contain solar system bodies and therefore provide relative position and attitude information. Commonly, OpNav measurements are combined with radiometric data or other measurements to compute a navigation solution. Nonetheless, the images can provide all the necessary information to estimate the spacecraft states which makes OpNav a good candidate data type for autonomous navigation.

The work in this dissertation focuses on the guidance, navigation, and control (GNC) algorithms that allow for unmanned and manned probes alike to travel the solar system. It revolves primarily around the autonomous navigation capabilities that OpNav provides when applied to known celestial bodies that appear resolved on the camera (therefore relatively near the space-

craft). By directly using the local environment, OpNav can aid in attitude (spacecraft orientation) determination and guidance, orbit determination, as well as science.

1.1.1 Historical Context for Spacecraft Navigation

The 1960s was the century that saw spacecraft navigation flourish. The conjunction of the *Apollo*[100] programs, *Gemini* program (which in 1965 flew the first OpNav instrument[194]), and the advent of Kalman filtering[111] in 1960 were the technological stepping stones that eventually lead to the moon landings in 1969. Further advances during *Apollo* paved the way towards the robotic missions of today. Since then, the field of statistical orbit determination has grown and links the general theory of state estimation and practical spacecraft orbit determination in the face of uncertainties. It also opens the door to rigorous fault mitigation through statistical outlier detection.

Crewed spaceflight has continued since *Apollo*. Space Shuttle flew astronauts to Low Earth Orbit (LEO), and allowed the construction of the International Space Station (ISS). Although LEO does not require extensive navigation capabilities, it does provide testing grounds for instruments and techniques before being used in deep-space.[217] The ISS is the most expensive international project to date[124] and a display of both the challenges of space exploration and the unwavering curiosity that drives mankind to explore.

Aside from crewed missions, robotics missions have been on the forefront of space exploration. *Voyager* 1 and 2 have now left the solar system[117] and used a conjunction of radiometric data from Deep Space Network (DSN) stations and OpNav. OpNav provides a direct measurement of the relative position of the spacecraft with the objects of interest, and therefore is not affected by uncertainties in ephemerides. These are some of the events that have led to the wide use of OpNav for interplanetary navigation which has now been utilized in a number of past missions like *Cassini*[19] and *Rosetta*,[31] as well as current missions such as *New Horizons*,[87] *Osiris-REx*,[125] and *Hayabusa* 2.[209]

OpNav is indeed particularly useful when navigating small bodies. With poorly known shape-

models, gravity fields, and ephemerides, OpNav provides in-situ measurements of the spacecraft states.[16] Throughout robotics missions, navigation has allowed for bold mission developments including *Cassini*'s Grand Finale[64] which saw the spacecraft dive between Saturn and its rings. Reference 46 contains a complete history on optical navigation methods for more details.

Several different measurements can be used for optical navigation. Among these are star horizon,[166] centroid and apparent diameter,[41] star occultation,[173] and landmark tracking.[133] Each of these have specific application scenarios depending on the object they require to track, and the distance at which they do so. Centroid and apparent diameter measurements, for instance, find the limb of a body and use the knowledge of its actual size and shape. By extracting direction and distance, the spacecraft's location relative to the body can be determined.[42]

With only a few exceptions, all OpNav is done on the ground. The current state-of-the-art method for small body navigation is Stereo-Photoclinometry[76] (SPC) which allows the spacecraft to map and navigate the spacecraft environment with high precision. SPC combines stereo techniques with photoclinometry to derive the tilt of a surface. Once the surface tilts covering the body are obtained, the shape of the surface across each image are determined by integrating over the resulting tilts. The individual surface maps are then collated together to produce a shape model. Nonetheless, it relies very heavily on Earth contact for its intensive image processing algorithms.

The advent of small and nano satellites around the turn of the century, notably as Puig-Suari of Cal Poly and Twiggs of Stanford proposed a reference design for the CubeSat in 1999, has largely democratized the access to space. With an increase in frequent low-cost missions comes the need to provide publicly available, accessible, and robust methods for spacecraft navigation.

1.1.2 Roadmap to New Technologies

Navigation with ground in the loop has been fundamental to nearly all successful robotics missions to date, but carries intrinsic limitations. The most fundamental limitation is caused by the delay related to signals traveling to and from Earth-bound stations to the spacecraft. Radio signals travel at the speed of light, and while for most Earth orbiters light-time delay incurred is

not impactful, interplanetary missions can see minutes to hours of communication lag. In addition, the time spent processing measurements, convening over results, and deciding on future maneuvers can be time-consuming. Total ground-in-the-loop time can take over a week, down to roughly eight hours for very time-critical events.[17]

The NASA Deep Space Mission System (DSMS) — amalgam of the DSN and the Advanced Multi-Mission Operations System (AMMOS) — are paramount to communicating with spacecraft throughout the solar system but are strained.[145] Simultaneously, missions will hopefully continue to grow more common and more reliable, which indicates a need to reduce the load on the ground systems in place even routinely.

In addition to technological limitations, the National Aeronautics and Space Administration (NASA) 2019 budget[1] has shown almost an order of magnitude increase in Advanced Exploration Systems within Deep Space Exploration Systems: 97 to 889.0 million dollars from 2017 to 2019. Although this increase shows ungoing support from the United States government to explore the solar system, it potentially takes away financial support from other systems such as Exploration Systems Development which has seen a decrease of 5.43% (212.2 million dollars) over the same time frame.

The increase of autonomous capabilities for deep space exploration has been identified by NASA as a key enabling technology. Spacecraft ground-independence opens the door to entirely new missions designs all the while enhancing safety in case of communication failures. As navigation technology is central to interplanetary mission success, it is therefore applicable to many of the NASA Technology Area Breakdown Structure (TABS)[2]. Several of the most relevant items to this research are summarized in Table 1.1. The right column of the table explain very briefly how autonomous OpNav fits in to the

The advancement of robotics and autonomous systems will be central to transition space missions from ground-in-the-loop architectures to self-sustainable, independent systems: a nec-

[1] www.nasa.gov/content/fy-2019-budget

[2] www.nasa.gov/offices/oct/taxonomy

Table 1.1: Deliverables Relative to NASA Interests

TAB	Objectives and Deliverables
5.4.3 Sensors and Vision Processing Systems	Autonomous optical navigation requires image processing, feature detection
5.4.2 Onboard Auto Navigation and Maneuver	Autonomy is tested in rigorous unit and integrated scenarios. Fully coupled simulation environments enable flight-like
4.1.2 State Estimation	Optical Navigation uses celestial bodies for state estimation.
4.5.1 Vehicle Systems Management and Fault Detection Isolation and Recovery (FDIR)	By generating faulty measurements the simulation environment allows FDIR development
4.6.2 Relative Guidance Algorithms	OpNav uses relative measurements

essary step for outer-planet exploration. NASA's Road Map of crosscutting technologies, places Autonomous Systems on the top of the list, illustrating their versatility in NASA's first strategic objective 1.1 to expand human presence into the solar system. This intent is mirrored in technology areas TA 5 "Communications, Navigation, and Orbital Debris Tracking and Characterization Systems" and TA 4 "Robotics and Autonomous Systems."

Paired with a enhanced use of images and image processing —"Image processing is an increasingly vital part of new sensor systems, and significant work needs to be done in extracting the maximum amount of information from these images" — OpNav seems to continue to provide many capabilities in line with technology roadmaps. Furthermore, increasing precision in guidance navigation and control is also an objective in TA 4.1.2 State Estimation which "provides multi-sensor, vision-aided pose and velocity estimation."

Interests in visiting the icy moons of our solar system also presents development opportunities. In January 2018 NASA published a call for proposals C.29 Astrodynamics in Support of Icy Worlds Missions (solicitation NNH18ZDA001N-ADYN) to develop open-source code for mission analysis. This not only illustrates the desire to navigate remote bodies of the solar system, but also encourages

shared code-bases.

1.1.3 Summary of Applications

In recent years, open-source software packages have also broadened the impact of developments in specific fields. Robotics and AI have become a widely used in part with the help of freely available software libraries, papers, and documentation. Although space is emerging as an increasingly popular domain, a few of these components are missing. Within the confines of the law, notably the International Traffic in Arms Regulations (ITAR), astrodynamics can still provide a shared algorithm base widely available in the literature.

Simultaneously, developing autonomous algorithms for OpNav requires a specific environment. OpNav requires a simulated camera, and environment. This means a camera simulator must generate images as would be seen by a navigating craft. Since these images change dynamically as autonomous commands are triggered, the environment must generate images fluidly and quickly for realistic Flight Software (FSW) testing. Then, in order to test autonomous algorithms, several best-practices are suggested. The most fundamental one is to fly the same FSW than was developed and tested. Although seemingly trivial, the wide variety of tools and environments that a FSW stack encounters before flight often generates multiple versions of code, sometimes in different languages.[54] Furthermore, faults must be easy to incorporate into the simulation in order to challenge the limits of the algorithms. All this should be provided in analysis-capable speeds.

The underlying drivers for the OpNav developments that follow in this dissertation are listed as follows:

- Public Availability: Provide open-source code, in a rigorous software environment.
- Resource Attainability: Do not require the use of high-end computation capabilities.
- Robustness: Prioritize robust methods in order to remain favorable to autonomy and wide instrument capabilities
- Generalizability: Create general tools that can be reused for other purposes

Applications that stem from available, robust, and general tools are plentiful. An example, could be the C.29 proposal which aims to navigate icy moons harnessing open-source software packages. Navigation around these objects almost necessarily require some form of OpNav, and if the methods are to be reused, they must be general enough.

Figure 1.2: Juno Approach of Jovian System, Credit: NASA/JPL

Small sats and CubeSats are also a target type of craft for this technology. Often developed in universities with limited resources, development is often speedy and instruments are sometimes of limited capability. For OpNav, a single low-cost star tracker could greatly simplify spacecraft design by avoiding the need for multiple sensors. The ability to maintain attitude tracking during moon incursions has been demonstrated[66] and is an important milestone in that direction. This also ties strongly with technology development TA 5.4.3.3 "Miniature, High-Accuracy, Multi-Function Star Tracker," which paired with vision-based navigation could support missions such as MarCO, or CubeSats of the Cube Quest Challenge. Low resolution images might not generate many visible surface features, and can contain artifacts that are less common to be found in high-end navigation instruments. Images taken from the *Juno* star tracker are displayed in Figure 1.2. These display the kinds of images that are intended to be used for navigation in this dissertation.

Finally, open-source, fast, and realistic spacecraft simulations open the door to harnessing Machine Learning (ML) techniques. OpNav algorithms paired with ML can be be integrated in the greater context of spacecraft GNC. Indeed, with the possibility of creating vast data sets of

synthetic images, as well as continuous testing of spacecraft performance, ML can take hold in the field of astrodynamics and presents a state-of-the-art implementation of the goals listed above.

1.2 Optical Navigation for Autonomy

Autonomy relies on different techniques depending on the mission to which it is applied. For instance, low earth orbit missions have access to Global Position System (GPS) measurements, as well as telemetry data from various sources. On the other hand, deep space missions rely heavily on range and range rate measurements from the Deep Space Network (DSN). Yet there is increasing demand to use the network with limited bandwidth and operational capabilities. OpNav uses images of visible celestial bodies to extract position and velocity information. Therefore, it can rely solely on the spacecraft's interaction with the deep space environment instead of relying on Earth contact, provided that the algorithms used are computationally tractable.

The proposed research will focus on navigation in proximity to celestial bodies rather than in deep space. There is first a focus on relative attitude determination using Coarse Sun-Sensor (CSS) based sun-heading estimation paired with planet centroid headings. Secondly, Centroid and Apparent Diameter (CAD) orbit determination about well known bodies is developed. Together these methods strive to extend the current state of the art of autonomous navigation, both through sequence planning and using Machine Learning (ML).

The choice of these algorithms is also motivated by developing hardware and imaging methods. Event-based sensors are inspired by a biological retina and only detect changes in the image.[53] Operating in a significantly different way to traditional CCD-based imaging sensors, these innovations aim to service the field of imaging, processing, and Artificial Intelligence (AI). At the same time, Star Tracker developments continue to improve with High-Dynamic-Range (HDR) imagery and miniaturization[110] which continue to broaden instrument capabilities and intends to support visual navigation.[15]

Autonomous OpNav intersects vision processing systems, auto-navigation and maneuver, and state estimation. This research seeks to harness sensors and vision processing systems to improve

image acquisition and processing, and to increase the amount of useful data that is extracted for navigation. Simultaneously, flight software algorithms must be developed for onboard use, providing robust state estimation and allowing for failure detection. These algorithms interact heavily with the space environment. Testing them reliably therefore requires high-fidelity andcomputationally fast simulations that allow simulated crafts to navigate visually. Reinforcement Learning (RL) trains guidance laws that are adaptable to this environment, all the while paving the way for more fault detection capabilities through error classification. Neural networks can provide powerful image processing solutions that have been under tremendous development in recent years. Developing new processing methods such as neural nets can broaden the capabilities of previous navigation algorithms and enhance autonomy as a whole.

Autonomous OpNav is one of the components that leads to an increase in spacecraft autonomy as it provides measurements that can be gathered in deep space (without contacting Earth). Yet these measurements are often noisy, imperfect, and sometimes do not fully observe the spacecraft's states. Through the development of novel filters, uncertainty mapping; the design of fast, modular, open-source tools; and the exploration of OpNav's limitations and the capabilities of machine learning, the proposed research aims to enhance autonomous OpNav.

1.2.1 Previous and Ongoing Developments

Aside from optical measurements, other autonomous navigation methods are in development. X-ray navigation using pulsars to estimate the spacecraft's position[190] was recently flown on the International Space Station (ISS).[217] The Deep Space Atomic Clock (DSAC),[65] is also being tested to permit one-way radio-frequency measurements sent from the Deep Space Network (DSN). Finally, *StarNAV*[44] aims to develop a new sensor system to detect relativistic perturbations in the wavelength and direction of observed stars to measure spacecraft velocity. Although just one in many methods, this research focuses on enhancements to autonomous OpNav for spacecraft state estimation. This has also seen developments and is notably planned for use on Orion's Exploration Mission 1,[99] as well as for Jupiter and Saturn exploration using the planet satellites.[21] Autonomous

OpNav remains a sought-after navigation method as it requires only cameras (which can be both light-weight and used for other purposes) and fundamentally relies on imaging the object that is being studied and orbited as opposed to Earth-based data, or distant pulsars.

Past missions such as *Deep Space 1*,[108] *Stardust*, and *Deep Impact*[123] relied heavily on OpNav. Nevertheless, only *Deep Space 1* and *Deep Impact* used autonomous OpNav (AutoNav[108]): as a technology experiment and mission enabler respectively. *Deep Space 1* used a first implementation of AutoNav in order to determine its orbit within the solar system, while *Deep Impact* used another implementation of AutoNav in order to ensure contact with the asteroid Tempel 1. The first implementation uses beacons (planets and certain stars) in order to triangulate its location. The second version used a center of brightness algorithm in order to instruct the guidance algorithms on the proper impact trajectory. Both were successful and built confidence in the potential use-cases for autonomous navigation. Optical images are also used for Entry Descent, and Landing (EDL), as seen in practice with the *DIMES*[35] system used to land Mars rovers. These use cross-correlation methods[29,143] developed specifically for EDL. These missions are examples of using autonomy as a mission enabler: the goals would otherwise not be attainable due to round-trip light time.

As stated previously, another advantage to autonomy is the reduction of costs by lightening the load on the ground resources doing navigation operations. Autonomous OpNav has up to now been only used for small body missions. There are not only more mission scenarios (deep-space formation flying or rendezvous) which require autonomous guidance, navigation, and control (GNC), but also no developments for on-orbit autonomous optical navigation for routine station-keeping. Currently radiometric doppler data is the preferable measurements used on-orbit.

OpNav has been tested and used since the 1960s in the Gemini and Apollo missions. The most recent missions that have used state of the art, on-board, autonomous OpNav have used AutoNav.[108,121,123] Many other missions such as New Horizons are using OpNav.[86] This is most frequently taking the form of OpNav campaigns[168] many months ahead of an encounter to progressively narrow down on the position of the object and the B-plane parameters. In contrast, the proposed research focuses on autonomy in proximity to a known planet using AutoNav as a

baseline, emphasizing autonomy on shorter time-frames.

1.2.2 Areas of Active Research

1.2.2.1 Simulation Developments for Closed-Loop Optical Navigation

Currently, the field does not provide an open-source software package with fully coupled spacecraft dynamics and Flight Software (FSW) capabilities, especially for OpNav mission scenarios. Although many simulation packages are available, few are paired with a high-fidelity, fast, open-source dynamics engine that can read in images and modify a spacecraft's trajectory; therefore changing future detected images. This does not preclude testing development of high-end GNC algorithms, but it requires the making of specific simulations for each algorithm. Testing algorithms piecewise is also a method for successful simulation,[146] yet it doesn't generalize well to other problems and scenarios. All the work presented seeks to make the novel contribution of a closed loop software-in-the-loop OpNav simulation package. The most competitive simulation packages to date are listed as below:

- DARTS[24] paired with DSENDS, Dshell, or ROAMS[1] provides high-fidelity dynamics and visualization capabilities.[132] These tools are developed in a closed software environment that are not generally extensible by researchers outside of the Jet Propulsion Laboratory. Furthermore, although DARTS provides closed-loop dynamics and control, it does not suggest the use of a visualization for image processing and OpNav.

- AGI-EOIR[2] is an STK-based visualization tool that uses physics based radiometric sensor and target image simulation. This software can provide highly accurate sensor images, but these are exported to file and not integrated into a closed-loop simulation. Although AGI-EOIR is not open-source, AGI is developing *Cesium*,[3] a realistic mapping and visualization package. Yet

[1] `https://dshell.jpl.nasa.gov`

[2] `http://www.agi.com/EOIR`

[3] `https://www.cesium.com`

nothing suggests this new software can support high-fidelity dynamics or closed-loop environment interaction.

- *Astos Solutions*[1] is a European based company developing software and equipment for space applications. In the past, they have provided support for missions,[206] and have recently developed a Camera Simulation in addition to their astrodynamics software. Astos describes a configurable dynamics and environment simulator running on a *dSPACE*[2] real-time platform and a LIDAR and camera simulator.[109] This software is written in Matlab/Simulink, which provides modularity but is not open-source and does not scale well when doing Monte-Carlo analysis.

- In the field of robotics, ROS[4,175] and its sister software package *Gazebo* are open source and provide hardware-in-the-loop capabilities. Yet, the applications and design are very robotics focused and do not provide sufficient spacecraft models and features. An open and extensible software solution like ROS has not existed for the spacecraft community in the past.

The above tools provide a wealth of solutions for simulation and visualization. Yet none of them provide the full functionality that is required for an OpNav simulator. The required simulation must provide faster than realtime, high-fidelity dynamics; have FSW integration and filter implementation, as well as hardware in the loop capabilities; be OpenSource for broad use in the astrodynamics field; and with closed-loop visual feedback.

The presented tool consists of the *Basilisk*[1,113] astrodynamics framework interfacing with a *Unity*-based *Vizard*[219] visualization that provides a synthetic image stream of a camera sensor. *Basilisk* has been developed around high-performing dynamics which are high-fidelity, fully coupled, and faster than realtime.[8]

[1] `https://www.astos.de`

[2] `https://www.dspace.com/en/inc/home.cfm`

1.2.2.2 Attitude Determination in Optical Navigation

The current ubiquitous, affordable, and high performing instrument for attitude determination is the star tracker.[131] Using well-known distant stars, spacecraft can determine their inertial attitude anywhere in the solar system (provided the instrument is not occulted). This will remain the de facto method for inertial attitude determination in the foreseeable future.

Heading based attitude determination — using CSS measurements[161] or OpNav measurements[13] — are still in development nonetheless.[141,157] Headings provide a secondary attitude measurement which can be used for flight software robustness checks and fault detection, while also being used for their primary purpose: in pointing modes (Sun-point, science-point, Earth-point, etc.). Indeed, heading based attitude determination methods provide relative attitude to their targets. Depending on the scenario, the position of the target might not be well known in the inertial frame: when pointing at an asteroid or at a science target. In this case, relative attitude is paramount to accurate pointing. In scenarios where the target's inertial position is well known, relative attitude allows to solve for offsets in the camera position and orientation. By combining CSS measurements for sun-heading with OpNav measurements of a nearby planet, a full attitude estimate can be extracted. Therefore, the proposed work attempts to add an attitude component to autonomous OpNav methods such as AutoNav.

1.2.2.3 Orbit Determination and Image Processing Uncertainties

Centroid and Apparent Diameter (CAD) measurements find the limb of a body and use the knowledge of its actual size and position. By extracting direction and distance, range information is extracted from planet images.[13,42] These methods require minimal image processing power, and are relatively fast to implement. CAD detection has continued to be used and improved,[20] while recent papers have mapped uncertainties for Cholesky factorization space[49] (CFS) and iterative horizon reprojection[47] (IHR) methods.[97]

Other OpNav methods also yield accurate navigation results. Measurements derived from

landmark observations,[133] point distribution methods,[203] or crater detection[170] are some of many feature tracking methods which provide promising results. However, they come at a computation cost. Indeed, the current state of the art for OpNav is Stereo-Photoclinometry[178] (SPC) which allows the spacecraft to map and navigate the spacecraft environment with high precision. Nonetheless, it relies very heavily on Earth contact for its intensive image processing algorithms. With the goal of autonomy, this research will focus on on-board methods for image processing. Similarly, ORB-SLAM[155,156] and other cross-correlation methods[29,143] are commonly implemented in GNC research. These hold great promise for small body autonomous orbiting and relative spacecraft navigation. For preliminary developments, the proposed research will focus on simple CAD, with the intent of allowing more intensive image processing methods in the future.

Current filter developments have given pixel and line measurements (angles on a camera defined by pixel location) as measurements for filters.[166] This has worked successfully on numerous OpNav missions.[46] The proposed research first seeks to provide a more modular solution to the Autonomous OpNav problem using simple, off-the-shelf circle fitting algorithms. In order to do this, the filters will take in as measurements their relative pose vector, agnostic of what image processing algorithm provided it. The same filters can be used with different image processing algorithms, and the latter can determine the quality of it's measurement prior to use for filtering. Secondly, the work aims to implement *Hough Circles* for the first time in a end-to-end navigation toolbox. Celestial bodies can generally be approximated as ellipsoids, while most planets can be approximated as spheres under certain assumptions. Both ellipsoids and spheres mathematically project to ellipsoids on a camera plane. This has forced ellipse fitting to become the de facto way to fit limbs for navigation. In this framework, clustering methods such as *Hough* transforms are too numerically complex and slow and have been discarded for navigation though discussed.[41,129,225] This research aims to prove that circle fitting can provide similar results in ideal cases.

1.2.2.4 Closed-Loop Autonomous Scenarios under Uncertainty

The current state of the art regarding autonomous navigation remains the AutoNav algorithm used for Deep Impact.[121,122] To this day, it remains the only algorithm to have been flown for on-board attitude and trajectory modifications using OpNav measurements.[46] AutoNav developments came with the many challenges that arise when targeting asteroids of unknown shape, position, and size.[123] Its success has encouraged further developments in this direction. These extensions have been primarily focused on flybys and landings on small bodies,[18] as well as approaches to Mars.[21]

The fifth chapter aims to simulate an enhanced, more general, version of AutoNav in *Basilisk*. The first novel addition will be the generalization of the applications: AutoNav has been used for flybys (*Deep Space 1*) and for impact (*Deep Impact*). This work will have similar guidance algorithms — which determine position and attitude and correct drifts and errors — generalized to orbit scenarios. The second novel component is in image processing: AutoNav has only used Center-of-Brightness for targeting. The challenge of robustness greatly motivates the use of *Hough Circle* transforms which are very resilient to image noise and corruptions. This research implements CAD methods which provide more information and ultimately allow the generalization to simultaneous orbit and attitude control.

1.2.2.5 Applying Machine Learning to Astrodynamics

Although pre-computed sequences have led to mission success in the past (Deep Impact), they remain vulnerable to failures and unexpected errors. A growing number of RL and ML methods are appearing in the literature, mostly focusing on the application of learning approaches to control problems in uncertain environments. Several works have considered RL in the context of autonomous aerobraking planners,[51,91] with mixed results; others explore ML techniques for asteroid proximity operations[79] or autonomous lunar landing.[179] In contrast, this work explicitly examines applications of RL to high-level spacecraft planning and decision-making problems that have traditionally been the domain of rigid expert policies or optimization-focused strategies.

In similar developments, neural networks have shown promising results in countless fields. Neural nets take many forms ranging from Artificial Neural Networks (ANN) and Feed-Forward Neural Networks (FFNN),[199] to Convolutional Neural Networks (CNN),[164] Boltzmann Machines[198] and Bayesian Networks.[34] As universal function approximators[138] which learn under supervision (based on example input-output pairs provided) they have seen a great success in fields ranging from medical imagery[150] to digit recognition.[188] In navigation, CNNs are recently used for depth mapping[149] and Terrain Relative Navigation (TRN).[179]

1.3 Research Overview

1.3.1 Thesis Goals

The proposed research combines Software and Simulation development with GNC. In order to enable robust testing of these developments, there is a need to create a realistic simulation that allows for the implementation and testing of OpNav methods. The first goal revolves around attitude determination, more specifically on using OpNav measurements for heading determination while avoiding unobservability. The second goal, in the context of relative orbit determination, is to analyze and develop the use of robotic's inspired algorithms. Algorithms instantiated here have been discounted in the past either due to speed or a lacking measurements quality. This is a novel goal as *Hough* transforms for circle finding have not been used for ellipses given prohibitively lengthy processing times. By fully linking and analyzing these algorithms in a closed-loop fashion, they can be tested next to higher fidelity methods. In a third goal, fault detection can be performed and robustness can be evaluated. These goals will together enhance capabilities of AutoNav to on-orbit orbit determination with a resolved planet. This motivates the use of circles-fitting with a clustering method such as *Hough Circles*. Indeed the research aims to demonstrate the robustness of the method in the presence of difficult measurements. Finally, Machine Learning methods are implemented in order to provide solutions to astrodynamics problems. Such developments aim to improve decision-making for spacecraft autonomy are tested and developed. Neural networks are

also explored for lower-level image processing improvements. The goals of each research topic are found in the following chapters:

- **Chapter 2:** Provide a closed-loop dynamic-visualization software package for OpNav simulations. This allows for realistic and accurate closed-loop OpNav testing.
- **Chapter 3:** Develop a robust attitude relative heading filter which uses CSS data or visual measurements.
- **Chapter 4:** Implement Centroid and Apparent diameter OpNav filters for state estimation using the *Hough* transform and compare performance to state-of-the-art methods.
- **Chapter 5:** Design fault injection into the simulation and analyze method robustness. Provide fault detection algorithm that combines method strengths to enhance autonomous capabilities.
- **Chapter 6:** Develop reinforcement learning methods applied to spacecraft decision making in the context of OpNav, and train neural nets for image processing.

1.3.2 Outline of Contributions

The following dissertation presents contributions to the field of aerospace engineering sciences in several regards. The first contribution is the development of an open-source closed-loop dynamic-visualization software package for OpNav simulations. The designed tool provides the backbone simulation used throughout this work, but remains available and general to a wide variety of other use-cases.

Another contribution lies in heading determination and filtering with limited measurements. By leveraging work done on sun sensors, heading determination for OpNav is also enhanced. The filters implemented, dubbed Switch-Filters, choose between different kinematic representations of the state in order to avoid singularities.

This is followed by a contribution to centroid and apparent diameter for orbit determination. The work analyzes the use of circle finding through the use of the *Hough* transform on orbit about known celestial bodies. Unlike previous work, this research moves away from ellipse-fitting in order to implement a more robust clustering method in a fully closed-loop environment. Results show that performance is on par with conic-fitting methods even in ideal cases. This work is subject to some assumptions, notably in situations where the planet is not largely oblate and a pointing algorithm allows it to remain centered on the image plane. The research then explores fault detection algorithms that can stem from the use of robust methods. These developments are found between Chapter 4 and 5. Chapter 4 includes the end-to-end analysis of the *Hough* transform for OpNav orbit determination and pointing. Then, in Chapter 5, fault detection algorithms implemented with two methods as inputs to test against the result of environmental and camera corruptions.

The last contributions lie in the use of machine learning methods for autonomy. Although these methods are under ongoing development, the core capability is displayed. The applications range from high-level autonomy for decision making to optimizing neural nets for image processing.

Chapter 2

Closed-Loop Optical Navigation Simulation Package

2.1 Overview

In this chapter, a software architecture provides a realistic environment to develop, run, and test novel and autonomous visual spacecraft navigation and control methods. This architecture harnesses two main components: a high-fidelity, faster-than-real-time, astrodynamics simulation framework in concert with a sister software package to dynamically visualize the simulation environment. Autonomy in OpNav requires a simulation that closes the loop between the astrodynamics simulation and image generation; this allows for realistically emulated maneuvers such as fly-bys and orbit insertions, as well as on-orbit autonomous activities. Yet, there are no open-source software packages that provide fully coupled spacecraft environments and FSW enabling these mission scenarios. A solution for dynamic simulations is the interfacing of the Basilisk astrodynamics framework with a Unity-based visualization Vizard. The coupled software tools generate a stream of synthetic images from a simulated camera sensor. This modular and extensible framework allows optical GNC algorithms to be run in a closed-loop format purely in software. The optical measurements are generated in the visualization and passed to the simulation, allowing for real-time control and FSW decision making. The *Vizard* software has the ability to import shape-models, planet maps, and move into an instrument point-of-view. Paired with open-source image processing libraries, these combined components provide all the necessary pieces to fully simulate autonomous, closed-loop, OpNav scenarios in a faster-than-real-time configuration. The tool establishes the foundation for progress in the autonomy sector. It makes full-fledged FSW testing with real fight

environments possible. Furthermore, this enables more realistic and extensive testing of the software, which in turn increases reliability of the GNC methods as they are refined. This chapter presents the *Basilisk* and *Vizard* interface architecture, its performance, and develops a example scenario. The image processing methods are displayed and the visualization scenes are validated for pointing purposes, which develops an autonomous pointing algorithm developed in the software environment.

2.2 Motivation and Previous Work

As previously described in Chapter 1, future space exploration requires autonomy as a key technology enabler.[195] Reducing the frequency of ground-in-the-loop communication allows for less expensive mission support systems. Aside from alleviating ground-based tracking, autonomous GNC with ground-in-the-loop navigation supplies the system with redundancy if communication fails or when maneuver time and spacecraft distance make ground based control impossible. Whether it be for robotic exploration of the solar system, manned spaceflight, or small satellite development, autonomy opens the door to new mission concepts.[108,121,123]

One key enabler for autonomy is on-board optical navigation, as it provides measurements that can be gathered without contacting Earth. Furthermore, it provides direct information on what is often the subject of the mission's scientific objectives. This chapter outlines a novel framework which seeks to combine navigation algorithms within a simulated spacecraft environment. These algorithms require a reliable and extensible testbed to be developed and refined. This simulation testbed must provide realistic spacecraft simulations, model the local space environment, and create three-dimensional visualizations of both the spacecraft and the environment. Since the guidance and control development typically involves extensive Monte-Carlo sensitivity analysis, computational speed is of paramount importance.

High-fidelity dynamics simulations provide a vital test environment for spacecraft and robotics development. Existing tools such as DARTS[24] paired with DSENDS, Dshell, or ROAMS[1] provide

[1] dshell.jpl.nasa.gov

high-fidelity dynamics and visualization capabilities.[132] These tools are used in a closed software environment that are not generally extensible by researchers outside of the Jet Propulsion Laboratory. Furthermore, although DARTS provides closed-loop dynamics and control, it does not permit the use of visualization snapshots for image processing and OpNav. AGI-EOIR[1] is an STK-based visualization tool that uses physics based radiometric sensor and target image simulation. This software can provide highly accurate sensor images, but these are exported to file and not integrated into a closed-loop simulation. In the field of robotics, ROS[4,175] and its sister software package *Gazebo*[2] are open source and provide hardware-in-the-loop capabilities. Yet, these are tailored for ground-robotics applications and do not provide sufficient spacecraft models and features. An open and extensible software solution like ROS has not existed for the spacecraft community in the past.

Stand-alone highly-realistic visualizations are also in development. With increasing image generation capability, pioneered notably by the video-game industry, space imaging is given more and more tools for creating synthetic camera data. This is seen for instance with the development of *PANGU*[3144] which has the ability to model the surfaces of planetary bodies such as Mars, the Moon, Mercury and asteroids using real and synthetic data. *PANGU* is designed to provide a high degree of realism yet only operating at near real-time speeds. Similarly *OpenSpace*[4] and Airbus' *SurRender*[5] software both provide highly realistic simulations. Issues arise when taking speed of image generation into account, as well as open-source codebases. Indeed, only *OpenSpace* is free to use. Nonetheless, the interface described in this chapter is applicable to a wide variety of simulation-visualization pairs and can be ported to other software packages. If other simulations or visualizations have more advantageous features in the future, this work will generalize well to include them.

[1] www.agi.com/EOIR

[2] gazebosim.org/

[3] pangu.software

[4] www.openspaceproject.com

[5] www.airbus.com/space/space-exploration/SurRenderSoftware.html

OpNav simulations have focused either on the image processing component,[128] on the estimation component,[40,43] or on using mission data.[61] These provide valuable insight on many facets of the problem; yet no common open-source software package exists that provides modularity and repeatability while bringing together contributions from many developers. Furthermore, most current simulations do not couple spacecraft dynamics and control into OpNav measurements.[133] Camera models are linked to the image processing and filter performances,[42] but this does not loop back to the spacecraft control algorithms.

Figure 2.1: Mars Orbit Insertion Scenario with Astrodynamics Simulated by *Basilisk* and Visualized in *Vizard*[219]

One example scenario is an orbit insertion maneuver, which occurs in close proximity to the body of interest during a short time span (too short for ground intervention) and is central to mission success. A spacecraft's on-board use of optical measurements can provide assurance of proper maneuver execution, and notably, if faults occur. Another example is the *New Horizons* Pluto fly-by. The mission studied the likelihood of having Pluto in the image frame,[186] whereas autonomous pointing could have provided more confidence by centering Pluto in the image frame.[89]

Both these examples showcase the potential for more autonomy in the chain between OpNav, attitude control, and trajectory modifications.

Basilisk[1,113] is a highly modular astrodynamics simulation framework that allows for the rapid simulation of complex spacecraft dynamics.[8] Key astrodynamics features include solar radiation pressure,[114–116] imbalanced reaction wheels,[3] imbalanced control moment gyroscopes,[2] flexible solar panels,[7] fuel slosh,[6,26] depletable mass,[169] as well as multiple body gravity and gravitational spherical harmonics. The sensor simulation and actuator components couple with the spacecraft dynamics through a publish-subscribe (pub-sub) messaging system.[113] A state engine allows for complex spacecraft dynamics to be setup without having to develop and code any dynamics differential equations.[5] An associated visualization is built using the *Unity* gaming engine, is called *Vizard*, and was developed by Jennifer Wood.[219] This dissertation focuses on the required interface with *Basilisk* to achieve OpNav simulation capabilities. Here the simulation messages are streamed directly to the visualization to illustrate the spacecraft simulation and environment states. Figure 2.1 shows a Mars Orbit Insertion (MOI) performed in *Basilisk* and visualized inside the *Vizard* software. As ROS and *Gazebo* do for the robotics community, combined *Basilisk* and *Vizard* provide an open and extensible software architecture to both simulate and visualize spacecraft dynamics and control scenarios for the astrodynamics community.

This work explores a new software architecture where *Vizard* is not just used to visualize the *Basilisk* simulation states, but becomes itself a visual sensor module for *Basilisk*, thus allowing for closed-loop visual control simulations to be performed. This allows for visual guidance and control algorithms to be tested in a faster-than-real-time software platform that is also suitable for Monte-Carlo type sensitivity studies. The created visualization images are controlled and shared via a new two-way connection between *Basilisk* and *Vizard*. This is a challenge as it requires frame synchronization such that any type of camera resolution can be simulated while maintaining synchronization with the dynamics simulation. Furthermore, it is desirable to design a flexible communication interface between two software packages such that they can be run on a single or multiple computers.

This new *Basilisk-Vizard* software integration has the ability to support many scenarios at the cutting-edge of autonomy; these include optical deep space navigation, formation flying, close proximity and servicing applications, as well as visual navigation about small celestial bodies such as asteroids. Because *Basilisk* also allows for formation flying capabilities,[165] formation flying dynamics have the potential to be paired into an OpNav framework for relative formation control. This allows for true-scale spacecraft models to be used for visual control, with features like self-occultation and realistic camera model. Furthermore, implemented star-maps could be used for realistic attitude determination and control, all within a closed loop software framework. For entry, decent, and landing (EDL) and asteroid missions' safety, these developments can add an important element of reliability by providing a testbed for autonomy and quantifying performance. Simultaneous Localization And Mapping (SLAM) and cross-correlation methods could also be implemented and tested in a realistic spacecraft environment. These algorithms are currently being developed for novel navigation purposes notably within NASA and ESA.[29]

This chapter details the new software architecture that allows the *Vizard* software to become a highly configurable visual sensor module for *Basilisk*. First, the numerical performance and computational cost of the communication overhead is explored; second, the visualization optical sensor module is validated for specific OpNav purposes; finally, a pointing scenario is developed in order to showcase the architecture's performance.

2.3 Software Interface Architecture

2.3.1 Overview of the Basilisk and Vizard Software

Basilisk is an open-source astrodynamics framework being developed by the University of Colorado Autonomous Vehicle Systems (AVS) lab and the Laboratory for Atmospheric and Space Physics (LASP). By implementing high-fidelity, faster-than-real-time dynamics, it allows to simulate spacecraft in realistic flight conditions. The inherent speed of the framework and its multi-process Monte-Carlo capability provides high-end analysis tools. In the *Basilisk* simulation, FSW

and spacecraft models are placed into different Processes (or Task Groups) to isolate their individual messages. By communicating through the pub-sub messaging system, blocks of code can be added and contribute to the simulation without necessary knowledge of other blocks, as seen in Figure 2.2. This interface allows for closed loop control algorithms and simulations to be developed and tested in a highly modular manner where each component has its own unit and integrated tests.

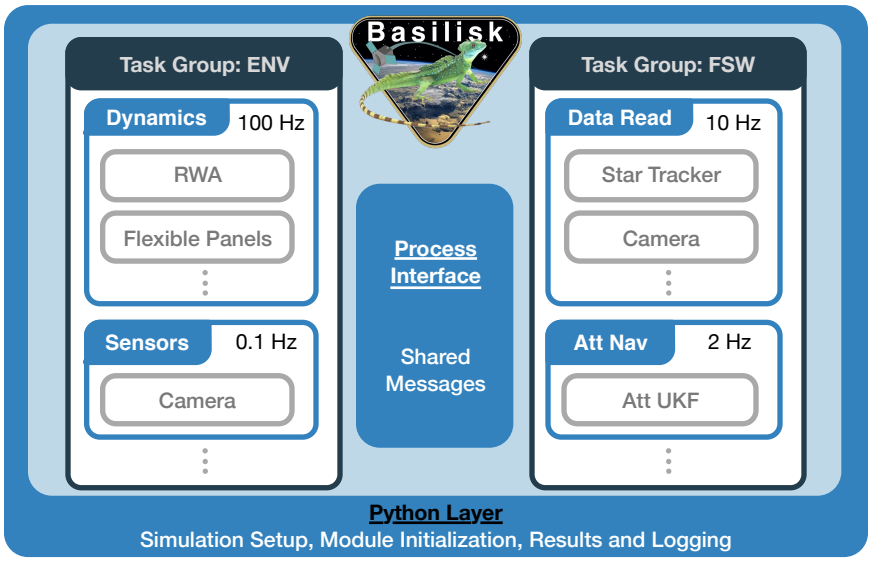


Figure 2.2: Schematic Illustration of the *Basilisk* Architecture[1,113,140]

Alongside this effort, *Vizard*[219] receives *Basilisk* messages and dynamically displays these states. *Vizard* has the ability to import shape-models and planet maps, as well as display and render instrument point-of-view windows. Paired with open-source image processing libraries, such as *OpenCV*[1], these combined components provide the necessary software components to fully simulate autonomous, closed-loop OpNav or other visual sensing and control scenarios. This thesis develops the software architecture that allows *Vizard* to be integrated into *Basilisk* as a visual sensor module. As a fully open-source project, *Basilisk-Vizard* allows for any user to contribute to the code base, and therefore centralizes progress within astrodynamics.

[1] opencv.org

The modularity of *Basilisk* comes from the fact that modules publish and subscribe without requiring knowledge of other existing modules. Processes (or Task Groups) as pictured in Figure 2.2 are containers that allow their internal Tasks to communicate, but prevent inter-Process communication. These message containers can also interface to allow inter-Process communication which helps manage the separation of FSW and simulation models. This naturally welcomes another actor: the visualization. By creating a module with access to the required messages, the communication between the software nodes is established.

2.3.2 Faster-Than-Real-time Interfacing

In this thesis, a the term 'realtime' is used to describe simulations that run at the same speed as the physical system being modeled. Although running one order of magnitude faster-than-real-time can be a clear speed up, it is often not enough for Monte-Carlo capabilities. Therefore the term 'faster-than-real-time' is reserved for a two order of magnitude speed-up or more.

The software architecture of *Basilisk* allows *Vizard* to capture information from the spacecraft's environment and communicate it to *Basilisk* mid-run. *Vizard* then creates a three-dimensional visualization of the space environment including planets, moons, stars, other spacecraft, all from the perspective of the current spacecraft location and orientation using a specific body-relative camera frame perspective. After rendering this view the resulting image bitmap must be transferred back to *Basilisk* as an image message.

Implementation of the *Vizard* to *Basilisk* interface produces several key challenges. The first is making two heterogeneous software entities written in different programming language of C/C++ and C# communicate. Next, the simulation must execute faster-than-real-time to be suitable for navigation and control sensitivity analysis. Finally, these heterogeneous components must be integrated while maintaining synchronous operation of the modules through each integration time step. Two types of connections between *Basilisk* and *Vizard* are considered: the direct connection, and a connection via *Black Lion*.[54,140]

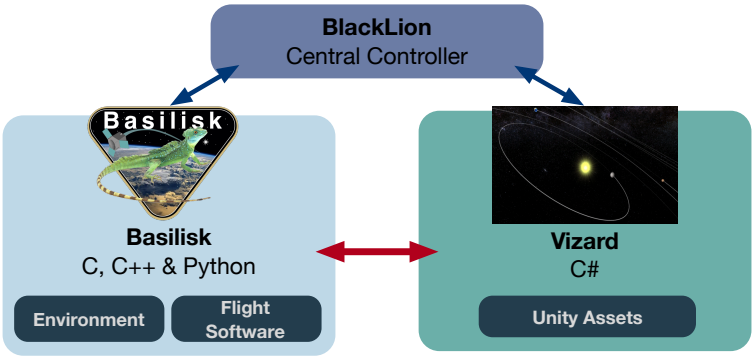


Figure 2.3: Interactions Between *Basilisk*, *Black Lion*, and *Vizard*

2.3.2.1 Black Lion

First consider the case where *Vizard* and *Basilisk* are part of a larger distributed spacecraft simulation which uses the *Black Lion* architecture[54,140] to communicate across simulation nodes. The benefit of this approach is that *Vizard* and *Basilisk* can be distributed simulation environments and on different machines, at the cost of additional central controller software. The *Black Lion* package developed in the AVS Lab is middleware that ensures proper interfacing between nodes in a heterogeneous, possibly distributed spacecraft simulation. Essentially the message passing interface concept of *Basilisk* is expanded to function across a range of heterogeneous simulation components such as a flight processor emulation or ground software system. For the scope of this paper the *Black Lion* nodes are *Basilisk* and *Vizard* as illustrated in Figure 2.3.

In summary, *Black Lion* ensures :

- The transport of binary data via a transport layer (Transmission Control Protocol or TCP). Although User Datagram Protocol (UDP) provides a faster connection, the three-packet exchange provided by a TCP provides the high-reliability necessary for physical simulations. This ensures the camera image is received at the correct time by *Basilisk*.

- The marshaling (or translation) of binary data. Each node must know how to convert the received bytes into structures that can be managed internally.

- The synchronization of nodes to keep all the nodes in lock-step during the simulation run.

The central controller acts as a master in the synchronization of the nodes, and a broker in the data exchanges.

Google Protobuffers[1] are used to provide a platform and language agnostic translation layer library to facilitate marshaling and unmarshaling of data between the two simulation applications. By creating these Protobuffer structures, both the C++ code in *Basilisk* and the C# code in *Unity* can read in and write out the necessary content. This method is currently in use at LASP for real-time *Basilisk*-based flat-sat testing while integrating the *Vizard* visualization. It notably allows running distributed simulations over a network. Users can distribute nodes across machines, use hardware in the loop, or run the *Vizard* on a computer with a high-end graphics card. This provides a wealth of optimization strategies with the slight added complexity of interfacing with middleware. Because *Black Lion* enforces synchronization across modules, the synthetic visual sensor images are guaranteed to remain in sync with the spacecraft dynamics simulation in *Basilisk*. This method is primarily aimed at more mature mission concepts. By using hardware in the loop, the faster-than-real-time aspect is lost, but more critical tests can be run.

2.3.2.2 Direct Communication

When performing fast analysis or making design choices, it is desirable to run *Basilisk* and *Vizard* on the same machine without having to synchronize with other spacecraft simulation components, such as ground software. In order to simplify the interface, a direct communication is implemented which allows for a two-way communication between *Basilisk* and *Vizard* without using *Black Lion* as a middle-ware interface layer. In the absence of a central controller, the `vizInterface` module written in C++ takes on the synchronizing responsibilities. Nevertheless the same methods and tools seen in the *Black Lion*-based implementation are used:

- The transport layer used is a TCP, implemented with *ZeroMQ*[2].
- The translation layer uses the same Protobuffer structure.

[1] developers.google.com/protocol-buffers

[2] zeromq.org

- The synchronization is enforced in the simulation through a blocking communication interface: *Basilisk* waits for critical responses from *Vizard* through *ZeroMQ* before continuing the simulation.

The direct communication protocol utilizes a separate thread to spawn *Vizard* from the python layer to start a request-response pattern. In this direct communication scenario, there are two main modes that the interface can work in: a lock-step mode and a performance mode. The core difference between these is the frequency of communication between the two nodes.

(1) Lock-step: In the lock-step mode, *Basilisk* sends updates at every time-step whether or not an image is requested. Lock-step provides a fluid visualization, and renders both the spacecraft camera and *Unity*'s main camera to screen allowing for user-feedback on the simulation setup and initialization. This also opens the possibility of controlling the simulation from the visualization, as it will always wait for a message verifying *Vizard* has received the simulation message. In lock-step mode, *Basilisk* always waits for a message from the *Vizard* saying it can move forward. This keeps the synchronicity as the message queues looks the same on each side and the visualization always has the latest message.

(2) Performance: In performance mode, the visualization and the interface are simplified. On the *Vizard* side, the spacecraft camera becomes the main camera. Furthermore, *Vizard* only places and updates simulation states if an image is requested and `vizInterface` only sends a simulation update when an image will be requested. This brings down the number of TCP pings to the camera image rate, instead of the simulation time step. A separate, OpNav-specific application is developed for this purpose. Paired with *Unity*'s 'batchmode' command-line argument — which makes the *Unity* application a rendering engine without initializing a user interface — the application can stay silent and run in the background effectively reducing the *Vizard* application to a simulation module.

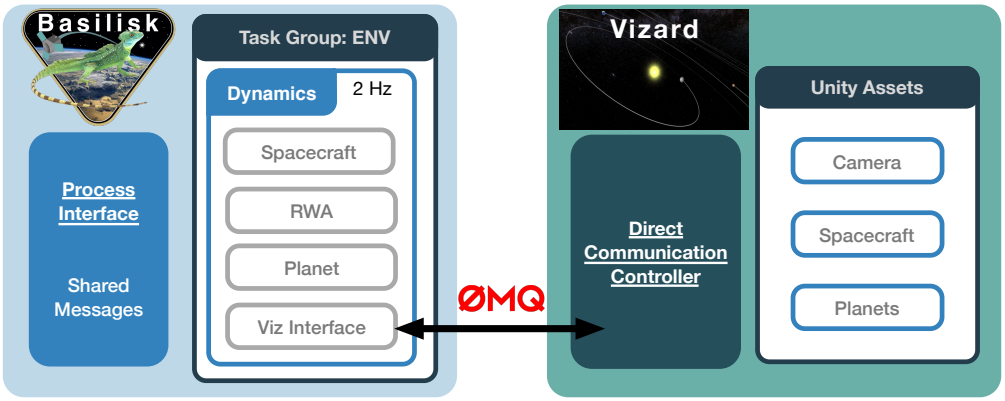


Figure 2.4: Direct Communication using the Viz Interface

2.3.2.3 Interface Implementations

Figure 2.3 shows the interaction between the major software nodes for both interface scenarios: via *Black Lion* (blue arrows) or directly (red arrows). Figure 2.4 outlines the details of the interfaces of the direct communication option. As stated previously, *Basilisk* modules publish and subscribe to messages via the Process (or Task Group) message memory space without any knowledge of other existing modules. *Basilisk* contains a new C++ *Vizard* interface module which reads the required *Basilisk* messages, writes them as protobuffers, and sends them via *Black Lion* or *Vizard* directly. The Visualization interface then unpacks the protobuffers in order for the rendering engine communication controller to use the data.

These design choices reflect the desire for a modular yet robust architecture. The use of Google Protobuffers allows for platform independent communication; *Unity* provides a user friendly and vast community for environment development; *Black Lion* is the middleware that connects and applications and synchronizes them across separate machines. Alternatively, the direct communication is created between the visualization and *Basilisk* for the ability to debug, test, and analyze simulations to greater effect. These tools provide the building blocks for the framework presented in this thesis.

2.3.3 Information Flow

In both simulation communication configurations (BlackLion or direct communication), the information flows back and forth between the two nodes. The main points of the information flow are detailed below and shown in Figure 2.5. The schematic presents an example simulation to showcase the general capabilities of the framework, and represents only a fraction of the possible implementations. This architecture's greatest strength is its potential to support further complexity and development, thus making it extensible. This section shows an example of the data flow that can be achieved with the architecture.

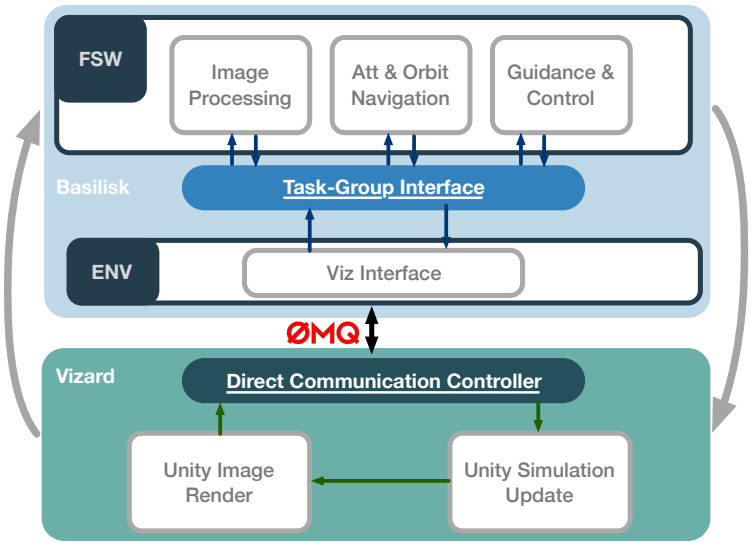


Figure 2.5: Information Flow Between the Visualization and the Simulation

(1) The `vizInterface` module in *Basilisk* checks for new information in the simulation. If any data has changed at a simulation time-step, the protobuffer message is updated. In the absence of change, the module will do nothing.

(2) The protobuffer is passed to *Unity* (black arrow next to *ZMQ* logo) and packed in a dictionary. This allows for a simulation update on the game engine side and an image render if requested.

(3) If an image is requested—this can be done through the presence of an image request message

in the simulation, or if the simulation time is a multiple of the camera render rate—*Vizard* renders the texture viewed by its internal camera according to specifications. The camera position and orientation are sent to *Vizard*, and the camera frame is rotated by $\boldsymbol{\sigma} = \left[\frac{1}{3}, \frac{1}{3}, -\frac{1}{3}\right]^T$ (rotation represented as an MRP) in order to abide by the *Unity* camera definition (which points out of the $-X$ direction). This is then sent back to Basilisk in a bit-map format while the simulation awaits the return message.

(4) The received image is unpacked in the `vizInterface` module, the endianness is swapped (discrepancy with incoming image), and repacked in a C structure for the rest of the *Basilisk* simulation to use (with relevant information such as time of capture and camera used). FSW modules are traditionally written in C, therefore the bitmap is recast to a void pointer in *Basilisk*. This prevents numerous copy operations of the image data and requires no dynamic allocation (which requires a `malloc` in C).

(5) This image will be read by the image processing module, which will extract spacecraft relative position with Centroid and Apparent Diameter (CAD) algorithms. Figure 2.5 could also picture camera models for additional realism. This can include the CCD's sensitivity to certain colors, realistic jitter using true attitude variations, etc.

(6) This value is next sent to an Orbit Determination filter or Attitude Guidance module along with the associated covariance as a measurement for position estimation. These FSW-specific exchanges are seen in Figure 2.5 with the blue arrows which represent the pub-sub messaging system calls.

(7) With an updated state estimate, the spacecraft can now control its attitude, position, and velocity.

(8) These updated states are tracked by the `vizInterface` and sent back to *Vizard* for a new sim update in the visualization.

Unity can save images to an external file. This allows for a log of the images that were taken to

be saved for debugging and validation. The `vizInterface` module also saves all the protobuffers from a run to file, this allows for playback capability on every simulation run.

2.3.4 Challenges of Memory Management

As discussed previously, one of the core advantages of this framework is its modularity. Therefore, the data flow between the software nodes and within *Basilisk* must adapt to the modular structure of the simulation.

2.3.4.1 Transferring Image Data

One of the difficulties that arise when passing images between modules is the potentially large amount of data that needs to be copies or formatted. Furthermore, the *Basilisk* messaging system is C-based in order to translate well to FSW algorithms, which limits the available types (interfaces, functors, classes etc.).

In order to mitigate the frequent manipulation of large blocks of data, messages contain the memory address which points to the image instead of the image itself. As the image first enters the astrodynamics simulation through *ZMQ*, the pointer to the image is owned by the *ZMQ* thread. For memory management, *ZMQ* will free the memory at a deterministic but arbitrary time, and so the address must be allocated by *Basilisk* modules for proper internal memory management. Every module that manipulates the image (starting with the `viz_interface`) must therefore own a pointer to the images memory which must be freed after every new call. At every update, the image message is populated with a new pointer to the image that has just been manipulated without copying it.

In order to not require *OpenCV* library be included across the entire *Basilisk* codebase, the image is formatted as a compressed bitmap (portable network graphics or PNG). By transferring a void pointer — which is a pointer that has no associated data type with it — each module to unpack the bitmap, and recast it to any desired format. This also allows for flexibility in the format and future development of the codebase as the void pointer will adapt well to any image type.

Algorithm 1 Image Memory Management

```
1: Initialize newPointer ← NULL
2: Begin Simulation
3: if newPointer != NULL then
4:     free(newPointer)
5:     newPointer ← NULL
6: if ImageMsg is valid then
7:     vector(uchar) buffer
8:     buffer ← ((char*)ImageMsg.pointer,(char*)ImageMsg.pointer
9: +ImageMsg.length))
10:    image ← decode(buffer)
11:    modified ← edit(image)
12:    outBuffer ← encode(modified)
13:    newPointer ← address(outBuffer)
14:    msgOut ← (newPointer, size(outBuffer))
15:    writeMsg(msgOut)
```

In summary, every module that manipulates the image must recast the void pointer to an unsigned character (or uchar) vector, decode it (into a CV Mat in most cases), manipulate it, then encode it and pass the pointer to that memory slot. This is written in pseudo code in Algorithm 1.

2.3.4.2 Manipulating Image Data

Another difficulty that arises when developing image processing modules is the treatment of memory allocation on the stack or heap.

Stack allocation happens on contiguous blocks of memory.[11] The size of stack memory to be allocated is known to the compiler and its variables get memory slots on the stack when the function is called. This memory is then deallocated automatically when the function call is completed. Allocating stack memory is easy, fast, and robust to memory leaks; yet, it only provides a small amount of memory (commonly less than 10 MB) which can easily be overflown when manipulating large amounts of data.

Heap memory is allocated during execution and is done so manually by the programmer. If this memory is not handled well — by being appropriately freed — memory leaks can occur in the program. Heap memory allocation incurs a performance cost because it cannot be optimized in

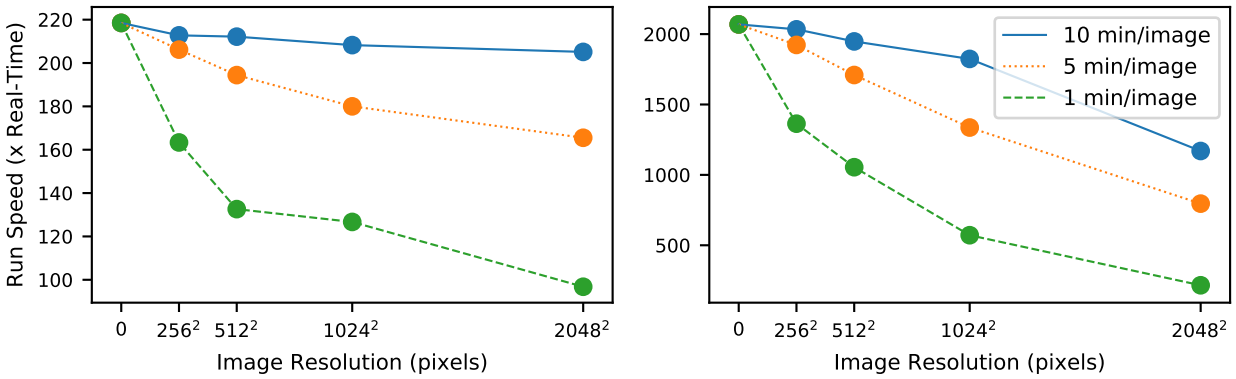
the same way as stack allocation during the code compilation stage. In contrast, heap allocations provide far more memory (limited by virtual and physical memory of the machine).

When manipulating large data sets, notably for image processing, stack versus heap memory allocation can become an issue. This issue has been seen in the course of this work, most notably when computing covariances matrices of OpNav estimates. Discussion on this issue are added throughout the following chapters where applicable.

2.3.5 Closed-loop Simulation and Performances

In the case that closed-loop simulations are to be used for Monte-Carlo analysis and Machine Learning, the software architecture presented must allow for faster-than-real-time speeds. As explained in the above subsection, there are two different modes that can be used, with different performance goals. The speed of both implementations are illustrated by running separate analyses. Figure 2.6 plots simulated time divided by run times (averaging over 5 runs) for varying camera quality and render rates. The x-axis represents the image size and although the ticks read total number of pixels, the scale is linear (square-root of the tick labels) for legibility. The Figure 2.6a shows the results of the lock-step mode, while Figure 2.6b shows the performance-mode results. All tests are run on a MacBook Pro running macOS Version 10.13.6, a 3.5 GHz Intel Core i7 processor with 16GB of memory, and Intel Iris Plus Graphics 650 graphics card.

The simulation used in this section is an OpNav-point scenario developed in the final section of this chapter. This 100min simulation has a 0.5s integration time step, and implements a spacecraft dynamics module alongside FSW algorithms for attitude control running at the same 2Hz. The spacecraft searches for Mars and points to it when able using an MRP-feedback control law derived in Example 8.14 of Reference 184.



(a) Lock-Step Mode

(b) Performance Mode

Figure 2.6: Performance of Both Closed-Loop Implementations

The architecture provides speeds that allow for Monte-Carlo analysis and machine learning scenarios to be run in a reasonable amount of time. The first thing to notice is that performance-mode provides an order of magnitude speed-up relative to its 'Lock-Step'-mode predecessor. This difference shows the main slow-down incurred comes from *Basilisk* needing to wait for *Unity*'s updates. Furthermore, Figure 2.6b shows that if the sim moves forward with minimal communication, the rendering of the image becomes the expensive operation.

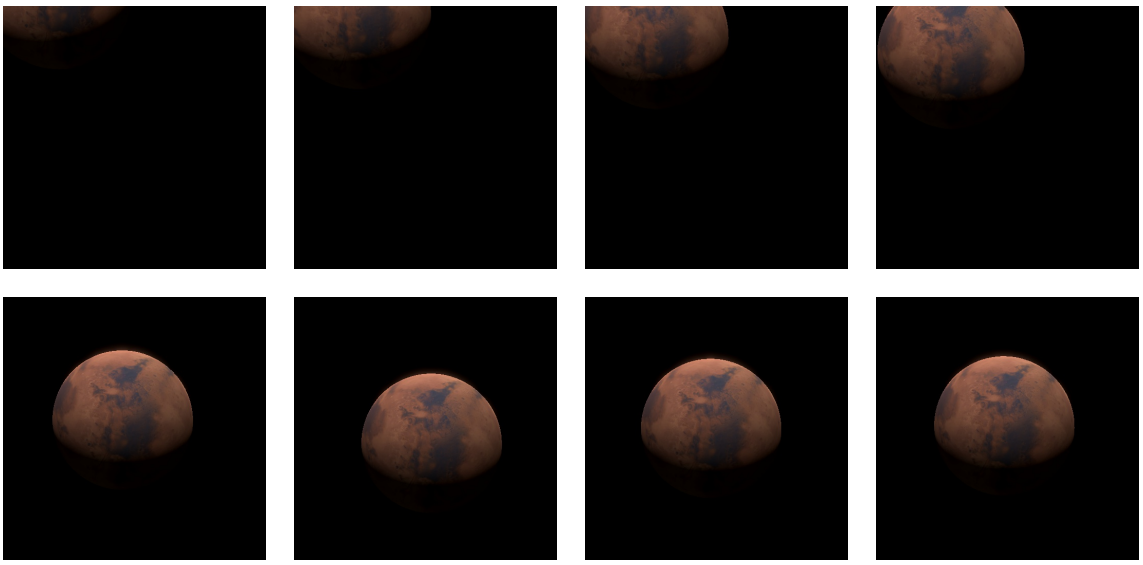


Figure 2.7: Camera View as the Spacecraft Moves the Simulation

Images in Figure 2.7 show the images that are received by the simulation for processing. This is done with a 60s camera render-rate in order to capture the motion of Mars. The planet is not initially visible from the camera's perspective, but the search algorithm brings the planet into the camera frame. In the second half of the run (bottom line), the planet is fully in view of the spacecraft. These images are from the scenario described in the last section of this manuscript, and can import a wide variety of Mars surface maps[1].

2.4 Optical Navigation Example

Optical navigation tracks planet and moon centroids and dimensions to determine the spacecraft location. Several optical navigation methods exist, such as star horizon,[166] centroid and apparent diameter,[41] star occultation,[173] and landmark tracking.[133] Each of these have their specific application scenarios depending on the object they are required to track.

Centroid and apparent diameter measurements find the limb of a body and use the knowledge of its actual size and position. By extracting direction and distance, range and position information is extracted from planet images. The shape of the partially illuminated moon alone permits the estimation of the direction vector to the sun using just a star tracker.[66] With the knowledge of the body in sight, its ephemeris, and its size, determination of both the spacecraft's orbit and attitude can be achieved. These methods require minimal image processing power, are relatively fast to implement, and provide a wealth of extractable information from images. The use of *OpenCV* has helped accelerate module design. For these reasons they will be the baseline methods used in simulating autonomous OpNav.

However, other OpNav methods yield better navigation results. Measurements derived from landmark observations,[133] point distribution methods,[203] or crater detection[170] are some of the many feature tracking methods which provide promising results. The real-time component of this framework creates a realistic environment to quantify and run more computationally extensive algorithms. Future work will include higher-fidelity star-maps in order to do star-horizon detection,[166]

[1] celestiamotherlode.net/catalog/mars.php

amongst other methods that have been described in the literature.

This software framework allows for rapid and high-fidelity testing, and can centralize progress from other fields within astrodynamics. In the aerospace field this has been seen with ORB-SLAM development[155,156] and cross-correlation methods.[29,143] These hold great promise for small body autonomous orbiting and have already proven to be useful on missions such as ESA's Rosetta and ongoing missions such as OSIRIS-REx. Although implementing such methods in *Basilisk* are currently advanced goals, this architecture allows for these additions. *Vizard* allows users to upload shape models for any celestial body as seen in Figure 2.8 with Vesta[1]. This provides the opportunity to train and test shape model reconstruction methods by using fully coupled spacecraft attitude and orbital dynamics.

Centroid tracking and apparent diameter measurements are the baseline OpNav methods in this design. In parallel, developments for feature tracking will be added in along with more image processing capabilities.

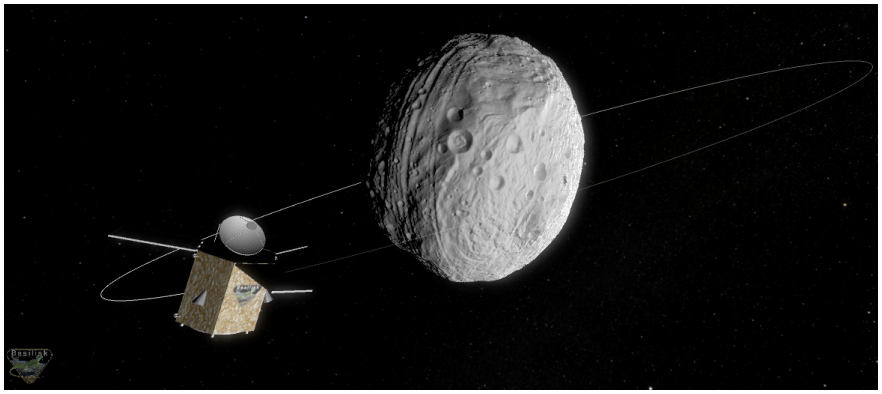


Figure 2.8: Vesta Shape Model Uploaded into *Vizard*

2.4.1 Camera models and validation

In order to realistically model OpNav scenarios, the images generated by the visualization software must be sufficiently accurate for the image processing accuracy: if sub-pixel resolution is

[1] nasa3d.arc.nasa.gov/detail/asteroid-vesta

not used by the algorithms, it should not en enforced to the visualization. *Unity* provides a large set of lighting libraries that can simulate self-shadowing and model lighting on imported shape-models. This allows for the generation of complex lighting scenes. By extension, it allows for the reading in of partially lit planets, showing crescent lighting. This lighting is seen in the visualization in Figure 2.11a.

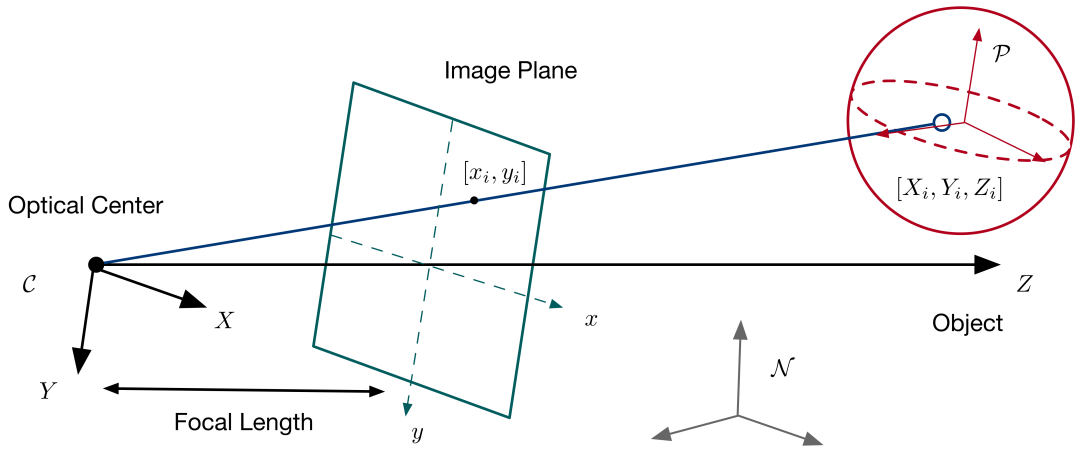
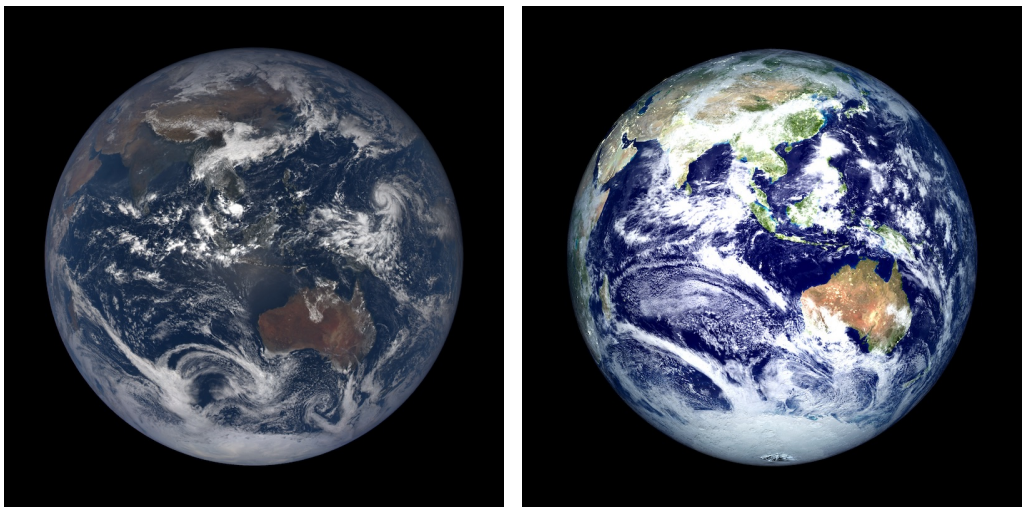


Figure 2.9: Pinhole Camera Model

In order to speed up the simulation as much as possible, *Unity* camera models are used to simulate a realistic camera model which is a pinhole model seen in Figure 4.4. A *Basilisk* camera model can be created as well in order to add more complicated errors. Lens-flaring or lens-distortion[52] can distort images and can be compensated for in post-processing.[196] Though not a method in *Unity 2018*, the modularity of the software package allows for such additions.

Figure 2.10 shows the visualization compared to true data taken by the Epic camera on DSCOVR. On the left is an image taken from Lagrange 1 on October 23rd at 4:35:25 UTC. The image is obtained from the Epic website[1] which also provides the camera specifications. These are provided in Table 2.1, and were used as such in the *Unity* camera model. Sensor size and field of view lock in the focal length, while the resolution and sensor size lock in the pixel size. Therefore all the needed information is provided regarding the camera.

[1] epic.gsfc.nasa.gov/?date=2018-10-23



(a) DSCOVR image (b) Raw Vizard Image

Figure 2.10: Comparing Vizard Images to Real Data

Table 2.1: Epic Camera Parameters

Parameter	Field of View [°]	Resolution [pixels]	Sensor Size [mm]
Value	0.62	2048×2048	[30.72 , 30.72]

Table 2.2: DISCOR Position

Parameter	Earth-S/C [km]	Sun-S/C [km]	Sun-Earth [km]	Sun-Earth-S/C angle [°]
Value	$1,405,708$	$147,451,774$	$148,846,039$	7.28

Regarding the spacecraft position, the source provides distance between DSCOVR and Earth, DSCOVR-Sun distance, Earth-Sun distance, as well as the Sun-Earth-Craft angle. These values are provided in Table 2.2. It is important to note that they do not provide a unique possible position for the spacecraft. The spacecraft therefore lies on a circle off the Sun-Earth direction by 7.28°. Since the exact position is not made public, the simulation placed the spacecraft exactly on the Earth-Sun direction, with the expectation of seeing some differences. The Earth and Sun were placed in the simulation using *SPICE*[1], which provides the Sun and Earth's ephemerides, as well as Earth's rotation in the inertial frame.

[1] naif.jpl.nasa.gov/naif/

Figure 2.10 illustrates that the actual mission image and the synthetic *Vizard* image look very similar as the Earth has the same apparent location and size in the photo, and the continents are lined up correctly as well. Only a slight shift can be seen in the Earth's relative position: Australia seems to be more to the South-West on the real image. This is certainly due to the 7.28° error in the camera's position. Besides this, the only significant differences are seen in the contrast and texture quality. It would seem that the colors seen by epic are more matte. This can be improved and modeled in the visualization as no color sensitivity is modeled, but for the purpose of centroid and apparent diameter, the results are sufficiently accurate.

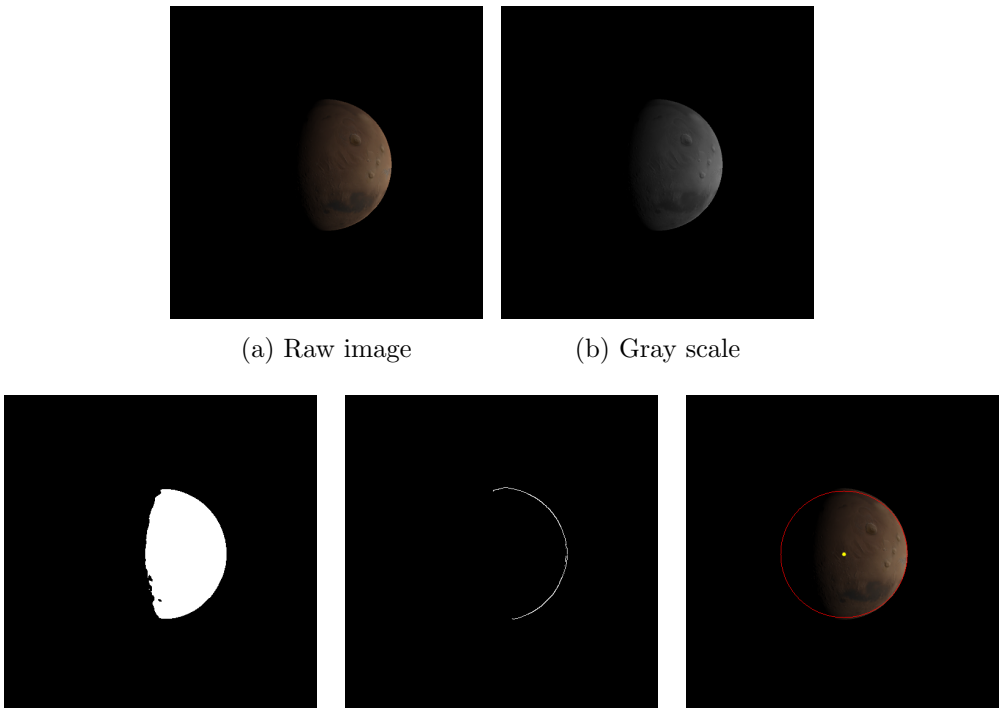
A simple CAD algorithm provides a center point at the pixel coordinates $(1023.71, 1023.03)$ and an apparent radius of 859 pixels for the real data. By running the same algorithm the simulation predicted a planet center at $(1023.50, 1027.68)$, and a radius of 854 pixels. This represents a relative error of 0.46% pixels on the center's position and 0.58% error on the radius. These are relatively small errors given the uncertainty in the spacecraft ephemeris information.

More camera modeling will be detailed in the Chapter 5 regarding autonomy. True cameras do not provide the crisp images that come out of *Unity* and therefore an effort must be dedicated to modeling the noise and artifacts present it common spacecraft imagery.

2.4.2 Image processing methods

The biggest advantage of the software framework presented is its modularity. Certain state-of-the-art limb-fitting algorithms for pose-estimation are on-board capable[43] and can be implemented with *Basilisk* and used in the simulation. This allows for computational speed tests as well as better general performance understanding. In this dissertation, *OpenCV*[1] is chosen as an open-source computer vision library, which saves development time by utilizing a robust software library with widely tested functionality. The transformation used here is a *Hough* transform for circle finding, which exists in many derivative forms.[126,171] Figure 2.11 displays the transformations that an image from the visualization is put through in order to extract apparent diameter and centroid

[1] opencv.org

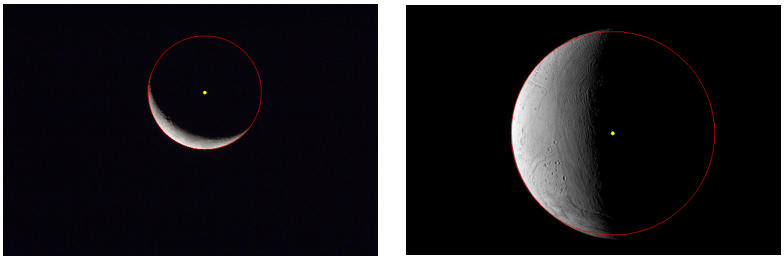


(a) Raw image (b) Gray scale

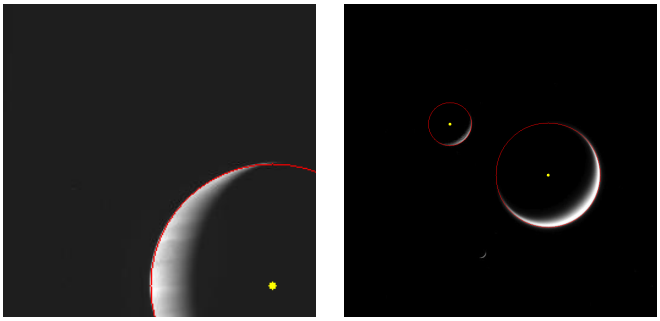
(c) Blur and Threshold (d) Canny edge detection (e) Hough circle

Figure 2.11: Extracting center and apparent diameter from visualization image using *OpenCV*

information. The raw image is turned into grey-scale (Figure 2.11b) before being fed into the HoughCircles function. This function then blurs and thresholds (Figure 2.11c) the image within a call to the Canny edge detection transform (Figure 2.11d). This is then the image used in order to accumulate votes[63] on the possible existing circles in the image. Figure 2.11e shows overall good performance by the algorithm. Other examples using images of the Moon and Enceladus are shown in Figure 2.12. It can be seen in some of the images in Figure 2.12 that although the algorithm is generally quite robust, sometimes the radius of the planet is underestimated. This is seen notably in Figure 2.12b. Image processing imperfections emphasize the necessity to output a measure of uncertainty with the *Hough* transform. More details on the development and use-case for this method are in Chapter 4.



(a) Slim Moon cresent (b) Enceladus



(c) Partial Jupiter (d) Saturn Moons

Figure 2.12: *Hough* circle finding on several real images (Courtesy NASA/JPL-Caltech)

2.4.3 Attitude Guidance and Control Example

In this section, an example scenario is developed in order to illustrate the *Basilisk-Vizard* capabilities. A spacecraft is on orbit around Mars and seeks to align its camera bore-sight with the planet center. It takes images periodically for attitude guidance and control, and uses the Hough algorithm to extract the center and the apparent diameter of the planet being observed. The pixel data is pre-processed before being used to determine the planet direction. Initial conditions for the simulation are given in Tables 5.2-5.3, while simulation and flight software parameters are given in Tables 4.4-2.6. All modules listed are currently available on the *Basilisk* bitbucket repository[1] with additional documentation. The main assumption in these models is that the reference is static, in this case that the planet does not move in the camera frame. This scenario puts this assumption to the test.

The Pixel and Line Transformation module performs the simple transformation from pixel

[1] bitbucket.org/avslab/basilisk

Table 2.3: Spacecraft Initial States

$\boldsymbol{\sigma}_{\mathcal{BN}}$	$\boldsymbol{\omega}_{\mathcal{BN}}$	Orbital Elements $(a, e, i, \Omega, \omega, f)$
$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$18000\text{ km}, 0, 20°, 25°, 190°, 100°$

Table 2.4: Camera Parameters

$\boldsymbol{\sigma}_{\mathcal{CB}}$	${}^{\mathcal{B}}\boldsymbol{r}_C$[m]	Resolution [pixels]	Sensor Size [mm]
$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}0 & 0.2 & 0.2\end{bmatrix}^T$	$\begin{bmatrix}512 & 512\end{bmatrix}^T$	$\begin{bmatrix}10 & 10\end{bmatrix}^T$

Table 2.5: Simulation Parameters

Simulation Modules Instantiated	**Necessary parameters at initialization**
Spacecraft Hub	Inertia $[I] = \text{diag}(900, 800, 600)$ kg·m mass $M = 750$ kg
Gravity Effector/Eclipse	December 12th 2019 at 18:00:00.0 (Z) $\mu_{\text{mars}} = 4.28284 \cdot 10^4 \text{km}^3/\text{s}^2$
Simple Navigation Star Tracker	Attitude error $\sigma_{\text{att}} = 1/3600°$ Rate error $\sigma_{\text{rate}} = 5 \cdot 10^{-5°}/\text{s}$
Reaction Wheel Effector	4 Honeywell HR16 Wheels[1]
Wheel orientations	Elevation $40°$, Azimuths angles $45°, 135°, 225°, 315°$ $\mathcal{B}$-Position [m] $\begin{bmatrix}0.8, 0.8, 1.79070\end{bmatrix}^T$ $\begin{bmatrix}0.8, -0.8, 1.79070\end{bmatrix}^T$ $\begin{bmatrix}-0.8, -0.8, 1.79070\end{bmatrix}^T \begin{bmatrix}-0.8, 0.8, 1.79070\end{bmatrix}^T$

Table 2.6: Flight Software Parameters

Flight Software Modules Instantiated	**Necessary parameters at initialization**
Image Processing (arguments for HoughCirlcle method[2])	param1 = 300, param2 = 20, minDist = 50 minRadius = 20, dp = 1, maxRadius = 409
OpNav Point	minAngle = $0.001°$, timeOut = 100s $\omega_{\text{search}} = [0.06, 0.0, -0.06]°/\text{s}$, ${}^{\mathcal{C}}h_c = [0, 0, 1]\text{m}$
Pixel Line Transform	Planet Target is Mars
MRP Feedback RW	K = 3.5 , P = 30 (no integral feedback)
RW motor Torque	Control axes are ${}^{\mathcal{B}}\begin{bmatrix}\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3\end{bmatrix}$

data to spacecraft relative position. This scenario feeds raw measurements of the planet center (x_c, y_c) in pixels to the guidance module, with the knowledge of the camera parameter. Notably the pixel size is given by $X = \frac{\text{SensorSize}_x}{\text{Resolution}_x}$ and $Y = \frac{\text{SensorSize}_y}{\text{Resolution}_y}$ in mm/pixel.

$$ {}^{\mathcal{C}}\boldsymbol{r}_{BN} = -\begin{bmatrix} \frac{X}{f} \cdot (\text{x}_c - \frac{\text{Resolution}_x}{2} + \frac{1}{2}) & \frac{Y}{f} \cdot (\text{y}_c - \frac{\text{Resolution}_y}{2} + \frac{1}{2}) & 1 \end{bmatrix} \tag{2.1} $$

where ${}^{\mathcal{C}}\boldsymbol{r}_{BN}$ is the relative vector of the camera bore-sight with respect to the celestial center, where the left superscript represents the frame a vector is projected onto. f is the camera field of view, and the transformations on the measurements also re-center the pixels.[13,166,167] Since the measurements in this scenario are given raw to the guidance module and without consideration of covariance, this completes the measurement transformation.

2.4.3.1 OpNav Point Guidance

This simulation specifically uses the OpNav-Point module for guidance. The attitude guidance module has the goal of aligning a commanded camera-fixed spacecraft vector $\hat{\boldsymbol{h}}_c$ with the measurement vector $\boldsymbol{h}$. Here, $\hat{\boldsymbol{h}}_c$ is the camera bore-sight, and so the attitude tracking errors seek to align the camera towards the target and achieve relative pointing.

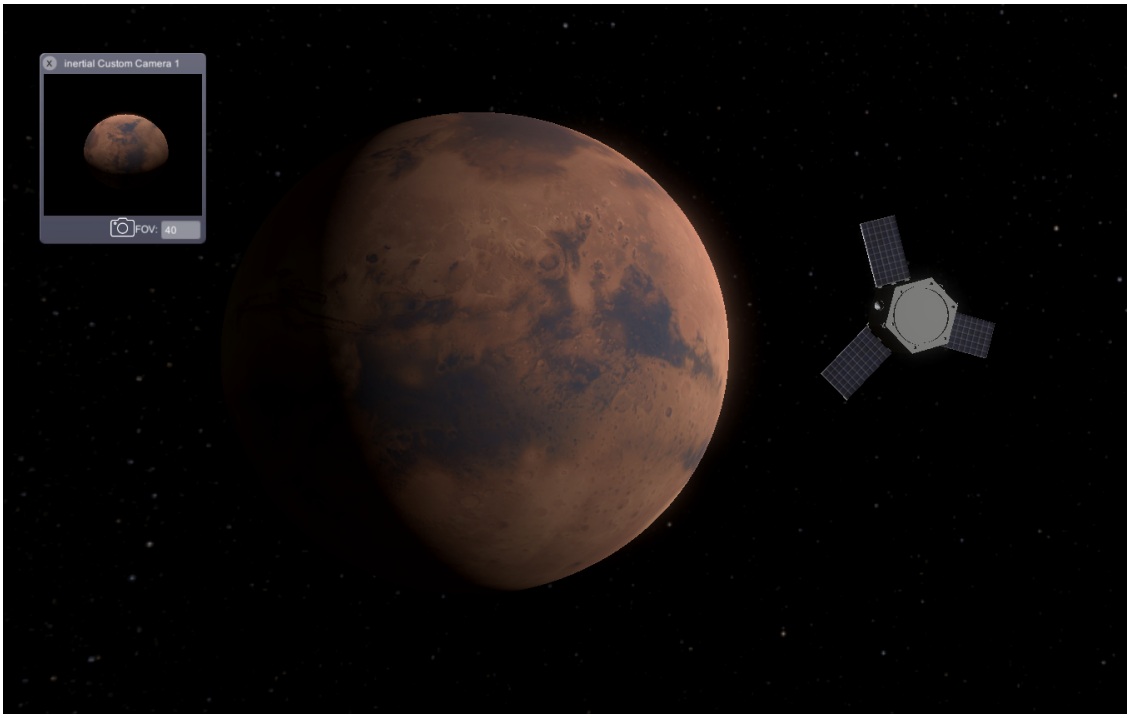


Figure 2.13: OpNav Pointing Scenario

In the following developments, all vectors are assumed to be taken with respect to a camera-fixed frame $\mathcal{C}$ if a frame is not specified. The attitude of the camera relative to the target reference

frame $\mathcal{R}$ is written as a principal rotation from $\mathcal{R}$ to $\mathcal{C}$. The target $\mathcal{R}$ is defined simply by the vector $\hat{\boldsymbol{h}}_c$ and body frame vectors. The body frame is defined as $\mathcal{B}: \{\hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2, \hat{\boldsymbol{b}}_3\}$.

At the start of the simulation, the camera does not have the planet in sight. In this situation, a search rate is requested: ${}^{\mathcal{B}}\boldsymbol{\omega}_{\text{search}}$. Once the planet is found, and measurements are provided, the requested rate is zeroed to keep the spacecraft at rest. The module also has the ability to request a rotation about the camera bore-sight for stability. This module is designed for simplicity and robustness. In order to be independent from a orbit determination solution, the inertial reference frame acceleration $\dot{\boldsymbol{\omega}}_{R/N}$ is set to zero. This means the guidance will need to constantly adjust to a moving reference. Figure 2.13 pictures the spacecraft once Mars has been found and is being tracked.

Similarly to a sun-safe point guidance law, this module does not establish a unique target-pointing reference frame. Rather, it simply aligns $\hat{\boldsymbol{h}}_c$ with $\boldsymbol{h}$, which is an under-determined 2 degree of freedom condition. If these two vectors are nearly collinear, numerical instabilities can occur, hence the `minAngle` variable set by the user.

The associated principal rotation vector $\hat{\boldsymbol{e}}$ and angle Φ between $\hat{\boldsymbol{h}}_c$ and $\boldsymbol{h}$ are

$$\hat{\boldsymbol{e}} = \frac{\boldsymbol{h} \times \hat{\boldsymbol{h}}_c}{|\boldsymbol{h} \times \hat{\boldsymbol{h}}_c|} \qquad \Phi = \arccos\left(\frac{\boldsymbol{h} \cdot \hat{\boldsymbol{h}}_c}{|\boldsymbol{h}|}\right) \tag{2.2}$$

If Φ is less then the module parameter `minAngle`, it is assumed that no valid planet heading vector is available and the attitude tracking error $\boldsymbol{\sigma}_{C/R}$ is set to zero. For valid planet headings, this rotation from $\mathcal{R}$ to $\mathcal{C}$ is written as a set of MRPs through

$$\boldsymbol{\sigma}_{C/R} = \tan\left(\frac{\Phi}{4}\right)\hat{\boldsymbol{e}} \tag{2.3}$$

The set $\boldsymbol{\sigma}_{C/R}$ is the attitude error of the output attitude guidance message. If the spacecraft is to be brought to rest, $\boldsymbol{\omega}_{R/N} = \mathbf{0}$, then the tracking error angular velocity vector is computed using:

$$\boldsymbol{\omega}_{B/R} = \boldsymbol{\omega}_{B/N} - \boldsymbol{\omega}_{R/N} \qquad \dot{\boldsymbol{\omega}}_{R/N} = \mathbf{0} \tag{2.4}$$

The attitude guidance message must specify the inertial reference frame acceleration vector. This

is set to zero and is the assumption that needs to be justified as it can be poorly representative of reality.

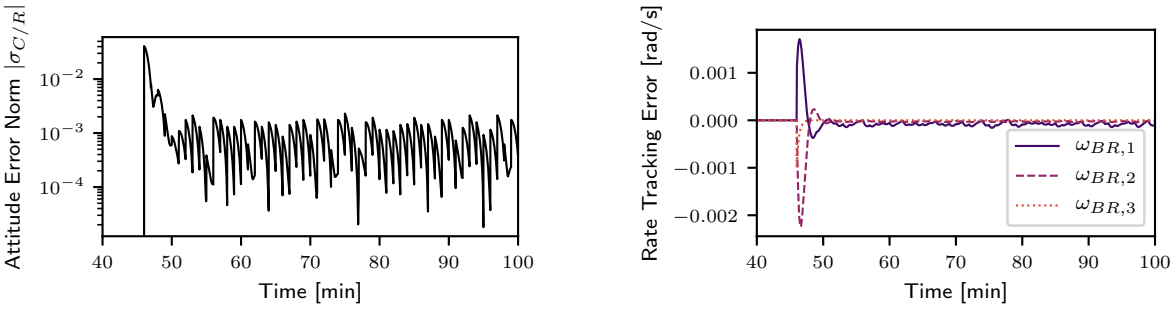
This concludes the module description, which is summarized in the following algorithm. The details of the implementation are currently available on the *Basilisk* open source package in the folder *fswAlgorithms/attGuidance/opNavPoint/*.

Algorithm 2 OpNavPoint

OpNavMeas $\leftarrow$ read(PixelLine)
firstPass $\leftarrow$ True
timeOut $\leftarrow$ False
if OpNavMeas is valid or (!firstPass and !timeOut) **then**
    if OpNavMeas is valid **then**
        ${}^{\mathcal{C}}\hat{\boldsymbol{h}} \leftarrow$ read(OpNavMeas)
        save(${}^{\mathcal{N}}\hat{\boldsymbol{h}}$)
        firstPass $\leftarrow$ False
    else if !firstPass and !timeOut **then**
        ${}^{\mathcal{C}}\hat{\boldsymbol{h}} \leftarrow [\mathcal{CN}]\,{}^{\mathcal{N}}\hat{\boldsymbol{h}}$
    angleError $\leftarrow \arccos({}^{\mathcal{C}}\hat{\boldsymbol{h}}_c \cdot {}^{\mathcal{C}}\hat{\boldsymbol{h}})$
    if angleError $\leq$ minAngle **then**
        $\boldsymbol{\sigma}_{\text{guid}} \leftarrow \mathbf{0}$
    else
        $\hat{\boldsymbol{e}} \leftarrow \text{cross}({}^{\mathcal{C}}\hat{\boldsymbol{h}}, {}^{\mathcal{C}}\hat{\boldsymbol{h}}_c)$
        $\boldsymbol{\sigma}_{\text{guid}} \leftarrow \tan(\frac{1}{4}\text{angleError})\hat{\boldsymbol{e}}$
    $\boldsymbol{\omega}_{\text{guid}} \leftarrow \boldsymbol{\omega}_{\mathcal{BN}} - \boldsymbol{\omega}_{\mathcal{RN}}$
else if Search **then**
    $\boldsymbol{\sigma}_{\text{guid}} \leftarrow \mathbf{0}$
    $\boldsymbol{\omega}_{\text{guid}} \leftarrow \boldsymbol{\omega}_{\text{search}}$

2.4.3.2 OpNav Relative Pointing Results

Running this scenario using the *Basilisk-Vizard* interface shows interesting control results, which test the validity of the assumption stated previously. The spacecraft finds the planet after 40mins of searching with a slow search rate defined in Table 2.6. The assumption of holding the target frame static in the inertial frame holds well with dense measurements. Furthermore, with a fast run-speed, it is easy to test different setups and tailor the simulation to a specific goal or



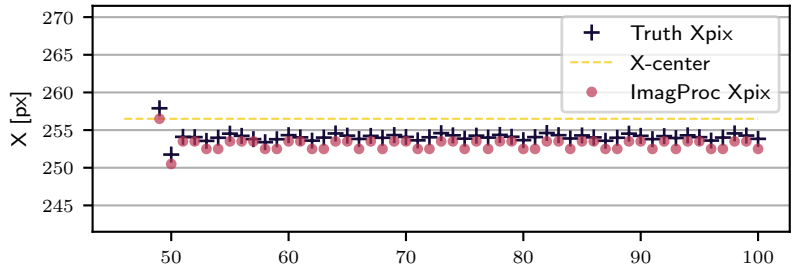
(a) Attitude Error Norm

(b) Rate Tracking Error

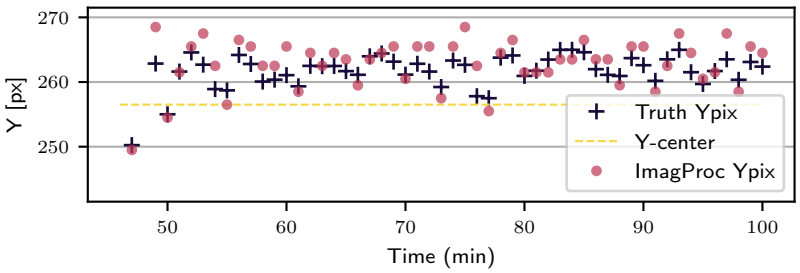
Figure 2.14: Attitude Control Results

requirement. The control plots once the planet is in sight are pictured in Figure 2.14; which shows the attitude error norm and rate tracking error once the planet is found, for a constant 1minute gap between consecutive images.

Although the attitude error in Figure 2.14a shows that the algorithm is constantly surprised by the new images, the error stays close to 10^{-3}. Figure 2.14b shows that the rates mirror this lag with the X and Y components oscillating to control the spacecraft onto the target.



(a) X-pixels



(b) Y-pixels

Figure 2.15: OpNav Pointing Scenario Measured Pixels vs Expected Pixels

Finally, Figure 2.15 shows the true pixels as crosses, alongside the pixels measurement by the HoughCircle transform as dots. It is important to note that this simple transform is performing well as its estimates are very close to expected values, which are computed with the true and noiseless spacecraft attitudes and positions. Furthermore, the measurements show the gap that is seen in Figure 2.14a: each new measurement appears off-center. This is due again to a changing reference frame that the guidance module needs to constantly keep up with. Despite this, the planet stays in frame throughout the control and pixel offsets, as long as they are representative of the truth and do not hinder orbit determination. In a situation akin to the *New Horizons* Pluto fly-by, this simulation can provide a test environment for autonomous pointing algorithms.

2.5 Conclusions

This software architecture provides a testbed for a modern simulation framework to research visual navigation and control applications, including optical navigation and other novel navigation methods. Through the closed loop, coupled interaction between the simulation and the visualization, scenarios provide high-fidelity data at fast rates. Amongst other future endeavors, this architecture opens the door to Machine Learning techniques and Monte Carlo analysis. The open source nature of the project allows for continuous validation from the community, and contributions from developers around the world. Using a simulated camera, optical navigation methods, and the closed loop visualization-simulation interaction, these visual control spacecraft scenarios are tested in a relative pointing scenario. The simulation framework allows a user to make, test, and verify a hypothesis with ease, showcasing the ability for fast and robust analysis.

The tool described in this section is both open-source and novel to the greater visual navigation community. The coming chapters in this dissertation rely heavily on the use of this tool and display the research that can be developed and tested within it.

Chapter 3

Attitude and Body-Rate Determination using Headings to Celestial Bodies

The previous chapter has shown possible attitude guidance using OpNav measurements directly. Although this provides a possible solution to a pointing problem, filtering the measurements allows to fluidly propagate the heading, and to better handle noisy data. Heading determination fits in to the general field of reduced attitude estimation, and has been developed in order to accommodate different data types. The first measurement type is a pyramid of Coarse Sun Sensors (CSS) for sun-heading determination, while the second is a generalized heading measurement which can be produced by OpNav instruments.

Whether pointing a camera towards a celestial body for navigation or a sensor towards the sun, this chapter will describe a novel filter formulation in order to estimate a heading that is inertially fixed. This estimate will then drive the guidance and control modules in order to point a desired instrument at the sun, or any other celestial body.

3.1 Overview

Heading determination algorithms are part of an area of active research, whether it be for attitude or pointing. In the case of CSS sun-heading estimation it is desirable to use only one measurement for heading determination. If the gyros are not sufficiently accurate, as might be the case with low-cost microelectromechanical systems rate gyros, using one measurement allows for a more robust sensor to determine attitude independently.[38,208] Setting aside issues of observability, not using dynamics or rate-gyros allows for minimalist and robust sun-heading estimation. Mass

properties change during the mission, particularly between trajectory correction maneuvers or insertion maneuvers. By being agnostic to mass properties, one filter can provide sun-heading information throughout a mission. This development is then extended to the determination of any heading.

A single inertial heading does not provide fully-observable measurements for sun-heading or body-rate determination: the rotation rate about the heading is geometrically unobservable. This chapter discusses novel improvements in attitude and body-rate determination with the limited information that heading provides. The progress made on the sun-heading estimation front is then generalized to planet centroid measurements from OpNav. Novel developments implemented revolve around avoiding and extracting the non-observable components of the spacecraft state and are discussed in detail. Finally, the measurement generalization is tested in order to understand the validity of assumptions made for inertially fixed targets.

Measurement types can be paired with rate gyros in order to estimate headings and spacecraft rotation rate. Relying solely on one measurement type is advantageous in scenarios where reliance on the fewest number of devices is desired. Here the challenge is to find a robust method for heading determination relying neither on rate gyros nor on spacecraft dynamics. In such a scenario, the rotation rate of the spacecraft is estimated in order to provide state derivative control or simply for better heading estimation. The simple filter inputs and outputs are a driving feature which provides mission longevity and produces a nearly atomic filter which doesn't rely on the accuracy of modules upstream.

A novel sun-heading filter is derived which estimates only the observable components of the body rate vector as the rate about the sun-heading axis remains unobservable. The filter elegantly switches between kinematic formulations to avoid singularities of a single description, and it provides significantly improved sun-heading estimates, as well as a partial body rate estimate. The new filter is compared with two filters for gyro-less sun-heading estimation. One comparison filter uses a projection method to remove the unobservable rate component and another comparison filter uses numerical heading differences to estimate a rotation rate. The filters vary in state vectors,

kinematics, and filter types, yet both have the goal of controlling or removing non-observability.

3.2 Heading Determination for Sun Pointing

Spacecraft pointing is an essential component to any mission scenario, whether it be to point a science instrument towards a target, or a sun sensor towards the Sun. Three-axis attitude estimation has been studied at length[57,127] with the use of a combination of many measurement types.[58,158] Although the star tracker provides a ubiquitous solution for inertial attitude pointing,[37,78] many applications require relative pointing or secondary measurements to ensure the quality of the default instruments. This often leads to the problem of conducting reduced-attitude estimation,[134] which can then lead to guidance and control schemes.[172] This work first develops a novel filter formulation in order to estimate a sun-heading that is inertially fixed, using only Coarse Sun-Sensors as measurements, while also estimating the observable component of the body rate vector.

Coarse sun sensors are small, relatively inexpensive, and regularly used for sunline heading determination. Cosine-type CSS devices output a voltage/current depending on the angle between the sensor normals and the sun direction. Although used in many micro and nano-satellite missions,[9,73] they are also widely used in cis-lunar missions including during safe-mode.[200] More generally, heading determination provides target directions for the use of spacecraft pointing[200] or to solve for attitude.[142] As a 2-degree-of-freedom measurement, one heading does not provide full attitude or rate information on its own. Previous work has efficiently used both rate gyros and CSS measurements[74] for efficient sun-heading determination, notably during periods of eclipse. The gyros help to forward integrate the sun-relative orientation until the spacecraft exits the eclipse. With enough CSSs—traditionally two pyramids of four with large fields of view—a spacecraft can always have at least one activated CSS, and frequently several activated devices. The resulting CSS data is sufficient for sun-heading determination during normal spacecraft operations. Outisde of sun-heading estimation, gyros are often used successfully for attitude and body-rate determination[36,55,104] while compensating for known or estimated drifts[22] and biases.[81] Other work focuses

on single-gimbal moment gyro control,[224] and attitude tracking with unknown gyro bias[148]

In contrast, some attitude determination modes use vector measurements[207] or quaternions[221] without gyros. If the gyros are not sufficiently accurate, as might be the case with low-cost microelectromechanical systems rate gyros, a more robust sensor should determine attitude independently. Setting aside issues of observability, in a safe-mode scenario, it would also reduce the chances of using compromised measurements, and would reduce the additional sensors' associated power draw. Spacecraft dynamics properties have been used to observe the full rate vector[208] through gyroscopic coupling. Yet, not using such dynamics also allows for minimalist and robust estimation. Mass properties change during the mission, particularly between trajectory correction maneuvers or insertion maneuvers. By being agnostic to mass properties and current actuator use one filter can provide sun-heading information throughout a mission. In a safe-mode context, the desire remains to use as little information as possible and to reduce complexity of devices and methods. If any actuators malfunction and their properties are hard-coded in the filter, its state estimation will be compromised because the filter dynamics will be incorrect.

In the absence of rate gyros, it is preferable to estimate spacecraft rate, both for better state estimation and eventually for control. However, the desire to use only CSS measurements for sun-heading determination exposes two observability issues. The first issue is that the spacecraft rotation vector's component about the sun-heading direction is unobservable. In order to use it more reliably in safe-mode, there needs to be progress made on this front: notably by decoupling the unobservable component from the states and eventually observing it through novel methods. Caution must be exercised regarding limited rate estimation using such measurements. Lessons learned from the malfunction and loss of the LEWIS spacecraft[10,107] show that unobservable rate components can build up without the attitude determination algorithm realizing it. If it is desirable to do full rate estimation using CSSs only, the dynamics must be added to couple the unobservable rate through Euler's equation.[38] This does require the use of potentially changing dynamics in the filter, which is undesirable for a minimalist and robust formulation.

The second challenge is specific to CSSs, especially those with small fields of view: the cone

in which each individual sensor can be activated by incoming sunlight. Indeed, the system can suffer from a more general lack of observability; this is due to the nature of CSS measurements,[162] as they only provide angular information between the sensor normal and the sun-heading. This means that one CSS yields a cone of possibilities for the sun direction, two sensors lead to two possibilities, and only with three or more activated sensors is full observability provided instantaneously. If the sensors have a limited field of view, the spacecraft can go through time-spans with little information—not enough to determine the sun-heading uniquely.

Given these two challenges, this work develops a novel kinematic formulation for sun-heading estimation. This formulation decouples the unobservable rate from the state vector. In previous works[38,161] the spacecraft body rate relative to the inertial frame is not estimated by the filter. In order to estimate at least a part of the body rate vector, a frame switching paradigm is implemented in order to avoid singularities, similarly to how Modified Rodrigues Parameters (MRPs) switch between alternate representations.[182] This implies rotating the states and covariance matrix when singularities are approached, and tracking the frames of interest, as well as deriving a mapping of the state noise compensation on the covariance.

After illustrating the observability problem at hand, this section derives five filters and compares their performances. The first filter only estimates the sun-heading vector, and computes a partial solution to the satellite rotation rate at every step using the sun-heading estimates. The second and third subtract the unobservable components out of the states in an Extended Kalman Filter (EKF) and a square-root unscented Kalman Filter (SR-uKF), respectively. In the final formulation, the kinematics of the problem are reduced to a five-by-one vector estimating the sun direction and the observable rotation rate by tracking two different frames: a minimal state vector with no unobservable states and is the main contribution. By switching between two frames, the singularities can be avoided. As a novel derivation, it presents a promising approach to decoupling one of the observability problems in heading filters. This formulation is implemented in an EKF and a SR-UKF, and are referred to as Switch filters.

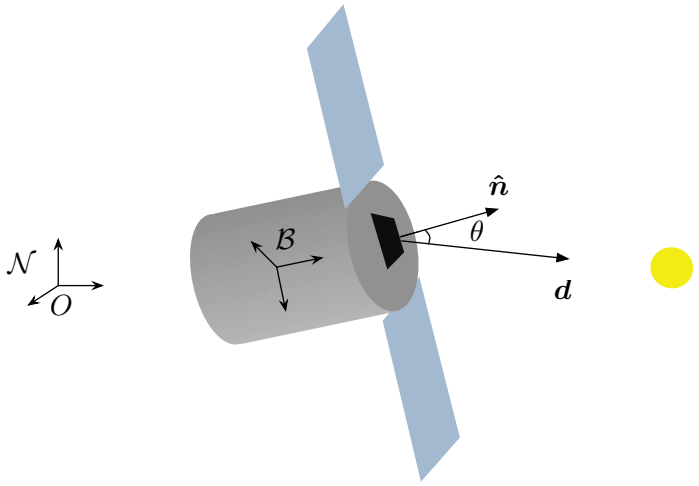


Figure 3.1: Spacecraft Equipped with a CSS

3.2.1 Scenario Description

This study analytically develops five filters, of which two are new sequential sun-heading and rate estimators and compares their performance to the three other gyro-less filter implementations. To compare the filters, a scenario is created where a spacecraft is tumbling in deep space, and attempts to determine its sun-heading direction and rotation rate vector.

The sun-heading vector is estimated as a non-unit vector due to scale factors from the instruments.[163] Indeed, $\boldsymbol{d} = C\hat{\boldsymbol{d}}$ where C is an instrument scale factor determined during ground testing using a calibration flux. It is not desired to estimate C in this work, therefore $\boldsymbol{d}$ is estimated directly. The sun-heading vector in the body frame is written ${}^{\mathcal{B}}\boldsymbol{d}$, its inertial derivative is $\dot{\boldsymbol{d}}$, and its body frame derivative is $\boldsymbol{d}'$. The direction cosine matrix from an arbitrary $\mathcal{S}$ frame into the spacecraft body frame $\mathcal{B}$ will be $[\mathcal{BS}]$, and the inertial frame is labeled $\mathcal{N}$. Finally the rotation rate between two frames is noted $\boldsymbol{\omega}_{\mathcal{BN}}$, and the $[\tilde{\boldsymbol{\omega}}]$ represents the skew-symmetric matrix such that for any $\boldsymbol{a} \in \mathbb{R}^3$, $[\tilde{\boldsymbol{\omega}}]\boldsymbol{a} = \boldsymbol{\omega} \times \boldsymbol{a}$ The filtering notation used complies with Chapter 4 of Reference 202, and the dynamics notation complies with Reference 183.

Table 3.1: CSS Constellation

CSS Group	${}^{\mathcal{B}}\hat{\boldsymbol{n}}_1$	${}^{\mathcal{B}}\hat{\boldsymbol{n}}_2$	${}^{\mathcal{B}}\hat{\boldsymbol{n}}_3$	${}^{\mathcal{B}}\hat{\boldsymbol{n}}_4$
1	$\left[\frac{\sqrt{2}}{2}, -0.5, 0.5\right]^T$	$\left[\frac{\sqrt{2}}{2}, -0.5, -0.5\right]^T$	$\left[\frac{\sqrt{2}}{2}, 0.5, 0.5\right]^T$	$\left[\frac{\sqrt{2}}{2}, 0.5, 0.5\right]^T$
2	$\left[-\frac{\sqrt{2}}{2}, -0.5, 0.5\right]^T$	$\left[-\frac{\sqrt{2}}{2}, -0.5, -0.5\right]^T$	$\left[-\frac{\sqrt{2}}{2}, 0.5, -0.5\right]^T$	$\left[-\frac{\sqrt{2}}{2}, 0.5, 0.5\right]^T$

3.2.2 Observability

One way to quantify the linear observability of a dynamical system is to compute the observability Grammian. The rank of this matrix determines the observability over a specified period of time: if it is full rank, the system is observable, if not, there are unobservable states in the system. In a linear discrete-continuous context, the equation for the Observability Grammian is given in Equation (3.1). In this equation $[\Phi]$ represents the state transition matrix and $[H]_k$ represents the linearized measurement model evaluated at step k, while t_m and t_n are times.

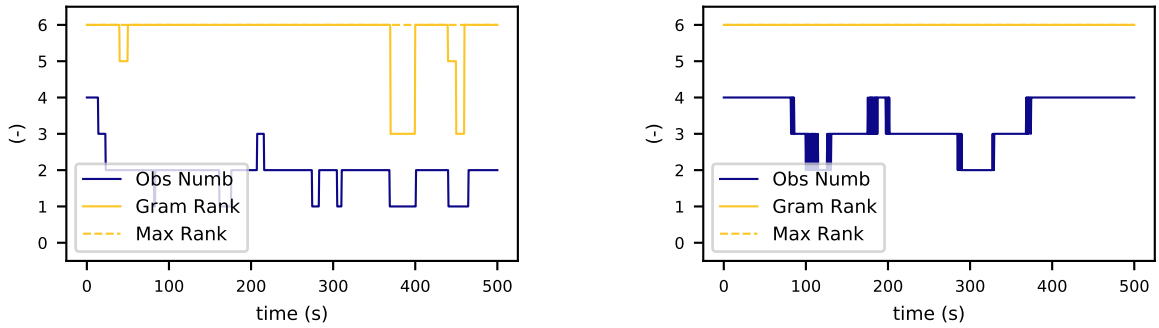
$$\forall (n,m) \in \mathbb{N}, m < n, \quad [G](t_m, t_n) = \sum_{t_k = t_m}^{t_n} [\Phi](t_n, t_m)^T [H]_k^T [H]_k [\Phi](t_k, t_m) \tag{3.1}$$

This equation provides a linear observability analysis. Although non-linear observability analysis[95][180] provides a valid extension, this section attempts to illustrate the problem, not to perform an end-to-end analysis. For this purpose, the state transition matrix and measurement model used are derived in detail in subsection 3.2.4.2 and are shown here:

$$[H] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{n}}_1^T & [0_{1\times(3)}] \\ \vdots & \vdots \\ {}^{\mathcal{B}}\hat{\boldsymbol{n}}_N^T & [0_{1\times(3)}] \end{bmatrix} \tag{3.2}$$

$$[\dot{\Phi}] = [A]\Phi \tag{3.3}$$

The matrix $[A]$ is discussed and defined in Equation 3.11. Throughout this chapter, a double pyramid of four CSS devices each is used. The normals for each of the sensors are displayed in Table 3.1 this allows a maximal sensor coverage. The field of view of each of these sensors will dictate the number of sensors that are activated for a specific attitude.



(a) CSS FOV of 60 °

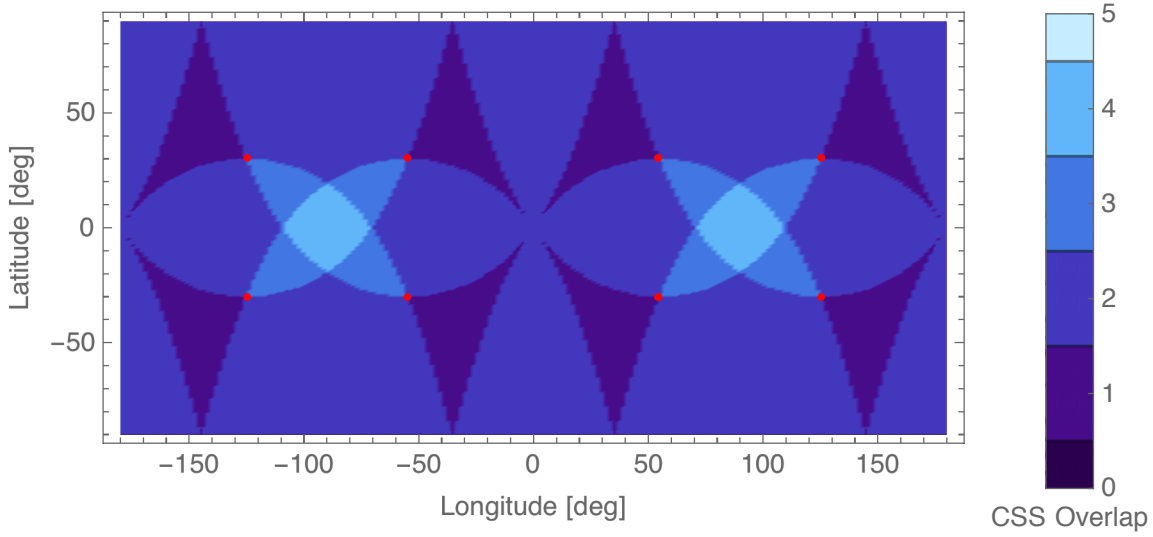
(b) CSS FOV of 85 °

Figure 3.2: Rank of Observability Grammian and Number of Observations

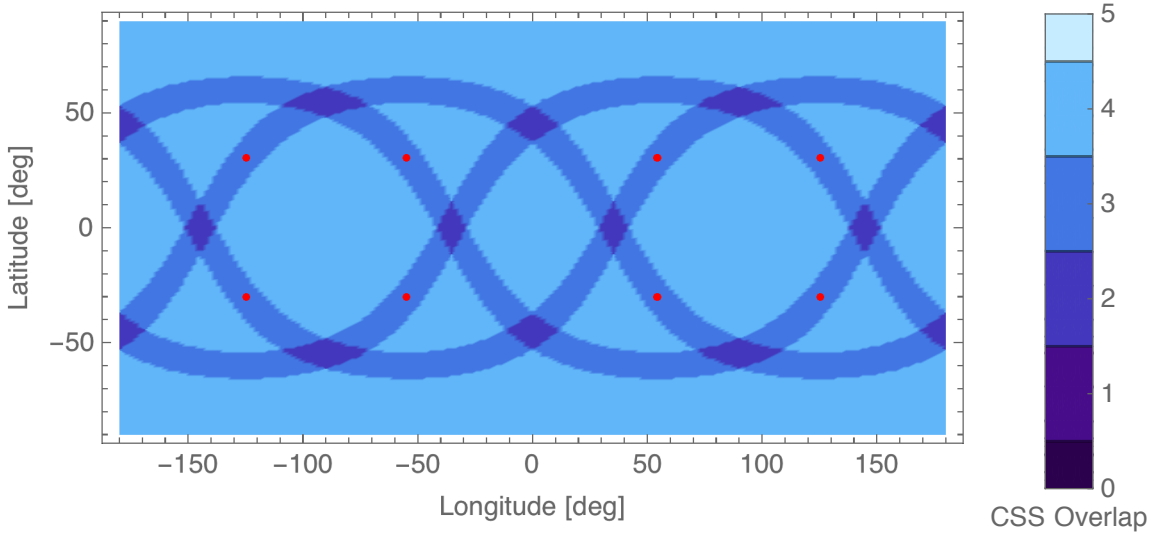
Figure 3.2 shows simultaneously the number of activated sensors, and the observability Grammian defined in Equation (3.1). The term "field of view" is used to describe the half-angle to the cone of visibility for each individual sensor. In the case where the sun sensors have a half-angle field of view of 60°, seen in Figure 3.2a, the Grammian is not always full rank. Because the rank value depends on the number of filter states, it is indicated with the yellow dotted line. For this plot, the sliding window used to compute the Grammian is of 10s, meaning $t_m - t_n = 10s$. Because the measurements are read at 2Hz, this sliding window used for the Grammian contains 20 measurements. This figure shows us that for a tumbling spacecraft there are several periods in which the states are not observable. This is corroborated by the coverage plot in Figure 3.3a.

Nevertheless, fields of view can reach 85° with better quality sun-sensors and the results with this field of view can be seen in Figures 3.2b, 3.3b. This leads to a much higher number of activated sun-sensors at every instant, as seen in Figure 3.2. The observability Grammian is always full rank as the periods with only two activated CSS are brief, which is again corroborated by Figure 3.3b.

It is key to remember that this observability issue occurs in addition to the rate component being unobservable. There are therefore two issues: the sparsity of measurements at times which leads to a partially observable state, and a physically unobservable rate component along the sun-heading direction. Figures 3.2a, 3.2b only speak to the former. The latter is the target of the kinematics derived in this work.



(a) CSS FOV of 60 °



(b) CSS FOV of 85 °

Figure 3.3: CSS Coverage Map Illustrations (CSS Normals are Shown as Red Dots

3.2.3 CSS Measurements

The measurement model is given in Equation (3.4), and the linearized measurement model $[H]$ is defined as $[H] = \left[\frac{\partial \boldsymbol{G}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right]^*$, where $\boldsymbol{X}$ is the corresponding filter state vector, and $\boldsymbol{G}$ is the measurement model. In the following filters, the only measurements used are from the N CSS

devices. For the i^{th} sensor, the measurement is simply given by the dot product of the sunline heading and the normal to the sensor, which defines each of the components of $\boldsymbol{G}$:

$$G_i(\boldsymbol{X}) = \hat{\boldsymbol{n}}_i \cdot \boldsymbol{d} \tag{3.4}$$

In this application, the normals are listed in Table 3.1. This yields the partial derivatives for the $[H]$ matrix:

$$[H] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{n}}_1^T & [0_{1\times(n-3)}] \\ \vdots & \vdots \\ {}^{\mathcal{B}}\hat{\boldsymbol{n}}_N^T & [0_{1\times(n-3)}] \end{bmatrix} \tag{3.5}$$

where the rows contain the transposed normal vectors of the sensors that received measurements. The left-exponent notation indicates the frame with respect to which the vector components are taken. Hence the $[H]$ matrix has a changing size depending on the amount of measurements. Additionally the size of $[H]$ matrix depends on the number of states n as seen in Equation (3.40).

3.2.4 Overview of Comparative Filters Kinematics

There are many possible implementations of gyro-less sun-heading filters.[160,205] This subsection describes the formulations of previously implemented filters. This development sets up the mathematical frame work of CSS filters and illustrates the particular challenges of these solutions. The EKF algorithm used in these developments is explained and derived in Reference 202.

3.2.4.1 Sun-Heading EKF

The sun-heading EKF ('Sun-EKF' in following numerical simulations) is developed to use rate gyro measurements if they are available. In the case in which they are not, the rate is computed with the two previous sun-heading estimates. The state vector of this filter only contains the sunline vector in body frame components: $\boldsymbol{X} = {}^{\mathcal{B}}\boldsymbol{d}$. Given the nature of the filter, there is no unobservable state component as the body frame derivative of $\boldsymbol{d}$ is not estimated. This solution is very simple yet has been shown to provide suitable sun-pointing performance in a safe-mode scenario.[162] This

filter is included in the performance comparison as it provides an interesting benchmark to compare the more sophisticated filters again.

The propagation function $\boldsymbol{F}$ is given in Equation (3.6), and is discretized using an Euler integration in Equation (3.7) with k indicating a time-step indice. This provides a simple and fast integration scheme. As a reminder, the tilde operator in Equation (3.6) is the matrix representation of the cross operator.

$$\boldsymbol{X}' = \boldsymbol{F}(\boldsymbol{X}) = {}^{\mathcal{B}}\boldsymbol{d}' = -[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]{}^{\mathcal{B}}\boldsymbol{d} \tag{3.6}$$

$${}^{\mathcal{B}}\boldsymbol{d}_{k+1} = {}^{\mathcal{B}}\boldsymbol{d}_k - \Delta t\ [\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]{}^{\mathcal{B}}\boldsymbol{d}_k \tag{3.7}$$

Next the state dynamics matrix $[A]$ is found through:

$$[A] = \left[\frac{\partial \boldsymbol{F}(\boldsymbol{d}, t_i)}{\partial \boldsymbol{d}}\right] = -\left[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}\right] \tag{3.8}$$

Gyro measurements are not being read by the filter but can be approximated[38,161] by logging an extra time step of the sun-heading vector estimate $\boldsymbol{d}$.

$$\boldsymbol{\omega}_k = \frac{1}{\Delta t}\frac{\boldsymbol{d}_k \times \boldsymbol{d}_{k-1}}{\|\boldsymbol{d}_k \times \boldsymbol{d}_{k-1}\|}\arccos\left(\frac{\boldsymbol{d}_k \cdot \boldsymbol{d}_{k-1}}{\|\boldsymbol{d}_k\|\|\boldsymbol{d}_{k-1}\|}\right) \tag{3.9}$$

Equation (3.9) uses the shorthand notation $\boldsymbol{\omega}$ to signify $\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$. Aliasing or noise issues are inherent to such a formulation. If the measurement times are too far apart with regard to the rate of change of the system, the rate may be poorly represented. On the other hand, if measurements are very close in time, the two vectors that are being crossed are nearly co-linear. This will lead to noise being amplified and an incorrect estimate of $\boldsymbol{\omega}$. This method is not expected to produce good estimates for rate, yet it has still be used successfully for attitude control in the literature.[160,161] It is therefore presented as a method of comparison.

3.2.4.2 Subtracting unobservability

The second filter derivation (called 'EKF' and 'SR-uKF' in the following numerical simulations) solves the rate unobservability by subtracting, from the state, the rate component along the

sun-heading axis. The states that are estimated in this filter are the sunline vector, and its rate of change in the body frame $\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d} & {}^{\mathcal{B}}\boldsymbol{d}' \end{bmatrix}^T$.

The dynamics are given in Equation (3.10). Given the nature of the filter, the rotation about the $\boldsymbol{d}$ axis remains unobservable. In order to remedy this, the states are projected along this axis and subtracted, in order to measure only observable state components. This is seen in the subtraction of $(\boldsymbol{d} \cdot \boldsymbol{d}') \cdot \boldsymbol{d}$ after normalization in both heading and heading-rate terms.

$$\boldsymbol{X}' = \boldsymbol{F}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{F}_1(\boldsymbol{d}) \\ \boldsymbol{F}_2(\boldsymbol{d}') \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}' - (\boldsymbol{d} \cdot \boldsymbol{d}')\frac{\boldsymbol{d}}{||\boldsymbol{d}||^2} \\ -2(\boldsymbol{d} \cdot \boldsymbol{d}')\frac{\boldsymbol{d}'}{||\boldsymbol{d}||^2} \end{bmatrix} \tag{3.10}$$

Using $[I]$ as a standard notation for the identity matrix, the associated state dynamics matrix $[A]$ is found through:

$$[A] = \begin{bmatrix} \frac{\partial \boldsymbol{F}_1(\boldsymbol{X},t_i)}{\partial \boldsymbol{d}} & \frac{\partial \boldsymbol{F}_1(\boldsymbol{X},t_i)}{\partial \boldsymbol{d}'} \\ \frac{\partial \boldsymbol{F}_2(\boldsymbol{X},t_i)}{\partial \boldsymbol{d}} & \frac{\partial \boldsymbol{F}_2(\boldsymbol{X},t_i)}{\partial \boldsymbol{d}'} \end{bmatrix} \tag{3.11}$$

$$= \begin{bmatrix} -\left(\frac{\boldsymbol{d}'\boldsymbol{d}^T}{||\boldsymbol{d}||^2} + (\boldsymbol{d} \cdot \boldsymbol{d}')\frac{||\boldsymbol{d}||^2[I] - 2\boldsymbol{d}\boldsymbol{d}^T}{||\boldsymbol{d}||^4}\right) & [I] - \frac{\boldsymbol{d}\boldsymbol{d}^T}{||\boldsymbol{d}||^2} \\ -2\left(\frac{\boldsymbol{d}'\boldsymbol{d}'^T}{||\boldsymbol{d}||^2} - 2(\boldsymbol{d} \cdot \boldsymbol{d}')\frac{\boldsymbol{d}'\boldsymbol{d}^T}{||\boldsymbol{d}||^4}\right) & -2\frac{\boldsymbol{d}\boldsymbol{d}'^T - (\boldsymbol{d}\boldsymbol{d}')[I]}{||\boldsymbol{d}||^2} \end{bmatrix} \tag{3.12}$$

In order to implement another type of filter for state-estimation comparison, a square-root unsecented Kalman Filter is implemented using the same formulation. The implementation of this filter is denoted as EKF or SR-uKF according to the algorithm used. The SR-uKF has no need for partial derivative calculation which simplifies the code development to implement seen in Equation (3.11), and is used routinely for attitude determination.[56] The uKF uses $\alpha = 0.02$ as a constant determining the spread of the sigma points.[210] The prior knowledge of the probability distribution of the state is set with $\beta = 2$ (which is optimal for Gaussian distributions).

The challenge with this filter is that the algorithms creates sun-heading rate $\boldsymbol{d}'$ estimates at first assuming it is fully observable, then uses a projection to force the unobservable velocity component to be zero. Of interest is a filter that directly addresses this partial observability, and see how this filter peforms relative to these earlier filters.

3.3 Switch Filter Formulation

This section derives the new switch-filter formulation (labeled 'Switch-EKF' and 'Switch-SRuKF' in the numerical simulations). This novel kinematic formulation utilizes the ability to switch between two frames to avoid singularities of the heading vector parameterization.

3.3.1 Frame Definitions

The switching filter attempts to avoid subtracting any terms from the estimate rate vector, while still enforcing the unobservable rate component is zero. In order to do this, an appropriate sensor frame $\mathcal{S} : \{\hat{\boldsymbol{s}}_1, \hat{\boldsymbol{s}}_2, \hat{\boldsymbol{s}}_3\}$ must be defined as pictured in Figure 3.4 alongside the body frame $\mathcal{B} : \{\hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2, \hat{\boldsymbol{b}}_3\}$.

In order to not track the rate component alongside the sunline direction, a frame is defined such that the sunline direction is one of the basis vectors. Without loss of generality the sun-heading measurement direction $\boldsymbol{d}$ is chosen to be the aligned with the first base vector $\hat{\boldsymbol{s}}_1$

$$\hat{\boldsymbol{s}}_1 = \frac{\boldsymbol{d}}{|\boldsymbol{d}|} \tag{3.13}$$

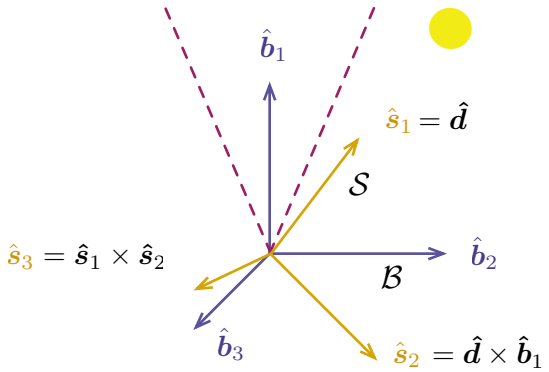


Figure 3.4: Frame Built off the Body Frame for Switch Filters

Thus the rate component about $\hat{\boldsymbol{s}}_1$ is unobservable. The second and third $\mathcal{S}$-frame base vector are arbitrary as any choice keeps the unobservable rate component along $\hat{\boldsymbol{s}}_1$. A simple choice is to

define

$$\hat{\boldsymbol{s}}_2 = \frac{\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_1}{|\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_1|} \tag{3.14}$$

$$\hat{\boldsymbol{s}}_3 = \hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{s}}_2 \tag{3.15}$$

The problem that arises is the singularity that occurs when $\hat{\boldsymbol{b}}_1$ and $\boldsymbol{d}$ are aligned as this switch frame $\mathcal{S}$ is then undefined. To avoid this singularity an alternate sensor frame $\bar{\mathcal{S}}$ is defined which also has the first base vector aligned with the sun-heading direction $\boldsymbol{d}$. This approach is similar in spirit to how the QUEST attitude estimation algorithm[193] avoids a singularity by switching between two kinematic descriptions, or how the Modified Rodrigues Parameters switch between two alternate attitude representations.[182,183] The underlying idea being that when approaching an ill-defined frame $\mathcal{S}$ definition, a second frame $\bar{\mathcal{S}}$ is used. This frame $\bar{\mathcal{S}} = \{\hat{\bar{\boldsymbol{s}}}_1 = \hat{\boldsymbol{s}}_1, \hat{\bar{\boldsymbol{s}}}_2, \hat{\bar{\boldsymbol{s}}}_3\}$ cannot be singular at the same time as $\mathcal{S}$: it uses the same first vector, but constructs $\hat{\bar{\boldsymbol{s}}}_2$ using $\hat{\boldsymbol{b}}_2$ of the body frame. The last vector, once again, completes the orthonormal frame.

$$\hat{\bar{\boldsymbol{s}}}_2 = \frac{\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_2}{|\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_2|} \tag{3.16}$$

$$\hat{\bar{\boldsymbol{s}}}_3 = \hat{\bar{\boldsymbol{s}}}_1 \times \hat{\bar{\boldsymbol{s}}}_2 \tag{3.17}$$

By switching between the $\mathcal{S}$ and $\bar{\mathcal{S}}$ frames the kinematic singularities are always avoided. The perpendicularity of $\hat{\boldsymbol{b}}_1$ and $\hat{\boldsymbol{b}}_2$ results in either $\mathcal{S}$ or $\bar{\mathcal{S}}$ being nonsingular at all times. For example, whenever the sunline $\boldsymbol{d}$ gets within a cone of 30° of $\hat{\boldsymbol{b}}_1$, the frame is switched to $\bar{\mathcal{S}}$ which is not singular since the body vectors are orthonormal. Similarly, when $\boldsymbol{d}$ approaches $\hat{\boldsymbol{b}}_2$ the frame is switched back to $\mathcal{S}$.

Because the two frames share the sunline vector $\boldsymbol{d}$, both frames have the unobservable rate component along the first axis. Further, the sun-heading to be estimate is the same first base vector. The vector components are mapped between the body frame $\mathcal{B}$ and the two sensor frames

$\mathcal{S}$ and $\bar{\mathcal{S}}$ using the following Direction Cosine Matrices or DCMs:

$$[\mathcal{BS}] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{s}}_1 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_2 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_3 \end{bmatrix} \tag{3.18a}$$

$$[\mathcal{B}\bar{\mathcal{S}}] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_1 & {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_2 & {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_3 \end{bmatrix} \tag{3.18b}$$

$$[\bar{\mathcal{S}}\mathcal{S}] = [\mathcal{B}\bar{\mathcal{S}}]^T[\mathcal{BS}] \tag{3.18c}$$

Given a sun-heading vector estimate $\boldsymbol{d}$ all these base vectors are know at any given time.

3.3.2 Filter Kinematics

The body rate relative to the inertial frame is projected onto the $\mathcal{B}$ frame and the $\mathcal{S}$ frame

$$\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} = \omega_1\hat{\boldsymbol{b}}_1 + \omega_2\hat{\boldsymbol{b}}_2 + \omega_3\hat{\boldsymbol{b}}_3 \tag{3.19}$$

$$= \omega_{s,1}\hat{\boldsymbol{s}}_1 + \omega_{s,2}\hat{\boldsymbol{s}}_2 + \omega_{s,3}\hat{\boldsymbol{s}}_3 \tag{3.20}$$

The rates of $\mathcal{S}$ relative to the body and inertial frame are related as such: $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} - \boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$. The first vector of $\mathcal{S}$ is the sun-heading which is considered to be constant in the inertial frame over the period of time required for heading determination. Hence, the only component of $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}}$ that can vary is the rate about the sun-heading: ${}^{\mathcal{S}}\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} = {}^{\mathcal{S}}\begin{bmatrix} \boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} \cdot \hat{\boldsymbol{d}} & 0 & 0 \end{bmatrix}^T$ Since the sunline rotation is impossible to extract from CSS measurements, the spacecraft rotation about the sun-heading axis is set to zero by the filter. This rate component is fundamentally unobservable and it is therefore set to zero.

In Equation (3.20), the previous statement leads to $\omega_{s,1} = 0$. It's important to note that no spacecraft rotation assumption is made, rather the filter zeros the component that it can not observe geometrically. The body rate vector with the previous assumption is defined as follows:

$$\boldsymbol{\omega}^* = \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}(\omega_{s,1} = 0) = \omega_{s,2}\hat{\boldsymbol{s}}_2 + \omega_{s,3}\hat{\boldsymbol{s}}_3 \tag{3.21}$$

$${}^{\mathcal{S}}\boldsymbol{\omega}^* = {}^{\mathcal{S}}\begin{bmatrix} 0 & \omega_{s,2} & \omega_{s,3} \end{bmatrix}^T \tag{3.22}$$

Zeroing this term prevents all motion of the $\mathcal{S}$ frame relative to the inertial frame, as it was the only possible motion given $\dot{\boldsymbol{d}} = \boldsymbol{0}$. Hence, as far as the filter can see, Equation 3.22 leads to $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} = \boldsymbol{0}$,

and the rate relationship becomes $-\boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = \boldsymbol{\omega}^*$. No spacecraft rotation assumption is made, but the kinematics of the filter are simplified given the constraints of observability. In summary:

- Since $\dot{\boldsymbol{d}} = \boldsymbol{0}$ and $\hat{\boldsymbol{d}}$ is the first component of $\mathcal{S}$, $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}}$ can only rotate about $\hat{\boldsymbol{d}}$.
- This rotation about $\hat{\boldsymbol{d}}$ is precisely the rotation that can not be observed by the spacecraft. Without any possible knowledge of this motion, the switch filters set this motion to zero in the kinematics.
- In the filter kinematics, this zeros $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}}$ and $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} - \boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$ therefore becomes $-\boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = \boldsymbol{\omega}^*$, where $\boldsymbol{\omega}^*$ also zeros the rate about the sun-heading.

This filter feature that the unobservable rate component is set to zero is ideal for the sun-pointing control application as the control solution should not try to control any rates about the sun-axis.

The filter state is therefore $\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d} & \omega_{s,2} & \omega_{s,3} \end{bmatrix}^T$ and the kinematics are given by

$$\boldsymbol{X}' = \boldsymbol{F}(\boldsymbol{X}) = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d}' \\ \dot{\omega}_{s,2} \\ \dot{\omega}_{s,3} \end{bmatrix} = \begin{bmatrix} -{}^{\mathcal{B}}\boldsymbol{\omega}^* \times {}^{\mathcal{B}}\boldsymbol{d} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -[\mathcal{BS}] \begin{bmatrix} 0 \\ \omega_{s,2} \\ \omega_{s,3} \end{bmatrix}^{\mathcal{S}} \times {}^{\mathcal{B}}\boldsymbol{d} \\ 0 \\ 0 \end{bmatrix} \tag{3.23}$$

$$[A] = \left[\frac{\partial \boldsymbol{F}(\boldsymbol{d}, t_i)}{\partial \boldsymbol{X}}\right] = \begin{bmatrix} -[{}^{\mathcal{B}}\tilde{\boldsymbol{\omega}}^*] & -[\tilde{\boldsymbol{d}}] \begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{s}}_2 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_3 \end{bmatrix} \\ [0]_{2\times 3} & [0]_{2\times 2} \end{bmatrix} \tag{3.24}$$

The 3×2 matrix in the dynamics matrix corresponds to the truncated DCM $[\mathcal{BS}]$, and $\dot{\omega}$ is the time derivative of the scalar component ω. Equation 3.42 shows the zeroed sunline rotation component in the filter kinematics.

This formulation leads to simple kinematics, much simpler than those of the filter which subtracts the unobservable states, yet can actually estimate the two observable vector components

of the rate, instead of using past estimates of $\boldsymbol{d}$. In regard to the SR-uKF version of this filter, the same coefficients are used: $\alpha = 0.02$, and $\beta = 2$.

3.3.3 Switching Frames

The challenge that comes with the novelty of using two frames for the kinematics is switching between them. The new states $\bar{\boldsymbol{X}}$ and covariance $[\bar{P}]$ after the switch are

$$\bar{\boldsymbol{X}} = [W]\boldsymbol{X} \qquad\qquad [\bar{P}] = [W][P][W]^T \tag{3.25}$$

Where $\boldsymbol{X}$ and $[P]$ represent the state and covariance in the $\bar{\mathcal{S}}$ frame. The $[W]$ matrix maps the rate components from the $\mathcal{S}$ frame to the $\bar{\mathcal{S}}$ frame when a switch occurs. The matrix $[W]$ is computed with

$$[W] = \begin{bmatrix} [I]_{3\times3} & [0]_{3\times2} \\ [0]_{2\times3} & \begin{bmatrix} \hat{\bar{\boldsymbol{s}}}_2 \cdot \hat{\boldsymbol{s}}_2 & \hat{\bar{\boldsymbol{s}}}_2 \cdot \hat{\boldsymbol{s}}_3 \\ \hat{\bar{\boldsymbol{s}}}_3 \cdot \hat{\boldsymbol{s}}_2 & \hat{\bar{\boldsymbol{s}}}_3 \cdot \hat{\boldsymbol{s}}_3 \end{bmatrix} \end{bmatrix} \tag{3.26}$$

using the previously computed $\mathcal{S}$ and $\bar{\mathcal{S}}$ frame base vectors. The sun-heading vector $\boldsymbol{d}$ is unmodified, while the rates are rotated into the switched frame. This equation assumes the switch is going from the $\mathcal{S}$ frame to $\bar{\mathcal{S}}$ (the reciprocal is equivalent), and

$$\begin{bmatrix} \hat{\bar{\boldsymbol{s}}}_2 \cdot \hat{\boldsymbol{s}}_2 & \hat{\bar{\boldsymbol{s}}}_2 \cdot \hat{\boldsymbol{s}}_3 \\ \hat{\bar{\boldsymbol{s}}}_3 \cdot \hat{\boldsymbol{s}}_2 & \hat{\bar{\boldsymbol{s}}}_3 \cdot \hat{\boldsymbol{s}}_3 \end{bmatrix} \tag{3.27}$$

corresponds to the bottom left 2×2 submatrix of $[\bar{\mathcal{S}}\mathcal{S}]$. Equation (3.26) therefore provides a first order frame change for the covariance, allowing for the filter to continue its state estimation nominally.

3.3.4 Process Noise for Switch-EKF

Another nuance that arises when writing EKFs is the process noise formula. This is addressed by deriving the $[\Gamma]$ matrix that transports the noise to the state space given the new state vector. The time update of the error covariance matrix $[P]$ from time t_k to t_{k+1} ($\Delta t = t_{k+1} - t_k$) is

given in Equation (3.28).[202] The process noise matrix $[Q]$ is added via the $[\Gamma]$ matrix defined in Equation (3.29).[202] The $[B]$ matrix seen in the integral maps the process noise only on the accelerations, meaning that $[B] = \begin{bmatrix} [0]_{3\times3} \\ [I]_{3\times3} \end{bmatrix}$ when there are 6 states.

$$[P]_{k+1} = [\Phi](t_{k+1}, t_k)[P]_k[\Phi](t_{k+1}, t_k)^T + [\Gamma](t_{k+1}, t_k)[Q][\Gamma](t_{k+1}, t_k)^T \tag{3.28}$$

$$[\Gamma](t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} [\Phi](t_{k+1}, \tau)[B](\tau)\mathrm{d}\tau \tag{3.29}$$

In the earlier filters (the EKF and the SR-uKF), the second half of the state vector is a direct derivative of the sun-heading vector. Regarding state noise compensation, this allows the approximation in Equation (3.30), along with the fact that measurements are received frequently with regard to the evolution of the dynamics.

$$[\Gamma](t_{k+1}, t_k) = \Delta t \begin{bmatrix} \frac{\Delta t}{2}[I]_{3\times3} \\ [I]_{3\times3} \end{bmatrix} \tag{3.30}$$

This does not hold for the switch filter as $[\Phi]$ is a 5 by 5 matrix, hence the development of the following section. In order to simplify the notation of partials in this section, $\bar{\boldsymbol{\omega}}$ will represent the 2×1 matrix $\begin{bmatrix} \omega_{s,2} & \omega_{s,3} \end{bmatrix}^T$

$$[\Phi](t_{k+1}, \tau) = \begin{bmatrix} [\Phi_1]_{3\times3} & [\Phi_2]_{3\times2} \\ [\Phi_3]_{2\times3} & [\Phi_4]_{2\times2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \boldsymbol{d}(t_{k+1})}{\partial \boldsymbol{d}(\tau)} & \frac{\partial \boldsymbol{d}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)} \\ \frac{\partial \bar{\boldsymbol{\omega}}(t_{k+1})}{\partial \boldsymbol{d}(\tau)} & \frac{\partial \bar{\boldsymbol{\omega}}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)} \end{bmatrix} \tag{3.31}$$

Equation (3.31) uses the fact that $[\Phi](t_{k+1}, \tau) = \frac{\partial \boldsymbol{X}(t_{k+1})}{\partial \boldsymbol{X}(\tau)}$, and that $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{d} & \bar{\boldsymbol{\omega}} \end{bmatrix}^T$. With this, Equation (3.30) can be re-written as Equation (3.32).

$$[\Gamma](t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} \begin{bmatrix} [\Phi_1]_{3\times3} & [\Phi_2]_{3\times2} \\ [\Phi_3]_{2\times3} & [\Phi_4]_{2\times2} \end{bmatrix} \begin{bmatrix} [0]_{3\times3} \\ [I]_{3\times3} \end{bmatrix} \mathrm{d}\tau = \int_{t_k}^{t_{k+1}} \begin{bmatrix} [\Phi_2]_{3\times2} \\ [\Phi_4]_{2\times2} \end{bmatrix} \mathrm{d}\tau \tag{3.32}$$

These submatrices of the state transition matrix now need to be approximated. As before, assuming dense tracking data, $[\Phi_4]_{2\times2} = \frac{\partial \bar{\boldsymbol{\omega}}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)} \approx [I]_{2\times2}$. Then, by defining the $[J]$ matrix as

$$[J] = \begin{bmatrix} [0]_{1\times2} \\ [I]_{2\times2} \end{bmatrix} \tag{3.33}$$

Table 3.2: Simulation Parameters

Parameter	$\boldsymbol{\sigma}(t_0)$	$\boldsymbol{\omega}(t_0)$ (°/s)	$[I]$ (kg/m^2)	Mass (kg)	simulation time (s)
Value	$[0,0,0]^T$	$[0.5,-0.5,-1]^T$	diag(900,800,600)	750	500

The rate notations are reconciled through ${}^{\mathcal{S}}\boldsymbol{\omega}^* = [J]\bar{\boldsymbol{\omega}}$. Without specifying a frame, the propagation function yields

$$\boldsymbol{d}_{k+1} - \boldsymbol{d}_\tau = (t_{k+1} - \tau)[\tilde{\boldsymbol{d}_\tau}]\boldsymbol{\omega}^* \tag{3.34}$$

By then moving into the body frame,

$$\frac{\partial^{\mathcal{B}}\boldsymbol{d}(t_{k+1})}{\partial\bar{\boldsymbol{\omega}}(\tau)} = (t_{k+1} - \tau)[{}^{\mathcal{B}}\tilde{\boldsymbol{d}_\tau}][\mathcal{BS}][J] \tag{3.35}$$

$$[\Phi_2]_{3\times 2} = (t_{k+1} - \tau)[{}^{\mathcal{B}}\tilde{\boldsymbol{d}_\tau}]\begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{s}}_2 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_3 \end{bmatrix} \tag{3.36}$$

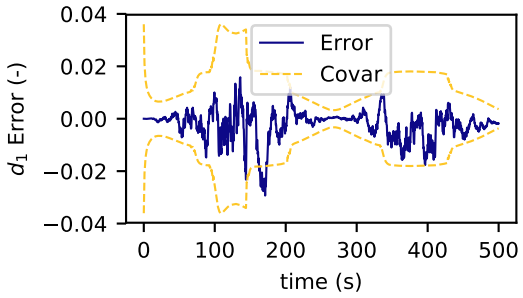
Therefore, assuming the state does not vary over the time between two updates, $[\Phi_2]_{3\times 2}$ can be integrated to approximate $[\Gamma]$.

$$[\Gamma](t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} \begin{bmatrix} [\Phi_2]_{3\times 2} \\ [\Phi_4]_{2\times 2} \end{bmatrix} \mathrm{d}\tau = \Delta t \begin{bmatrix} \frac{\Delta t}{2}[{}^{\mathcal{B}}\tilde{\boldsymbol{d}}_k]\begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{s}}_2 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_3 \end{bmatrix} \\ [I]_{2\times 2} \end{bmatrix} \tag{3.37}$$

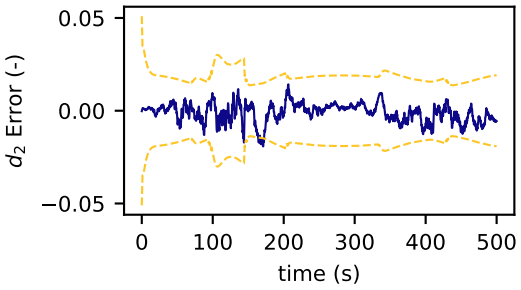
This leads to the new $[\Gamma]$ matrix in Equation (3.37), which is used for state noise compensation.

3.4 Sun-Heading Estimation Results

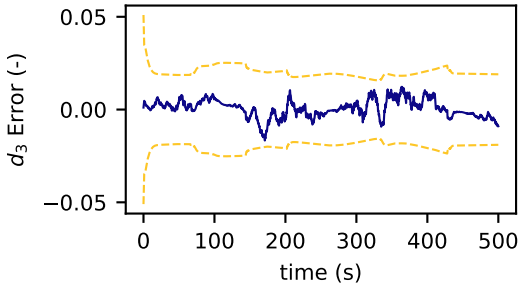
Five filters were developed out of the three kinematic formulations described in the previous section. The subtraction of the unobservable states formulation is written into a square-root unscented Kalman Filter (SR-uKF), and an Extended Kalman Filter (EKF). The formulation which only estimates the sunline direction is implemented in an EKF (Sunline-EKF). Finally the novel formulation is written in a EKF and SR-uKF (Switch-EKF, and Switch-SRuKF).



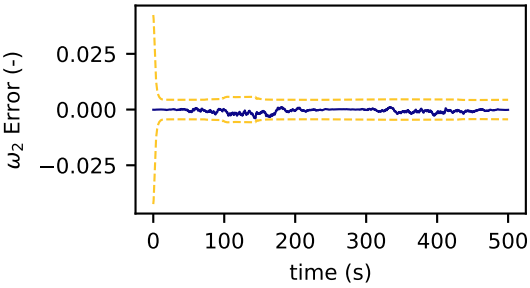
(a) First Heading Component



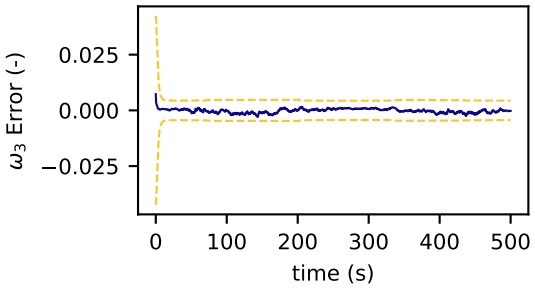
(b) Second Heading Component



(c) Third Heading Component



(d) $\boldsymbol{\omega}_2$ Component



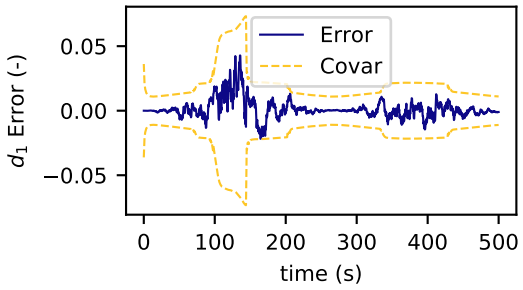
(e) $\boldsymbol{\omega}_3$ Component

Figure 3.5: State Error and Covariance Plots of Switch-EKF, FOV: 85°

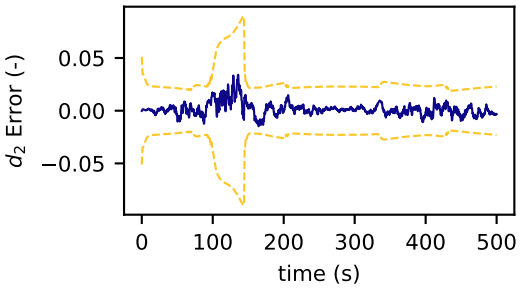
The simulation used is created using the Basilisk Software Package[1,1131]. All runs simulate a tumbling spacecraft in deep space, at 1AU from the sun. The problem assumes that the time needed for control is much smaller than the time needed to orbit around the sun, meaning that $\dot{\boldsymbol{d}} \approx \mathbf{0}$. The satellite is therefore not put on orbit around the sun, but kept in a constant position in the inertial frame. The simulations inputs are listed in Table 3.2. This framework allows for a

[1]http://hanspeterschaub.info/bskMain.html

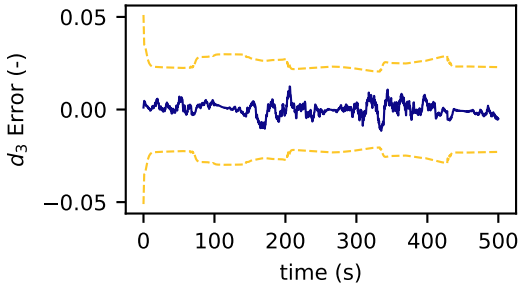
fully coupled dynamic simulation, and the runs use the same physical scenario (including noise), with only the filters changing between runs. The general simulation parameters used outside of Monte-Carlo analysis are summarized in Table 3.2.



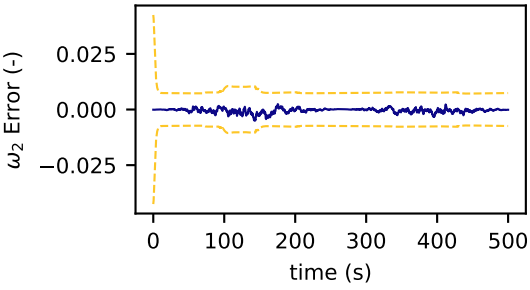
(a) First Heading Component



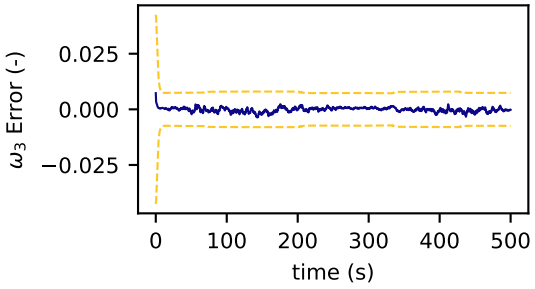
(b) Second Heading Component



(c) Third Heading Component



(d) ω_2 Component



(e) ω_3 Component

Figure 3.6: State Error and Covariance Plots of Switch-SRuKF, FOV: 85°

For all of the results, the filters retain the same process noise which is listed in Table 3.3. These values are chosen by reducing the post-fit residuals to noise at slow spacecraft rotation

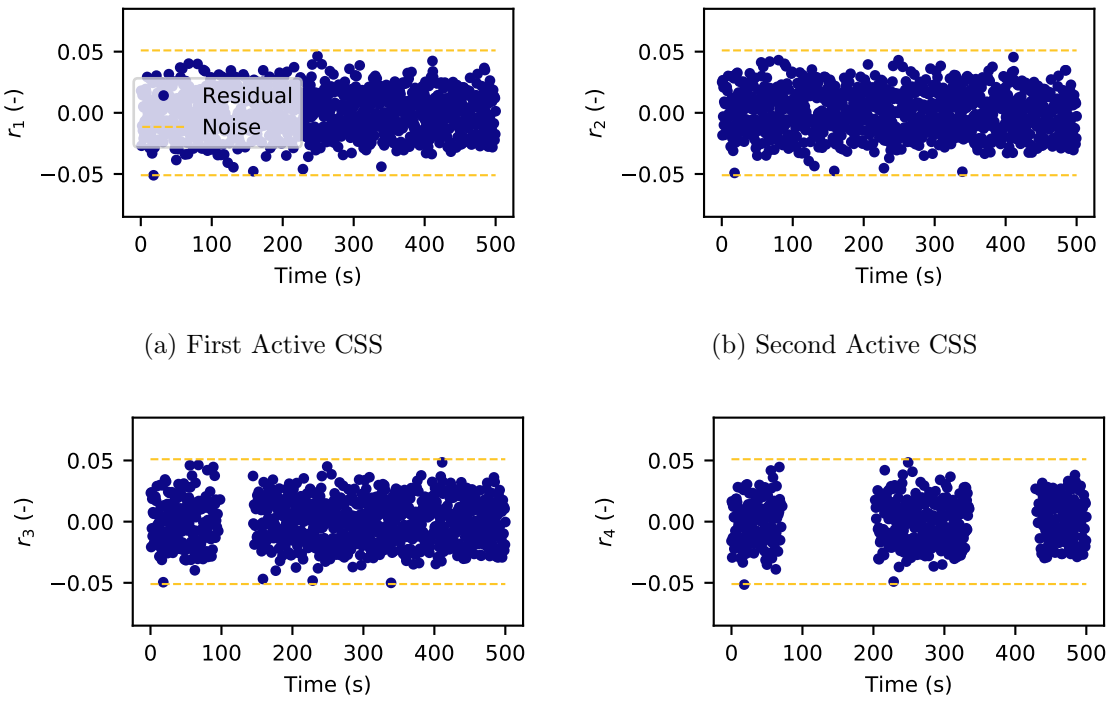
rates which is the most common state for a controlled spacecraft. It is then desirable to test the robustness of these filters as such in order to determine which ones are the best overall.

Table 3.3: State Noise Compensation (SNC)

Filter	Sunline-EKF	EKF	SR-uKF	Switch-EKF	Switch-uKF
SNC on $\boldsymbol{d}$	10^{-2}	N/A	10^{-3}	N/A	10^{-3}
SNC on rates	N/A	$2 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$8 \cdot 10^{-4}$	$8 \cdot 10^{-4}$

First the Switch filters are examined to ensure proper implementation and behavior. Second, all the implementations are compared in a scenario in order to observe overall performance and covariance behavior. Finally, the best filters are run in Monte-Carlo simulations with low and high observability to show the best overall performing filters.

3.4.1 Switch Filter Results



(a) First Active CSS

(b) Second Active CSS

(c) Third Active CSS

(d) Fourth Active CSS

Figure 3.7: Post Fit Residuals for Switch-SRuKF, FOV: 85°

This subsection examines the implementation of the Switch-EKF and Switch-SRuKF. These results are created using the simulation parameters of Table 3.2, which initialize the spacecraft in a mild tumble. Figure 3.5 shows the state error and covariance for the Switch-EKF filter, while Figure 3.6 shows the Switch-SR-uKF. Both these filters perform well as the state errors are within the covariance bounds and unbiased.

This is seen more specifically with the post-fit residuals seen in Figure 3.7 from the Switch-SR-uKF run. The measurements are brought down to noise, with the expected standard deviations which is expected since the simulation doesn't have any un-modeled forces acting on the spacecraft. The Switch-EKF post-fits are not displayed, but are nearly identical and provide confidence that the filter is working optimally. The gaps in the post-fits corresponds to changing number of sensors which can observe the sun. With this CSS pyramid, there are always 2 active sensors, with only brief passages down to 3 or 4 sensors.

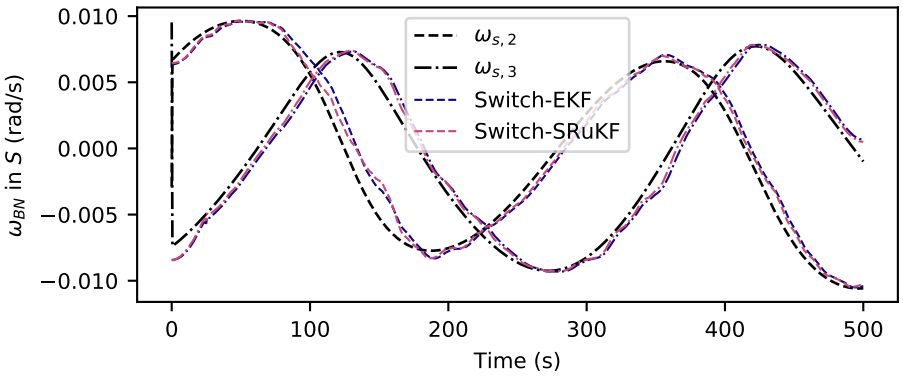


Figure 3.8: Switch Filters Tracking the Rates in the $\mathcal{S}$ Frame

Figure 3.8 shows one of the novel components of the Switch filters: direct body rate estimation. The filters can be seen tracking the true body rates in the $\mathcal{S}$ frame (disregarding the unobservable component which the filter ignores). Although just two components of this vector does not yet allow to fully estimate the body rate without extra information, it proves that the filters are functioning. This plot also shows the filter starting off in the middle of the singularity (heading and body frame vector aligned), switching frames, and pursuing the estimation with no

more singularities encountered. Furthermore, if additional headings were tracked and fused full body rate estimation would be achievable.

3.4.2 General Results

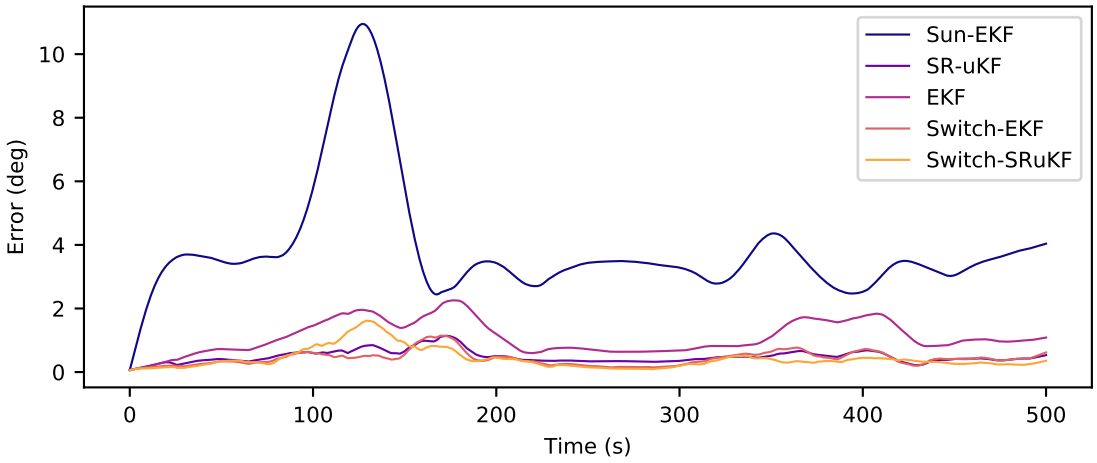
The simulation runs the filters with good and low quality measurements. As described previously, the filters are calibrated for their post-fit residuals to be noise at low speeds. With a field of view of 85° on each sensor, the problem has good observability, as seen in Figures 3.2b, 3.3b. Table 3.7 shows the post fit residuals' means and standard deviations for each of the activated devices. All the means are near zero which indicates no biases, while standard deviations are very close to the measurement noise of the instruments. These specific results are plotted in this section. The lower quality measurements (Figures 3.2a, 3.3a) are studied as well in the Monte-Carlo subsection. The results are summarized in Table 3.8 with the same initial conditions as the higher quality measurements.

$$\dot{\boldsymbol{d}} = \mathbf{0} = \boldsymbol{d}' + \boldsymbol{\omega}_{\mathcal{BN}} \times \boldsymbol{d} \tag{3.38}$$

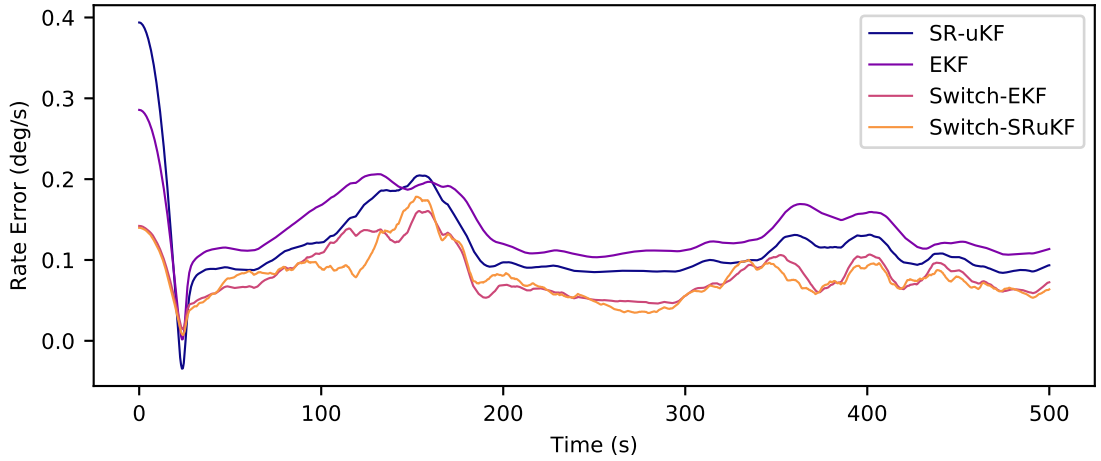
These filters are compared by plotting their off-pointing in degrees and the norm of the error on $\boldsymbol{d}'$ in Figures 3.9. For the switch filters (which do not estimate $\boldsymbol{d}'$) the rate is mapped back using the transport theorem as seen in Equation (3.38). Knowledge of $\boldsymbol{d}$ and $\boldsymbol{d}'$ does not allow identification of the body rate uniquely due to the rank deficiency of the cross operator. Hence, the current estimate of the sun-heading and the observable components of the body rate are used to compute $\boldsymbol{d}'$. The data is smoothed using a Savitzky-Golay algorithm[181] in order to differentiate between the curves more easily. This algorithm does lead to a spike at the end of Figure 3.9a and Figure 3.9b.

Table 3.4: RMS Errors from Truth, FOV: 85°

Filter	Sunline-EKF	EKF	SR-uKF	Switch-EKF	Switch-uKF
$\boldsymbol{d}$ RMS Pointing Error (°)	2.388	0.678	0.315	0.304	0.334
$\boldsymbol{d}'$ RMS Error (-)	N/A	0.089	0.087	0.055	0.056



(a) Pointing Miss-Angle (°)



(b) $\boldsymbol{d}'$ norm error

Figure 3.9: Comparative Performance of the Filters, FOV: 85°

Figure 3.9a shows the off-pointing errors of all the filters and Figure 3.9b displays the rate error. Tables 3.4 and 3.5 show the computed RMS errors for the filters in both the 85° and 60° FOV

Table 3.5: RMS Errors from Truth, FOV: 60°

Filter	Sunline-EKF	EKF	SR-uKF	Switch-EKF	Switch-SRuKF
$\boldsymbol{d}$ RMS Pointing Error (°)	7.651	4.695	3.003	3.568	2.151
$\boldsymbol{d}'$ RMS Error (-)	N/A	0.168	0.139	0.17	0.117

cases. The results show that the Switch filters outperforms the others both in rate and heading estimation. Due to the process noise on the body rates, the Switch filters sun-heading errors are slightly higher than some other results at low-speed tumbles. Yet at these speeds, all the filters provide errors that are less than half of a degree off. This can be seen more clearly in Figure 3.10a.

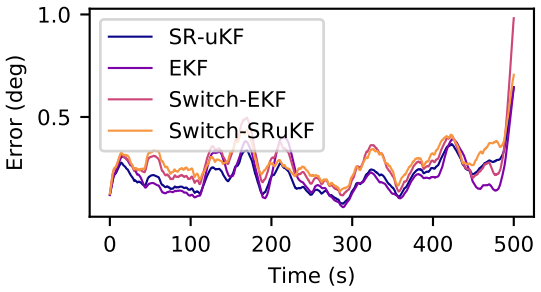
3.4.3 Monte-Carlo Analysis

In this subsection, the a Monte-Carlo analysis is run on the scenario. The dispersed parameters are the initial conditions to the spacecraft tumble: initial attitude and attitude rate. This general study allows one to ensure that the better performance of a specific filter is not attributed to favorable initial conditions. The dispersions are applied in 3 different scenarios. The first being

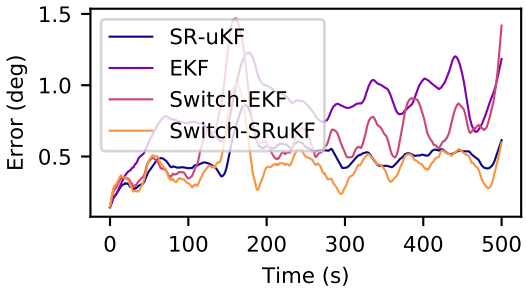
Table 3.6: Monte-Carlo Dispersions

Parameter	$\boldsymbol{\sigma}(t_0)$	$\boldsymbol{\omega}(t_0)$ (°/s)
Fast Distribution	$\mathcal{U}[0, 2\pi]$	$\pm\mathcal{N}[0.45, 0.55]$
Nominal Distribution	$\mathcal{U}[0, 2\pi]$	$\pm\mathcal{N}[0.05, 0.15]$
Slow Distribution	$\mathcal{U}[0, 2\pi]$	$\pm\mathcal{N}[0.001, 0.01]$

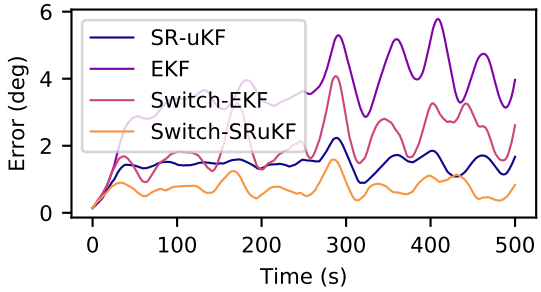
a slowly rotating spacecraft scenario. This is the scenario to which all the filters are calibrated. The second scenario is a nominal rotation, akin to a slow maneuver. The third scenario is a fast rotation spacecraft similar to a tumble. The dispersions applied in each of these cases are listed in Table 3.6.



(a) Slow tumble scenario



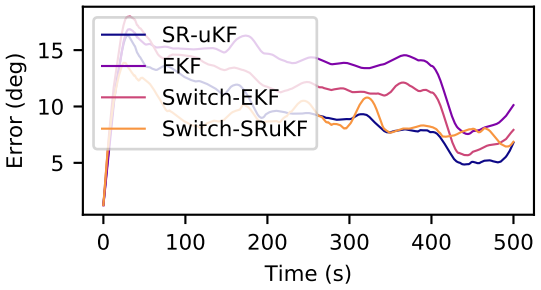
(b) Nominal tumble scenario



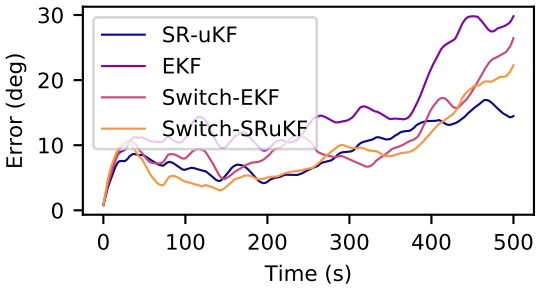
(c) Fast tumble scenario

Figure 3.10: Average of 10 MC Runs, FOV: 85°

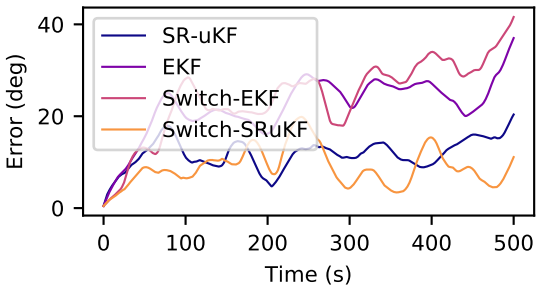
Figure 3.10 shows the results of 10 Monte-Carlo runs in the high-observability scenario. For clarity, the Sunline-EKF filter—which was not performing as well as the others—is removed from this analysis. This allows for a more focused analysis on the best filters. These runs show that the Switch-uKF performs consistently better than its competitors. At slow speeds the difference between all the filters is hard to gauge since this is the run that calibrated the process noise. It does seem that despite overall excellent performance, this is the only realm where the Switch formulations do not estimate sun-heading better than the others. Yet the Switch formulations, and more notably the Switch-SRuKF, handle the faster spacecraft rates considerably better than the other filters.



(a) Slow tumble scenario



(b) Nominal tumble scenario



(c) Fast tumble scenario

Figure 3.11: Average of 10 MC Runs, FOV: 60°

Figure 3.11 shows the results of 10 Monte-Carlo runs in the low-observability scenario. The Monte-Carlos at low speeds also contain very low observability scenarios where no more than two or three sensors are activated, which yield high errors. With fewer measurements all the filters perform less well, yet once again the Switch-SRuKF consistently yields the smallest heading errors. This shows the value of this formulation: in the event of component failure, the Switch filters will provide consistently better sun-heading estimates. This contributes to the robustness of the attitude determination system.

3.4.4 Sun-Heading Conclusions

The results shows the comparative performances of several filters and formulations attempting to solve the CSS-only heading determination problem. In order to provide a better more robust algorithm, kinematics akin to MRPs shadow set switching are implemented. This leads to a change

Table 3.7: Post Fit Residuals in nominal case, FOV: 85°

Filter	Statistics	Obs 1	Obs 2	Obs 3	Obs 4
Sunline-EKF	Means	-0.0007	0.0023	-0.009	0.0139
	Standard Deviations	0.0331	0.0313	0.0508	0.0351
EKF	Means	-0.0019	0.0004	0.0001	-0.0021
	Standard Deviations	0.0181	0.0198	0.0186	0.0182
SR-uKF	Means	0.0013	0.0027	0.0029	0.0096
	Standard Deviations	0.0187	0.0244	0.0344	0.0692
Switch-EKF	Means	-0.0026	-0.0004	0.0006	-0.0012
	Standard Deviations	0.0191	0.033	0.0356	0.0175
Switch-SRuKF	Means	0.0	0.001	0.0025	0.0096
	Standard Deviations	0.0186	0.0243	0.0343	0.0692

Table 3.8: Post Fit Residuals in nominal case, FOV: 60°

Filter	Statistics	Obs 1	Obs 2	Obs 3	Obs 4
Sunline-EKF	Means	-0.0001	-0.0011	-0.0027	0.0256
	Standard Deviations	0.0359	0.0362	0.0524	0.0311
EKF	Means	-0.0014	-0.0062	-0.0094	-0.0118
	Standard Deviations	0.0203	0.0363	0.0457	0.0226
SR-uKF	Means	0.0021	0.0023	0.0036	0.0068
	Standard Deviations	0.0188	0.0295	0.0628	0.0786
Switch-EKF	Means	-0.0039	-0.0033	-0.0058	-0.0138
	Standard Deviations	0.0219	0.0388	0.0604	0.0273
Switch-SRuKF	Means	0.0001	0.0011	0.0051	0.0067
	Standard Deviations	0.0187	0.0291	0.0623	0.0786

in the process noise derivation for an EKF, and requires a switch in the covariance on the rate states as well. At slow rates, all filters perform approximately the same. Then at higher rates, the switch formulations provide better results than all other filters implemented on the problem.

The Switch-SRuKF performs the best all around, whether the CSS have a narrow or wide field of view. The non-linear propagation of sigma points combined with the novel switch-formulation provides a good propagation step and allows for full utilization of the measurements despite inherent unobservability. In fact, Switch filters have removed the problem of the unobservable rate component from the estimation entirely. If combined with wide field-of-view CSS instruments it does not suffer from any observability issues, numerical or analytical.

3.5 Generalizing Switch-Kinematics to OpNav Headings

The results presented for sun-heading estimation encourage generalization of this filter for heading determination. In OpNav, heading estimates have several utilities. On the one hand, they provide a secondary attitude measurement. This can be used for flight software robustness checks and fault detection, while also being used for their primary purpose: in pointing modes (Science-point, Earth-point, etc.). Indeed, heading based attitude determination methods provide relative attitude to their targets. Depending on the scenario, the position of the target might not be well known in the inertial frame: when pointing at an asteroid, or at a science target such as 486958 Arrokoth.[90,137] In this case, relative attitude is paramount to accurate imaging. In scenarios where the target's inertial position is well known, relative attitude still allows to solve for offsets in the camera position and orientation. Furthermore, one method of OpNav uses distant objects or beacons to determine position through triangulation.[108] This was done by Deep Space 1 when demonstrating the AutoNav FSW. Finally, in an attitude determination context, combining CSS measurements for sun-heading with OpNav measurements of a nearby planet, a full attitude estimate can be extracted.

This heading filter is generalized as the Switch-SR-uKF, which was the highest performing filter in the previous section. The state vector is $\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d} & \omega_{s,2} & \omega_{s,3} \end{bmatrix}^T$, and the dynamics of the filter are given by Equation (3.41) and (3.42). It is used in order to complete the GNC system for the attitude guidance and control module displayed in the previous Chapter.

3.5.1 OpNav Measurements

The measurement model is given in Equation (3.39), and the linearized measurement model $[H]$ is defined as $[H] = \left[\frac{\partial \boldsymbol{G}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right]^*$, where $\boldsymbol{G}$ is the new measurement model. In all of this work, the measurement models are extracted from the filter to compare and contrast methods. This means the measurements input to the heading filter are $\boldsymbol{r}_{\mathcal{BN}}$. This measurement is normalized for heading

determination, giving the following $\boldsymbol{G}$ function:

$$G_i(\boldsymbol{X}) = {}^{\mathcal{B}}\hat{\boldsymbol{r}}_{\mathcal{BN}} \tag{3.39}$$

This yields the partial derivatives for the $[H]$ matrix:

$$[H] = \begin{bmatrix} [I_{3\times 3}] \\ [0_{2\times 3}] \end{bmatrix} \tag{3.40}$$

This filter can therefore work with a set of different measurement models which output the same measurement type. In this section, the *Hough Cirlces* algorithm is used and is described in more detail in subsection 2.4.2. Since the measurement vector is normalized, the only component of the image processing method that are used are the center points. In this regard, other image processing methods such as center-of-brightness finding would perform similarly.[122]

3.5.2 Kinematics and Assumptions

The generalized switch filter uses the same kinematic formulation as the sun-heading version. The propagation equations are recapitulated here:

$$\boldsymbol{X}' = \boldsymbol{F}(\boldsymbol{X}) = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d}' \\ \dot{\omega}_{s,2} \\ \dot{\omega}_{s,3} \end{bmatrix} = \begin{bmatrix} -{}^{\mathcal{B}}\boldsymbol{\omega}^* \times {}^{\mathcal{B}}\boldsymbol{d} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{S}}\begin{bmatrix} 0 \\ \omega_{s,2} \\ \omega_{s,3} \end{bmatrix} \\ -[\mathcal{BS}] \\ 0 \\ 0 \end{bmatrix} \times {}^{\mathcal{B}}\boldsymbol{d} \tag{3.41}$$

$$[A] = \left[\frac{\partial \boldsymbol{F}(\boldsymbol{d}, t_i)}{\partial \boldsymbol{X}}\right] = \begin{bmatrix} -[{}^{\mathcal{B}}\tilde{\boldsymbol{\omega}}^*] & -[\tilde{\boldsymbol{d}}] \begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{s}}_2 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_3 \end{bmatrix} \\ [0]_{2\times 3} & [0]_{2\times 2} \end{bmatrix} \tag{3.42}$$

The main assumption made in the previous section was the $\dot{\boldsymbol{d}}$, the inertial derivative of the heading vector being estimated, was constant and equal to zero. This lead to the fact that the only possible rate between the $\mathcal{S}$ and $\mathcal{N}$ frame was along the unobservable direction. This was therefore ignored by the kinematics as it was impossible to parse from the measurements. In an OpNav context, the

planet is likely to be moving in the inertial frame. For instance, if the body being observed is close or if the spacecraft is on a low orbit. In such a case, since $|\dot{\boldsymbol{d}}| > 0$, the rate about the second and third axis of the $\mathcal{S}$ frame will be non zero:

$$^{\mathcal{S}}\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} = {}^{\mathcal{S}}\begin{bmatrix} 0 & \omega_{\mathcal{SN},2} & \omega_{\mathcal{SN},3} \end{bmatrix}^T \tag{3.43}$$

Equation (3.43) has the about-heading rate zeroed since it is the un-observable component, as done previously. A natural idea would be to fully generalize these equations the main modification that must be made is in this assumption:

$$\dot{\boldsymbol{h}} = \dot{\boldsymbol{r}}_{\text{Mars}} - \dot{\boldsymbol{r}}_{\mathcal{BN}} \tag{3.44}$$

This would provide the movement of the spacecraft about the planet, and therefore would allow to compute all the terms for the rate relation $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} - \boldsymbol{\omega}_{\mathcal{B}/\mathcal{S}} = \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$. This presents a number of issues nonetheless:

- In the case of a non-inertial pointing target, the position of the target must be know and added to the filter.
- Knowledge of the the spacecraft position produces a coupling between an orbit determination solution and the attitude solution.

Both of these points imply that the heading solution is tethered to other estimates, spacecraft states, and environment states. It therefore breaks the simple and minimal formulation that the Switch Filter Kinematic proposes. If this is to be the case, adding other sensors in order to resolve the rate un-observability would be a simpler solution than implementing this filter. In order to remain in a simple input paradigm, the filter is tested to see how well the zero-inertial heading rate assumption holds. This is to say: although $|\dot{\boldsymbol{d}}| > 0$, does the filter still converge and at what rates does the assumption no longer hold.

Feasibility of this method for on-board, autonomous use requires guidance and control in the loop. Therefore, the primary means of validating the assumption is done with scenarios that

include guidance and control with the filter solution. De-coupled estimation and control scenarios are also run to ensure that they perform better than the control cases and therefore that the fully simulated cases are bounding.

3.5.3 Filter Results

The filter performance has been evaluated for distant beacons such as the sun. In order to evaluate the filter quality of the estimate, a similar scenario than the guidance example shown in Chapter 2 is developed. The *Basilisk* simulation modules implemented are identical and can be found in Tables 4.4 and 5.3. Tables 3.9 and 3.10 show the simulation parameters used for these specific test cases. These are similar to previous simulations, except for a few points.

Table 3.9 shows the three main scenarios run which differ only by the semi-major axis values a. The first case is a $28,000$km orbit, which is at a high altitude and therefore slow moving. For reference, Areosynchronous Orbits (ASO) have semi-major axes of approximately $20,400$ km. The second orbit is lower and used nominally throughout this thesis, with $a = 18,000$km, or an altitude of about $14,600$km. Finally, a close orbit with semi-major axis $a = 8,000$km and altitude of $4,600$km is tested.

Table 3.9: Scenario Parameters

Scenario	$\boldsymbol{\sigma}_{\mathcal{BN}}$	$\boldsymbol{\omega}_{\mathcal{BN}}$	Orbital Elements $(a, e, i, \Omega, \omega, f)$	Camera FOV
Distant (1)	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$28,000\text{km}, 0, 20°, 25°, 190°, 100°$	$20°$
Nominal (2)	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$18,000\text{km}, 0, 20°, 25°, 190°, 100°$	$40°$
Proximity (3)	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	$8,000\text{km}, 0, 20°, 25°, 190°, 100°$	$80°$

In order to see the differences in orbital rates, Kepler's third law can be applied, where $\mu_{\text{Mars}} = 4.28284 \cdot 10^4 \text{km}^3/\text{s}^2$

$$T = 2\pi\sqrt{\frac{a^3}{\mu}} \tag{3.45}$$

This yields the values in Table 3.11 using the identity $2\pi = T\omega$. These rates give some insight into the sensitivity of the filter to a rotating inertial frame. Given a time-step of 0.5 seconds on the filter, the moving planet will prescribe errors up to $0.009°$ for the Proximity scenario in one

Table 3.10: Flight Software Parameters

Flight Software Instantiated	**Necessary parameters at initialization**
Heading Filter	ProcNoise = $10^{-12}[I]_{5\times5}$, HeadingCovar = $0.2[I]_{3\times3}$, RateCovar = $0.005[I]_{2\times2}$, $\alpha = 0.02$, $\beta = 2$, $\kappa = 0$, noiseSF = 1.001
Image Processing (arguments for HoughCirlcle method[1])	param1 = 300, param2 = 20, minDist = 50 minRadius = 20, dp = 1, maxRadius = 409, noiseSF = 0.5
OpNav Point	minAngle = 0.001°, timeOut = 100s $\omega_{\text{search}} = [0.06, 0.0, -0.06]°/\text{s}$, ${}^{\mathcal{C}}h_c = [0., 0., 1]\text{m}$
Pixel Line Transform	Planet Target is Mars
MRP Feedback RW	K = 3.5 , P =30 (no integral feedback)
RW motor Torque	Control axes are ${}^{\mathcal{B}}[\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3]$

Table 3.11: Mars Orbit Scenarios

Scenario	T (hours)	ω (°/s)
Distant (1)	39.51	0.0025
Nominal (2)	20.37	0.0050
Proximity (3)	6.035	0.0166

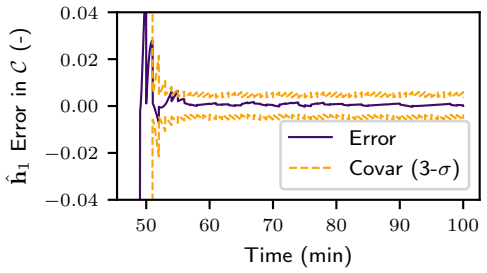
time-step. These errors on the estimate are outside the fact that the rate will also have difficulty converging given that the kinematics do not account for this motion.

The following subsections provide results for each of the scenarios including the reaction wheel control law and OpNav guidance law derived in Chapter 2. The result summary also shows results for the pure navigation case where the control does not influence the future images and therefore excludes any induced oscillations. In these results, all parameters outside of camera field of view and semi-major axis are unchanged. The time step for FSW algorithms and simulation algorithms is held constant at 0.5 seconds.

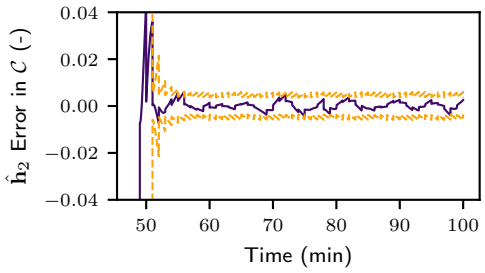
3.5.3.1 Distant Planet Case

In this distant case, at $a = 28,000$km, the results are promising and show that the filter has no issue converging. This is seen notable by looking at the state error and covariance plots seen in Figure 3.12 where the covariance comes down as the measurements are processed. The noise

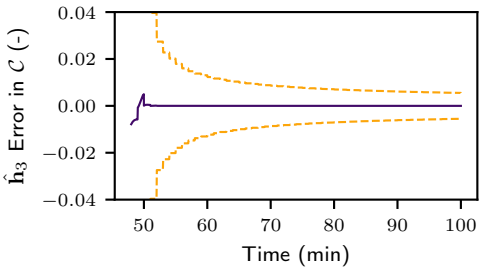
scale-factor (noiseSF) seen in Table 3.10 helps slow down the collapse of the covariance. This is applied on the covariance during the update and not directly to the measurement as suggested by filtering best practices.[27]



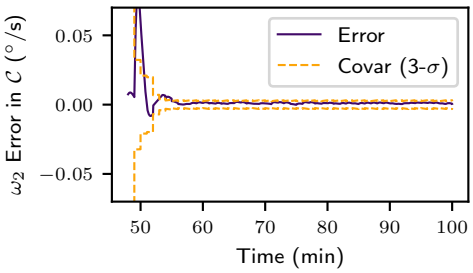
(a) First Heading Component



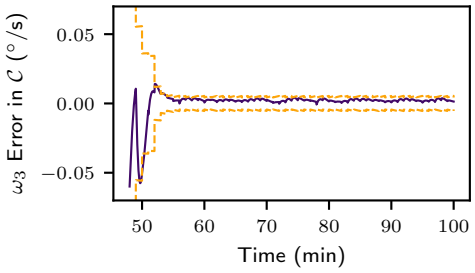
(b) Second Heading Component



(c) Third Heading Component



(d) $\boldsymbol{\omega}_2$ Component



(e) $\boldsymbol{\omega}_3$ Component

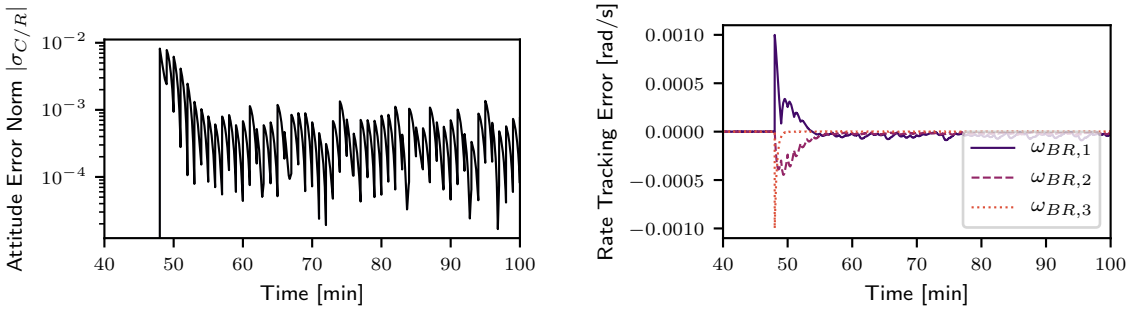
Figure 3.12: State Error and Covariance Plots of Heading-SRuKF, FOV: 20°, $a = 28,000$km

These results are all presented in the camera frame. This explains why the $\hat{h}_3$ component shows almost no error, and the slight oscillation in the $\hat{h}_2$ term comes from the spacecraft sweeping

over the planet along that axis. This is the only trace of the assumption in these results.

The camera used for this scenario has a field of view of 20°. This is a nominal value for a space imaging camera that has been used in the literature[46,59,71] which can provide both a detailed map of a close object, all the while having a wide enough field of view to scan for distant bodies and resolve full disks.

Figure 3.13 show the results from the guidance and control modules. These indeed show that the spacecraft stabilizes and that the attitude comes to a Mars-pointing frame within seconds. The gaps in the propagation are due to the spacecraft halting the control when close to the target in order to avoid bouncing.

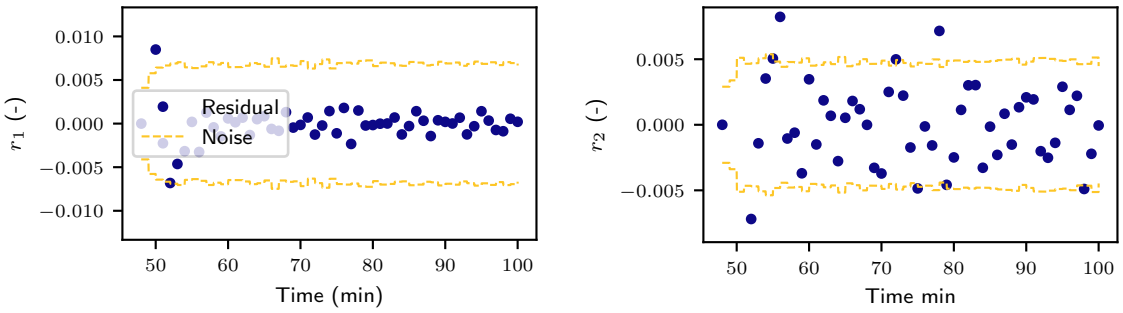


(a) MRP Error Norm

(b) Rate Tracking Error

Figure 3.13: Control Results for Heading-SRuKF, FOV: 20°

Finally, Figure 3.14 show post fit residuals seen by the filter on the first and second axes of the measurement.



(a) First Residual Component

(b) Second Residual Component

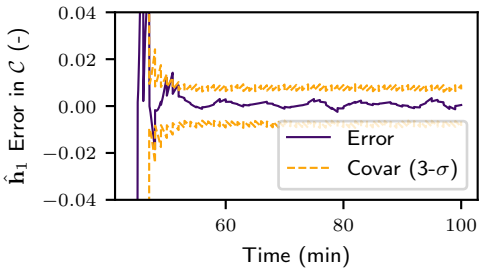
Figure 3.14: Post Fit Residuals for Heading-SRuKF, FOV: 20°

These are mostly noisy, and despite a slight over-estimation of the measurement noise about the first axis, they do not display any clear signal. This shows that the spacecraft movement about the orbit is not significant enough to disturb the filter. This scenario shows positive results. It is far from a beacon detection orbit in which the assumption would hold all the more, yet it functions as expected.

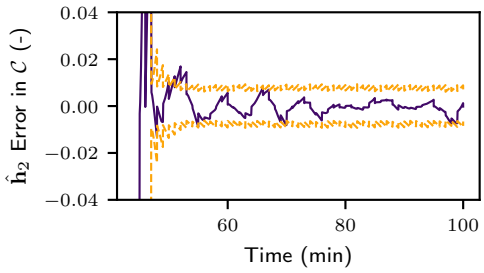
3.5.3.2 Nominal Planet Case

In the nominal planet case, the spacecraft is on an orbit with semi-major axis $a = 18,000$km. The camera used for this scenario has a field of view of 40°. The field of view of the camera must increase as the spacecraft's orbit comes closer to the surface in order to image the entire disk. This is indeed a wider field of view camera that is better suited for imaging a full planet.

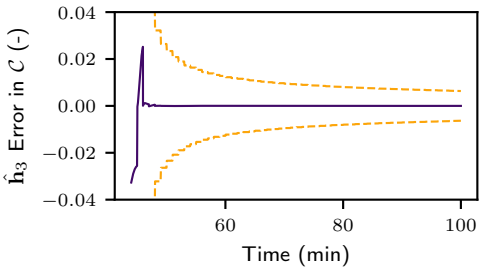
In this scenario, the filter still converges but displays more symptoms of a broken assumption. This is seen notably in Figure 3.15b where the oscillation along the moving axis is more prominent. This is accentuated by the guidance and control algorithm and could be accounted for in several ways. At this altitude, it could be interesting to take more frequent images for measurements, or the filter could run on a shorter time step in order to not see the errors as much.



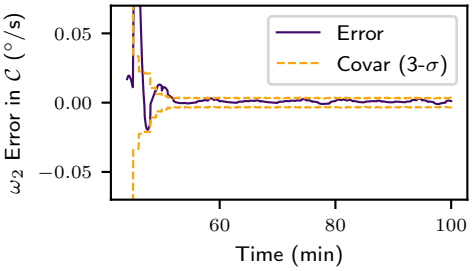
(a) First Heading Component



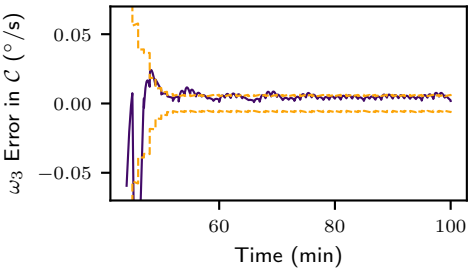
(b) Second Heading Component



(c) Third Heading Component



(d) $\boldsymbol{\omega}_2$ Component

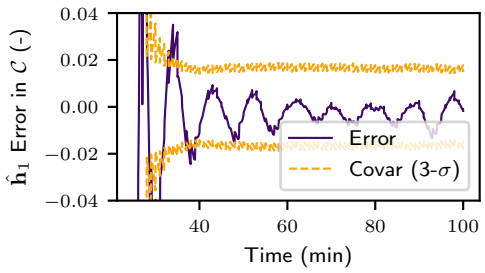


(e) $\boldsymbol{\omega}_3$ Component

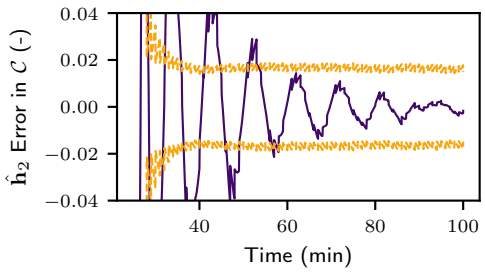
Figure 3.15: State Error and Covariance Plots of Heading-SRuKF, FOV: 40°, $a = 18,000$km

In the interest of seeing limits of the assumption, all simulation pararmeters kept identical to the previous scenario. This allows to show the edge where the filter converges easily but oscillates around a zero error significantly.

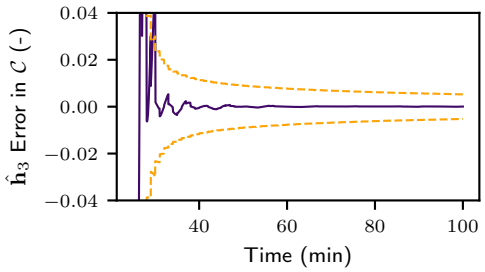
3.5.3.3 Proximity Case



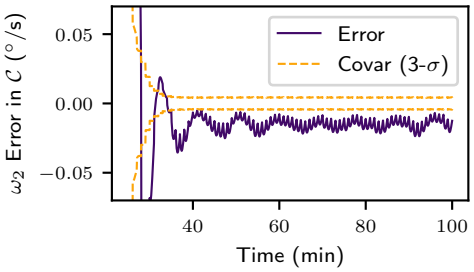
(a) First Heading Component



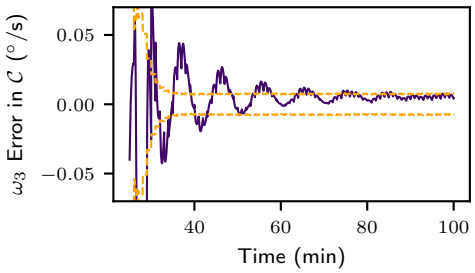
(b) Second Heading Component



(c) Third Heading Component



(d) $\boldsymbol{\omega}_2$ Component



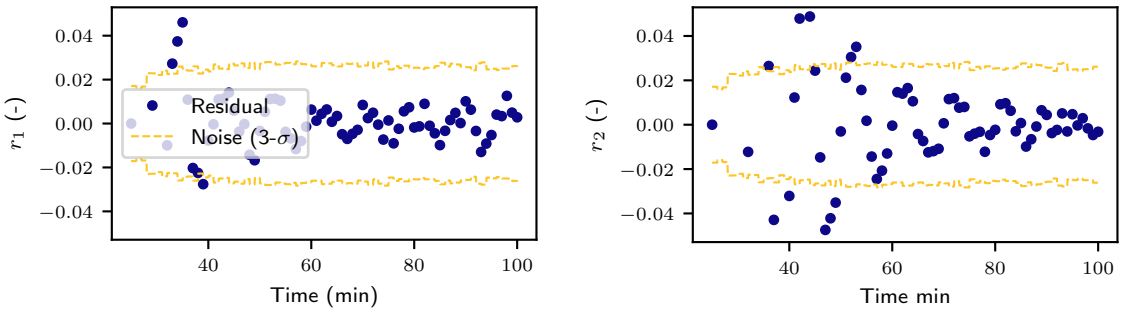
(e) $\boldsymbol{\omega}_3$ Component

Figure 3.16: State Error and Covariance Plots of Heading-SRuKF, FOV: 80°, $a = 8,000$km

In the proximity planet case, the spacecraft is on an orbit with semi-major axis $a = 8,000$km. The camera used is now very wide angled with a 80° field of view. At this altitude, a narrower angle would not allow to image the planet. Although not unreasonable for entry decent and landing

— which have cameras with 70° fields of view[59,71] — this camera would rarely be used for these purposes. Indeed at this altitude, a camera with a more narrow field of view[69] and higher resolution would allow for Terrain Relative Navigation (TRN) through feature tracking.[151]

Figure 3.16 shows the now limited behavior of the fully coupled navigation, guidance, and control scheme. Indeed the oscillations of the filter are more notable than before and the post fit residuals in Figure 3.17 show the structure tied to this unmodeled rotation.



(a) First Residual Component

(b) Second Residual Component

Figure 3.17: Post fit Residuals for Heading-SRuKF, FOV: 80°, $a = 8,000$km

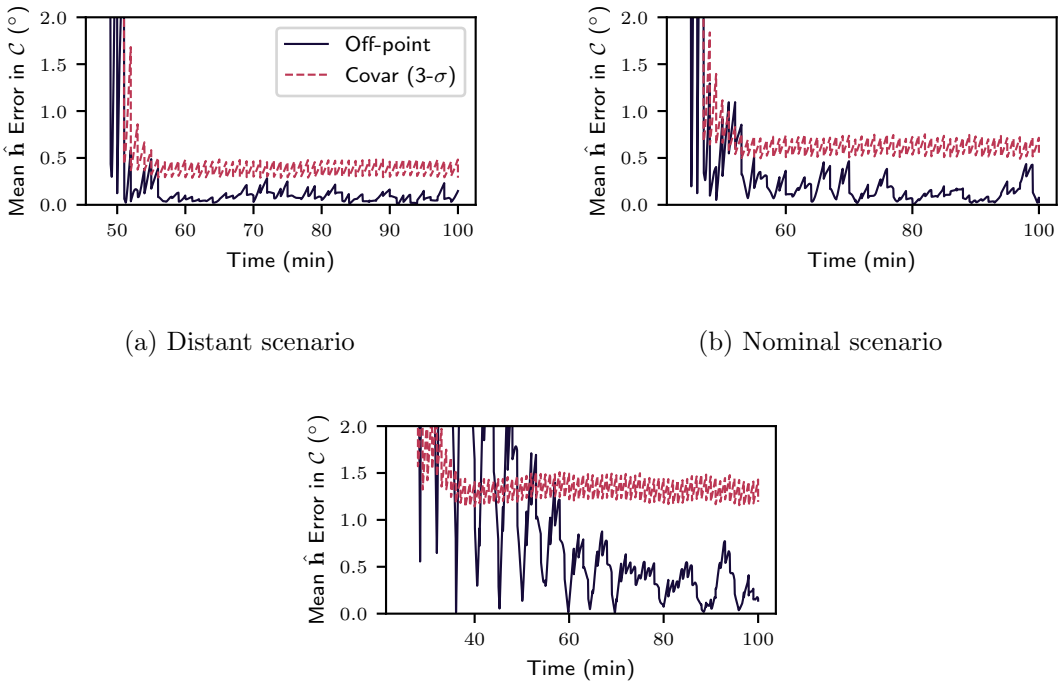
It is valuable to note, nonetheless, that the filter converges with the control. The oscillations are likely due to the control misguiding the spacecraft due to state errors, as seen in the next section. A separate study could continue this work in order to identify the coupling between the control and navigation solution.

3.5.4 OpNav Heading Results Summary

This section summarizes the results of the OpNav pointing scenarios. The off-pointing errors and covariance are provided for the previous scenarios as well as for de-coupled scenarios in which the control does not use the measurements.

Figure 3.18 summarizes the performance of the implemented filter in all three orbit cases. This shows the increasing sensitivity to the fast changing orbit dynamics, and suggests the limits

of the current ADCS system.



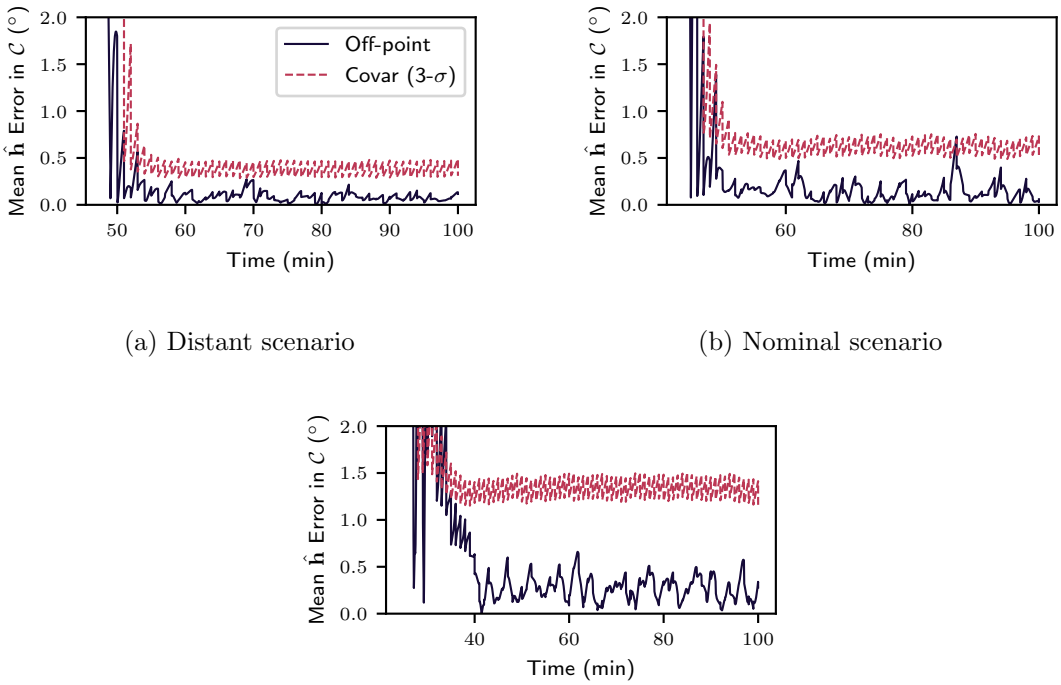
(a) Distant scenario

(b) Nominal scenario

(c) Proximity scenario

Figure 3.18: Off-Pointing Results in Coupled Control-Navigation Scenarios

Figure 3.19 shows the performance of the filter in all three orbit cases without using the measurements for control. This shows that the filter performs better without the feedback of the control and the estimate. This figures shows that the performance is not limited at this proximity case, as an more adapted control law could provide better results.



(a) Distant scenario

(b) Nominal scenario

(c) Proximity scenario

Figure 3.19: Off-Pointing Results in De-Coupled Control-Navigation Scenarios

3.6 Conclusions

Using switch-kinematics to avoid un-observability is a novel solution for heading estimation. Through these more complex kinematics, the Switch formulations analytically extract rate unobservability. This provides confidence in regard to the numerics of the filters as well as the overall state error. Furthermore, these results generalize well to OpNav scenarios. Hence, these filters provide a general solution to the array of problems that come from heading estimation.

The filter also performs well on cases that are outside of it's nominal operating range. Using the listed cameras for limb-based image processing at those altitudes is uncommon. Indeed, at that stage of a mission, much better results can be found by doing terrain relative navigation. This shows that not only does the filter perform well within it's assumptions, but it also needs to violate them significantly in order to see a dip in performance.

This chapter also displays the capabilities of the simulation developed in Chapter 2. Indeed, the last results show that a de-coupled attitude scenario does not provide full insight on the stability of the issue.

Future work could see the full attitude knowledge from heading information. This can be implemented using standard methods such as TRIAD, OLAE, QUEST[184] or stochastic methods.[56] Full attitude using several headings fits well within the context of Wahba's formulation[212] which uses relative weights to quantify the accuracy of each vector. Methods exist to minimize Wahba's cost function with one dominant measurement[141,220] which could apply well to these results.

Chapter 4

Orbit Determination using Centroid and Apparent Diameter

4.1 Overview

This research goal focuses on improving orbit determination using OpNav measurements. The errors and uncertainties from the camera, the distance from the body, and the image processing methods directly determine the quality of the state estimation. Harnessing the designed OpNav test-bed introduced in Chapter 2, all of these parameters can be analyzed both independently and simultaneously.

This chapter focuses on how to extract CAD or Limb information from an image, and how to quantify the quality of the pose result. The contribution to the field is, in part, the implementation and end-to-end validation of the *Hough Circle* transform for on-orbit navigation. By allowing the FSW to fit simple circles, the robustness of the FSW can increase while the performance is shown to be equivalent. Error quantification needs to be done as a function of camera parameters and the uncertainty of the fitting algorithm. This is done specifically for simple image processing algorithms with the intent of flying on-board and in the loop.

4.2 Motivation and Assumptions

OpNav uses information in the environment as seen through an on-board camera in order to determine spacecraft states. Although just one in many methods, this research focuses on enhancements to autonomous OpNav for spacecraft state estimation. Limb finding has seen recent developments and is notably planned for use on Artemis 1 (previously referred to as Orion's Ex-

ploration Mission 1[99]), as well as for Jupiter and Saturn exploration using the planet satellites.[21] Autonomous OpNav remains a sought-after navigation method as it requires only cameras (which can be both light-weight and used for other purposes) and fundamentally relies on imaging the object that is being studied and orbited as opposed to Earth-based data, or distant pulsars.

In OpNav, measurements derived from landmark observations,[133] point distribution methods,[203] star-occultation methods,[173] crater-tracking,[170] are some of many feature tracking methods which provide promising results. However, they come at a computation cost. Indeed, the current state-of-the-art for OpNav is Stereo-Photoclinometry[76,77,178] (SPC) which allows the spacecraft to map and navigate the spacecraft environment with high precision. Nonetheless, it relies very heavily on Earth contact for its intensive image processing algorithms. With the goal of autonomy, this research will focus on on-board methods for image processing.

Similarly, ORB-SLAM[61,155,156] and other cross-correlation methods[29,143] are booming in aerospace. These hold great promise for small body autonomous orbiting and have already proven to be useful on missions such as ESA's Rosetta and ongoing missions such as OSIRIS-REX. For preliminary developments, the proposed research will focus on simple CAD, with the intent of allowing more intensive image processing methods in the future.

In previous chapters, the primary information that has been extracted were centroids, which have been used for heading determination and pointing. This chapter now intends to use the apparent diameter in order to get a measurement of range by using the size of the fitted circles. This goal introduces a few immediate caveats:

- In order to measure relative distance to an object using it's apparent size, the object and it's physical shape must be known
- For the circle to be a good approximation of the object, it must be close to spherical
- The inertial attitude of the observer must be known in order to fully determine the position given the symmetries of the sphere.

Addressing these issues creates the platform for the OpNav study of the chapter. Indeed, addressing

the first point immediately narrows the applicability of these results. Knowing the properties of the celestial body excludes visiting asteroids or comets which are not yet explored. This ties in to the second point: most small bodies do not have sufficient mass in order to become spherical, whereas telluric planets and dwarf planets become spherical under their gravitational pull. Finally the last point is solved by assuming a star tracker is on-board the spacecraft. This is generally not considered a difficult assumption to fulfill as most spacecraft fly several of these cheap and reliable instruments.

The first two points force this application towards inner planets, icy moons, or any other body which is roughly spherical with a known size. Table 4.1 shows that the ratio of the equatorial and radius for the 8 planets and the Moon are generally low. In fact, apart from gas giants, all of the planets show smaller than 1% oblateness. These contextualize the type of planets and moons

Table 4.1: Celestial Body Oblateness

Celestial Body	Body Oblateness
Mercury & Venus	< 0.05 %
Earth	0.3 %
Moon	0.1 %
Mars	0.7 %
Jupiter	6 %
Saturn	10 %
Uranus	2.3 %
Neptune	1.7 %

that can be used for the methods developed in this chapter. In order to constrain the simulation possibilities, a similar orbit to the to the nominal heading situation is introduced and detailed in subsection 4.3.3. This provides continuity and focuses on Mars navigation. Here, the autonomy does not happen during a maneuver, but rather it is achieved on a routine level on orbit.

Routine autonomy presents a different kind of challenge than during mission-critical phases. Both scenarios require high levels of fidelity and robustness to a wide set of conditions but with more frequent use of the algorithms comes more exposure to potential faults. The use of such algorithms requires commensurately thorough analysis in order to develop confidence in the long-

term performance. One reason that makes this difficult to achieve is the difficulty in reproducing flight-like conditions on Earth. Furthermore, a flight-like simulation must provide the analysis capabilities required for mission design: it must enable Monte-Carlo (MC) capabilities as well as easy scriptability to modify scenarios with ease. Finally, developing open-source code bases also enables continuous validation, testing, and community support.

The method developed in this chapter focuses on the autonomous navigation about Mars. In this way, this work provides an extension on the AutoNav[108] framework, extending the image processing to *Hough Circle* instead of center-of-brightness, and to on-orbit navigation instead of deep-space cruise or impact trajectories. Furthermore, this study builds on previous chapters and therefore couples in the pointing problem with all the results. Both the coupling of pointing and orbit determination, and the end-to-end performance of the *Hough Circle* transform are the novel components of this work. Indeed, although this circle finding algorithm has been used extensively in robotics, it's applications to the spacecraft navigation field have primarily been for crater detection[213,218] and has not been applied to CAD navigation, usually for speed concerns[41] for fitting conics. In order to evaluate this method, a state-of-the-art algorithm is implemented: the Non-Iterative Horizon-Based method by Single Value Decomposition (NIH-SVD).[49]

This work therefore presents an implementation of a state-of-the-art limb-based OpNav method, while simultaneously using a more basic yet robust *Hough Circle*[171] finding algorithm. The Hough transform technique follows the principle of maximum likelihood estimation, it can thus be considered as a discretized version of a maximum likelihood estimation process.[225] Both ellipsoids and spheres mathematically project to ellipsoids on a camera plane. This has made ellipse fitting the default way to fit limbs for navigation. For ellipses, clustering methods such as *Hough* transforms are too numerically complex and slow and have been discarded for navigation though discussed.[129] This research presents fully coupled Attitude-OD OpNav on orbit using circle fitting instead of ellipse fitting, as well as comparative capabilities for different methods in a ideal flight-like simulated environment. Given the pairing of the pointing tasks which centers the planet on the image — hence limiting the projection errors — and the use of planets which are not largely

oblate (see Table 4.1) the results are compared to ellipse fitting methods in order to display the capabilities of *Hough Circles* for navigation.

The example scenario referenced in this research is a spacecraft on Mars orbit using only Centroid and Apparent Diameter for OpNav.[42,43] This scenario — camera images seen in Fig. 4.1 — enables testing of a full OpNav sequence, from taking images to control. Images processed are color, though the different color channels are not used as is common in space imagery. Building off previous work in Chapter 2[204] exploring closed-loop optical navigation simulations, this work develops a novel autonomous on-orbit OpNav architecture using Centroid and Apparent Diameter (CAD) measurements about Mars for simultaneous pointing and Orbit Determination (OD).

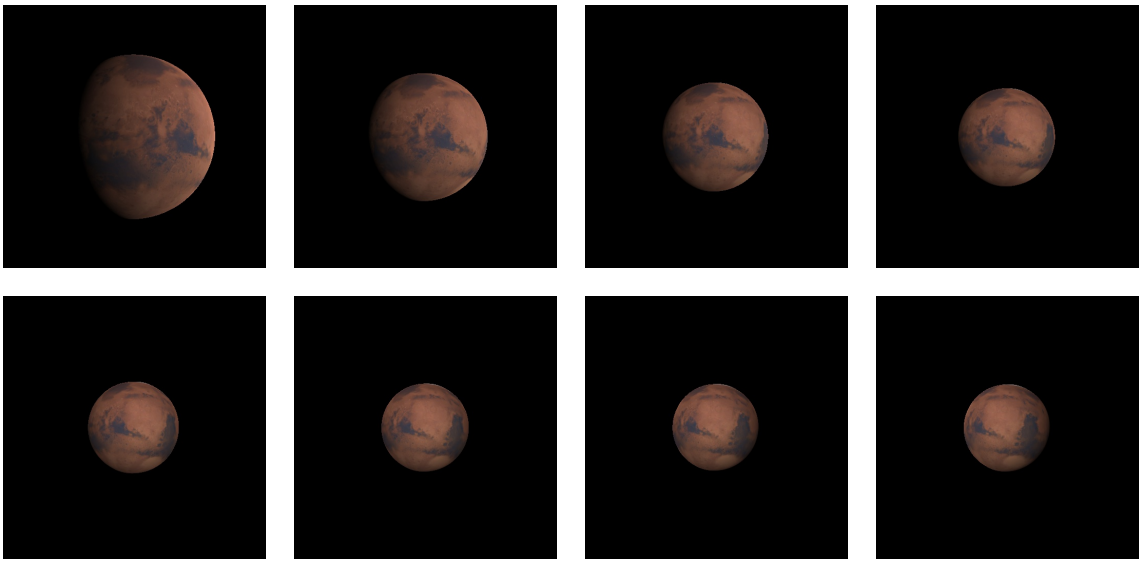


Figure 4.1: Camera View as the Spacecraft Moves the Simulation

The first section will describe the architecture of codebase, which allows for the level of testing desired. A description of the scenario implemented follows, as well as a recapitulation of the image generation method. The two methods are then described in detail in order to highlight differences. Results are shown in a the nominal scenario, and Monte-Carlo results are also presented. This last section allows to discuss some the of sensitivities discovered empirically, and opens the door to more in-depth mission analysis.

4.3 Simulating a Mars OpNav Orbiter

4.3.1 Filter Architecture

Part of the motivation behind the development of *Basilisk* has stemmed from the desire to test flight code from it's inception all the way to flight. This is done by embedding the FSW algorithms in virtual environments and hardware in the loop environments, and aims to fly exactly the code that is tested. One of the key enables towards this goal is the use of modular building blocks for algorithm creation. This allows to compare and contrast, couple and decouple, and test the parts of the GNC chain for performance testing. This paradigm is therefore also applied to the filtering developed for OpNav. In order to keep filters modular and agnostic of the image processing method used. This is done by changing the filter input from pixel and line to the relative pose of the camera relative to the body. In order to achieve this decoupling, the uncertainties of the image processing methods must be mapped back to the measurement being extracted.

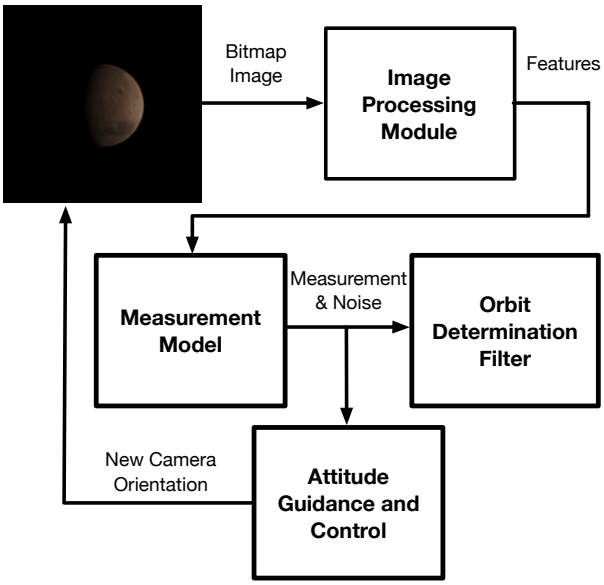


Figure 4.2: Information Flow Between Modules

Figure 4.2 shows the OpNav FSW algorithms used in order to navigate autonomously off of an image. Classically the image processing module directly feeds the filter measurements which are processed in the measurement update. In this filter architecture, the measurement noise is extracted

from the filter in order to be used for heading and orbit determination purposes. Furthermore, this allows to exchange methods of measurements, or even filter implementations all the while comparing all the other components. This therefore follows the paradigm described regarding testing flight algorithms. Another tuning parameter that helps the filter convergence is the `noiseSF` scale factor which is applied onto the covariance as it is updated according to the measurement noise. This is done as suggest in Reference 27 by not scaling the measurement noise itself but the covariance after the time update.

In this subsection, a circle-fitting algorithm extracts the centroid and the apparent diameter of the planet being observed. Before being used for navigation, they must be translated to relative position of the camera with respect to the body. This is seen in the top block of Figure 4.2 where the image processing extracts centroid and apparent diameter (or limb points in the horizon-based navigation method). These points are then transformed into a spacecraft position measurement through the measurement model of the filter. The noise is also transformed to the proper frame to provide measurement noise to the filter. The estimator can then reconstruct the spacecraft states. Figure 4.2 also pictures the coupled nature of the simulation: the images processed are used for attitude guidance and control, which then generate new images.

Extensive testing of autonomous systems — on the ground and in flight — builds trust in the desired performance of the algorithms. Flight tests, or technology demonstrators, are rare and also require rigorous testing in order to not endanger the main mission. Additional tests can be achieved primarily through simulated environments. This work describes novel results achieved for simulated autonomous optical navigation on orbit about Mars. The coupled nature of the simulation enables simultaneous pointing and orbit determination with dynamic image generation. Navigation is done solely using optical images, and by means of limb or centroid/diameter extraction. This is applicable on a wide range of orbits depending on the camera parameters. Through the implementation of pre-existing algorithms and the development of novel optical navigation methods, a fault detection capability is also introduced in order to test methods in off-nominal cases. This research provides insight into achievable navigation accuracy and image processing methods, as

well as outlier detection and mitigation for mission readiness.

4.3.2 Synthetic Images

This section reminds the simulation used for the following example scenarios. This architecture harnesses two main components: a high-fidelity, faster than real-time, astrodynamics simulation framework *Basilisk*; and a sister software package — *Vizard* — to dynamically visualize the simulation environment.[204] More details are found in Chapter 2 but the general codebase design is reminded in Figure 4.3.

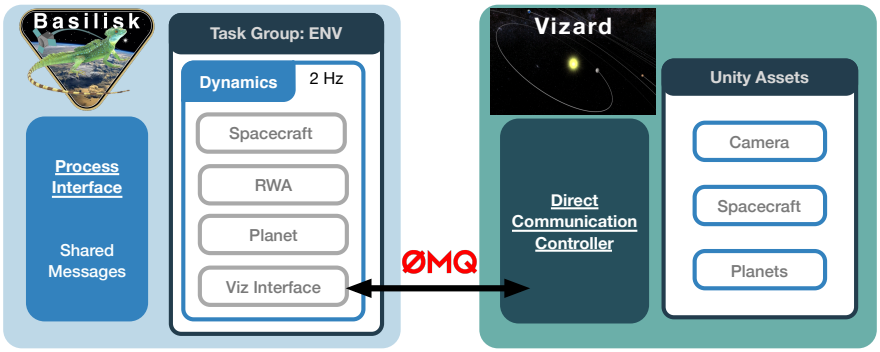


Figure 4.3: General Schematic of Simulation Framework

As a modular astrodynamics simulation framework that allows for the rapid simulation of complex spacecraft dynamics, *Basilisk* provides the environment needed to run and test a wide variety of GNC algorithms. The associated visualization named *Vizard* creates the sensor simulation by emulating a camera and rendering the environment. Here again, the *Basilisk* simulation messages are streamed directly to the visualization to illustrate the spacecraft simulation and environment states.

Figure 4.4 shows the camera model and defines the coordinate frames used widely in the literature[46] and provided by *Vizard*. Once the images are available, different OpNav methods can extract a variety of features from them. Although there is potential for using more computationally intense methods on-board, this research focuses on implementing CAD autonomous OpNav. It provides the necessary information for on-orbit navigation in the current use-case: the celestial

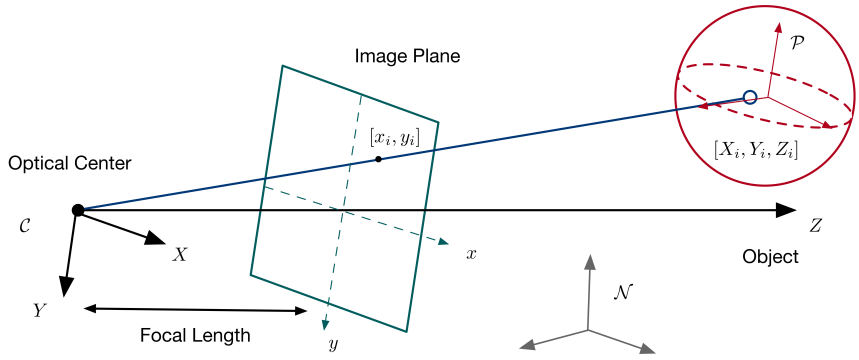


Figure 4.4: Pinhole Camera Model

object is resolved but not taking up the entire field-of-view, and the assumptions described in the introduction of this chapter are verified.

An important transform applied to the pixel measurements is given by the pinhole camera model. Indeed in order to transform the pixels coordinates (x_i, y_i) to the image plane coordinates (x, y) with the following expression:[135]

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{f}{d_x} & -\frac{f^2\alpha}{d_x d_y} & f^2\frac{\alpha v_p - d_y u_p}{d_x d_y} \\ 0 & \frac{f}{d_x} & -\frac{f v_p}{d_x} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{4.1}$$

where f is the focal length, d is the pixel pitch (in the x or y direction according to the subscript), α is the detector array skewness, and (u_p, v_p) is the center point of the image. In this study, $d_x = d_y$ and $\alpha = 0$.

The images are rendered in a *Unity* camera simulator in realistic lighting conditions using its integrated GPU ray-tracing capability. Camera specifications, such as resolution and focal length, are generated in the camera module as well. The simulation also provides corruption capabilities through a *Basilisk* camera module in order to render faulty images which will be explored in the following chapter. It's important to note that images are processed in color, but none of those channels are used. In fact, the *Hough* transform performs a greyscale on the image while the *Canny* transform uses the between pixel intensities without differentiating Red-Green-Blue (RBG)

channels.

The images are once again generated in a closed-loop manner: the spacecraft position and attitude determine the image generation. Furthermore attitude and trajectory control update the images live. This needs to be done in a fast and accurate way. The difficulty comes in providing speeds for several-orbit simulations that tightly couple in attitude variations for image generation. The simulation with a 0.5 second time-step taking 512×512 every minute for attitude control, alongside dynamics models, flight software algorithms, and OpNav OD algorithms during a 10 hour long orbit approximately one minute ($\sim 600\times$ real-time).

Table 4.2: Camera Parameters

$\boldsymbol{\sigma}_{\mathcal{CB}}$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$
${}^{\mathcal{B}}\boldsymbol{r}_{\mathcal{CB}}$[m]	$\begin{bmatrix}0 & 0.2 & 0.2\end{bmatrix}^T$
Resolution (pixels)	$\begin{bmatrix}512 & 512\end{bmatrix}^T$
Sensor Size (mm)	$\begin{bmatrix}10 & 10\end{bmatrix}^T$
Field of View (°)	$\begin{bmatrix}40 & 40\end{bmatrix}^T$

Throughout this chapter, the camera frame is noted $\mathcal{C}$, the spacecraft body frame is $\mathcal{B}$, while the inertial frame is $\mathcal{N}$. Direction cosine matrices are noted $[\mathcal{BN}]$ to represent the rotation from the inertial frame to the body frame which can also be represented a Modified Rodrigues Parameters (MRPs) $\boldsymbol{\sigma}_{\mathcal{BN}}$. The rotation rate of the body frame relative to the inertial frame is noted $\boldsymbol{\omega}_{\mathcal{BN}}$ and the left superscript notation indicates the frame a vector is expressed in.[184] Positions are noted $\boldsymbol{r}$, subscript $\mathcal{CB}$ represents the camera position relative to the spacecraft center of the frame B, similarly, $\boldsymbol{r}_{\mathcal{BN}}$ is the vector from the center of $\mathcal{N}$ to the spacecraft body frame.

The camera parameters are given in Table 5.3 and use a square image with a wide field of view. The sensor size of 1cm is, equivalent to choosing a focal length of 1.373cm. The position and orientation of the camera are arbitrary in this scenario, as long as they don't create any self-shadowing. This is avoided easily with the parameters implemented.

4.3.3 Simulated Astrodynamics

The scenario simulates a spacecraft on an elliptical orbit around Mars. The conditions are designed to allow for variation in the apparent size of the planet. This section specifies the simulation parameters as well as the flight-software algorithms used.

The simulation uses *SPICE*[1] data, where the simulation begins on December 12th at 22:00 (GMT) 2019. Given the initial conditions of the spacecraft in orbit — seen in Table 5.2 — Mars first appears as a waxing crescent and as the spacecraft reaches apoapse Mars becomes full. Near the end of the simulation Mars begins to go through a waning crescent phase. This allows the FSW algorithms to be tested along a wide variety of lighting conditions and planet sizes. The

Table 4.3: Spacecraft Initial States

$\boldsymbol{\sigma}_{\mathcal{BN}}$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$
$\boldsymbol{\omega}_{\mathcal{BN}}$[rad/s]	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$
Orbital Elements $(a, e, i, \Omega, \omega, f)$	$(18000\text{km}, 0.6, 10°$ $25°, 190°, 80°)$

simulation modules assigned to modeling the spacecraft dynamics and environment are described in Table 4.4. These modules simulate spacecraft attitude gyroscopics and gravity,[5] eclipse, reaction wheels,[3] and star trackers. Eclipsing is also modeled: this usually creates a halt in the spacecraft measurements. In a fully coupled Attitude-OD simulation this means the spacecraft enters a search mode, which intends to allow for the modeling of a fully independent spacecraft. In this scenario, a rough pointing to Mars can be accomplished even with inaccurate knowledge of the spacecraft position. Therefore, in order to focus on the OD solution, the spacecraft goes through a 3 minute guided pointing mode before starting to navigate and point using OpNav.

Table 4.4 defines the spin axes of the wheels[184] in the body frame through the elevation and azimuth angles with the equation:

$$^{\mathcal{B}}\hat{\boldsymbol{g}}_s = \begin{bmatrix}\cos(el)\cos(az) & \cos(el)\sin(az) & \sin(el)\end{bmatrix}^T \tag{4.2}$$

[1] `naif.jpl.nasa.gov/naif/`

Table 4.4: Simulation Parameters

Simulation Modules Instantiated	**Necessary parameters at initialization**
Spacecraft Hub	Inertia $[I] = \text{diag}(900, 800, 600)$ kg·m^2 mass $M = 750$kg
Gravity Effector/Eclipse	December 12th 2019 at 18:00:00.0 (Z) $\mu_{\text{mars}} = 4.283 \cdot 10^4$, $\mu_{\text{earth}} = 0.399 \cdot 10^6$, $\mu_{\text{jup}} = 1.267 \cdot 10^8$, $\mu_{\text{sun}} = 0.327 \cdot 10^{11}$ [km^3/s^2]
Simple Navigation Star Tracker	Attitude error $\sigma_{\text{att}} = 1/3600^\circ$ Rate error $\sigma_{\text{rate}} = 5 \cdot 10^{-5\circ}$/s
Reaction Wheel Effector	4 Honeywell HR16 Wheels
Wheel orientations	Elevation Angles (el): 40° Azimuths angles (az): $45^\circ, 135^\circ, 225^\circ, 315^\circ$ Pos in $\mathcal{B}$ [m]: $\big[0.8, 0.8, 1.79070\big]^T$ $\big[0.8, -0.8, 1.79070\big]^T$ $\big[-0.8, -0.8, 1.79070\big]^T$ $\big[-0.8, 0.8, 1.79070\big]^T$

More specific information on the wheel specifications are available on *Honeywell* documents[1]. Table 4.5 implements methods from the *OpenCV*[2] library and more information is found in their code documentation.

Beyond the pure simulation modules, the simulation also implements several FSW algorithms. These are all developed in C for speed, and compatibility with heritage FSW. These are broken up in two groups: imaging FSW (summarized in Table 4.5) and Pointing/OD FSW (Table 4.5, 4.6). The imaging modules encompass the OpNav raw measurements (limbs and circles) as well as the measurement models to provide spacecraft position.

Table 4.6 shows the modules implemented for centroid-based pointing guidance,[204] OD, and control using MRP-Feedback seen in Example 8.14 of Reference 184.

4.4 Image Processing and Filtering

This section describes the details of the OpNav data chain from captured image to orbit estimate. This is done more modularly by taking the measurement model outside of the filter in order to more easily interchange models. The flow of data through the simulation is shown at a

[1] `aerospace.honeywell.com`

[2] `docs.opencv.org/`

Table 4.5: Flight Software for Imaging

Flight Software Modules Instantiated	**Necessary parameters at initialization**
Image Processing (arguments for *Hough Circle* method)	param1 = 300, param2 = 20, minDist = 50 pix minRadius = 20 pix, dp = 1,voteThresh = 25
Limb Finding (arguments for *Canny* transform)	cannyThreshLow = 50, cannyThreshHigh = 100 blurSize = 5
Pixel Line Transform	Planet Target is Mars
Horizon Nav	noiseSF = 70 , Planet Target is Mars

high level in Fig. 4.2, and is summarized in Table 4.5.

4.4.1 Limb Detection

In recent years, with interest in applying autonomous navigation around the Moon,[45] high-fidelity image processing algorithms have been developed. With the primary application to spherical or ellipsoidal bodies, these revolve mostly around finding a ellipse centroid,[152] or fitting a limb.[48] These methods provide accurate measurements of a spacecraft's relative position to the body provided an unambiguous limb.

The baseline method chosen to measure spacecraft position is the Non-Iterative Horizon-Based Optical Navigation by Singular Value Decomposition[43,50] (NIH-SVD). This method was chosen for its high performance not only analytically but also numerically. The method takes a set of limb-points as an input and outputs a camera position in the planet frame. The algorithm is briefly summarized here.

Table 4.6: Flight Software Pointing and Orbit Determination

Flight Software Modules Instantiated	**Necessary parameters at initialization**
OpNav Point	minAngle = 0.001°, timeOut = 100s $\omega_{\text{search}} = [0.06, 0.0, -0.06]°/\text{s}$, ${}^{\mathcal{C}}h_c = [0, 0, 1]\text{m}$
relativeOD	$\alpha = 0.02$, $\beta = 2$, $\kappa = 0$, noiseSF = 5 $\boldsymbol{r}_{\text{error}} = [10, 10, -10]\text{km}$ $\boldsymbol{r}_{\text{error}} = [0.1, -0.01, 0.01]\text{km/s}$
MRP Feedback RW	K = 3.5 , P =30 (no integral feedback)
RW motor Torque	Control axes are ${}^{\mathcal{B}}[\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3]$

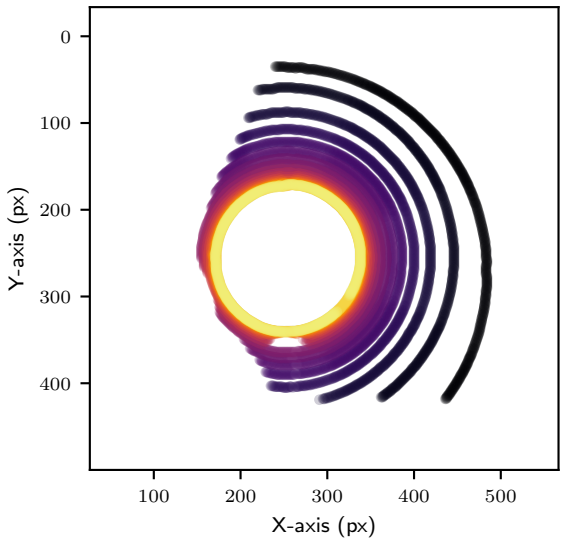


Figure 4.5: Every 30$^{\text{th}}$ Mars Limb Fit

Assume a set of N limb point vectors $\boldsymbol{s}_k = (x_k, y_k, 1)$ (km) are given in the camera frame and in the image plane (normalized focal length). Pairs (x_k, y_k) are the position of the k^{th} limb point in the camera frame.

$$[B] = [Q_m][\mathcal{CP}] \tag{4.3}$$

$$\bar{\boldsymbol{s}}_k = [B]\boldsymbol{s}_k \tag{4.4}$$

$$\bar{\boldsymbol{s}}'_k = \frac{\bar{\boldsymbol{s}}_k}{||\bar{\boldsymbol{s}}_k||} \tag{4.5}$$

Where $[Q_m] = \text{diag}\left(\frac{1}{r_a}, \frac{1}{r_b}, \frac{1}{r_c}\right)$ in the planet-fixed frame, with r_a, r_b, r_c are the radii of the potentially ellipsoidal body along it's principal axes. $[B] = [Q_m][CP]$, where P is the planet frame. In

this chapter the planet frame is taken as the inertial frame N, and $[Q_m]$ represents a circular Mars with $r_a = r_b = r_c = 3396.19$km. After rotation and normalization according to Equations 4.6, these are concatenated in order to solve for

$$[H] = \begin{bmatrix} \bar{\boldsymbol{s}}_0'^T \\ \vdots \\ \bar{\boldsymbol{s}}_N'^T \end{bmatrix} \tag{4.6}$$

$$[H]\boldsymbol{n} = \mathbf{1}_{N\times 1} \tag{4.7}$$

This is done by performing a QR decomposition on $[H]$ which constructs $[Q_H]$ orthonormal and $[R_H]$ upper triangular such that $[H] = [Q_H][R_H]$. This leads to the equation $[R_H]\boldsymbol{n} = [Q_H]^T\mathbf{1}_{N\times 1}$ which is solved by back-substitution. With $\boldsymbol{n}$ now given, the spacecraft position is given in the camera frame by:

$${}^{\mathcal{C}}\hat{\boldsymbol{r}}_{\mathcal{BN}} = -(\boldsymbol{n}^T\boldsymbol{n} - 1)^{-\frac{1}{2}}[\mathrm{B}]^{-1}\boldsymbol{n} \tag{4.8}$$

The only computation left is to rotate into the desired frames. This is done using the star-tracker estimate of $[\mathcal{BN}]$ and the known camera frame $[\mathcal{CB}]$, and the module outputs the covariance and estimates in the body, inertial, and camera frames for downstream use. This method is implemented in C (under the *Basilisk* module name of `imageProcessing/HorizonNav`). The last thing to do in order to implement this method fully is to create a limb finding method. This is done by using the *Canny* transform[25] implemented in *OpenCV*, preceded by a greyscale transform and a 3×3 pixel Gaussian blur. Figure 4.5 shows the limb points found (every 30 images) and used in this image processing method during the simulated scenarios presented in the later sections of this work. The x and y axes are pixel numbers in the focal plane, and the colors illustrate the time the image was taken: in chronological order from dark to light shades. It illustrates the changing Mars crescent throughout the orbit as the darker larger limbs are only quarter circles, while the brighter points in the center are full circles.

These limb points are extracted in a separate module found in *Basilisk* under the folder: `fswAlgorithms/imageProcessing/LimbFinding`. Given the clean images provided to it during

these simulations, the limbs found accurately hug the planet. The simulation allows up to 2000 limb points to be extracted from a single image. This maximum is only approached for high resolution images (10^6 pixels and over) in which the planet takes up a large portion of the field of view. A difficulty in this method arises when computing the covariance around this estimate. Indeed, computations in Reference 49 show that the covariance is given by

$$[P_r] = \mathbb{E}\left[\delta r \delta r^T\right] = [F][P_n][F]^T \tag{4.9}$$

Where $\mathbb{E}[]$ is the expectation operator, and

$$[F] = -(\boldsymbol{n}\boldsymbol{n}^T - 1)^{\frac{1}{2}}[B]^{-1}\left([I]_3 - \frac{\boldsymbol{n}\boldsymbol{n}^T}{\boldsymbol{n}^T\boldsymbol{n} - 1}\right) \tag{4.10}$$

$$[P_n] = \left([H]^T \mathrm{diag}\left(\sigma_1 \ldots \sigma_n\right)[H]\right)^{-1} \tag{4.11}$$

Where $[R_y] = \mathrm{diag}\left(\sigma_1 \ldots \sigma_N\right)$ is the covariance of the pixel measurement residuals. More information can be found in the reference. The important component of these equations is that in order to compute the $[P]_r$ matrix, $N \times N$ matrices must be computed, and multiplied.

This therefore requires a large matrix multiplication computation which grows as N^2 (N limb points). This is dynamically allocated memory, which in C requires the use of *malloc* and therefore accesses heap memory instead of stack memory. Although not an issue in most applications, some FSW developments are reticent to use these function calls in flight. This is because of the potential memory leaks that can occur if heap memory is not freed, and from the potential hazard of having erroneous inputs lead to requests for more memory than can be provided by the hardware.

4.4.2 Hough Circle Detection

The novel method for OpNav discussed in this section uses the *Hough Circle* transform. This method is routinely used in robotic applications[197] where measurements are plentiful. This development attempts to apply some of these paradigms to spacecraft navigation.

The purpose of this image processing method is to find objects within a certain class of shapes by a voting procedure. The procedure is carried out in a parameter space, and in the case of

circles the parameter space is three-dimensional: X component of the center, Y component of the center, and radius. Circle candidates are obtained as local maxima in this accumulator space that is explicitly constructed by the algorithm. Therefore the algorithm builds a point cloud of possible circles in the parameter space, and a maximum, or points above a threshold can be extracted. Gradient detection, which extracts edges from the image, helps to reduce the search space. This aids the computation time and reduces the number of extraneous votes, which is implemented in *OpenCV*.

In the *Basilisk Hough Circle* implementation, a geometrical method is used to extract pose information from center and apparent diameter information. The norm of the position vector is given by the apparent size, it's direction is given by the pixel and line data. Using ${}^{\mathcal{C}}\boldsymbol{r}_{\mathcal{BN}} = {}^{\mathcal{C}}\begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}^T$ as the relative vector of the camera with respect to the celestial center, A as the apparent diameter of the celestial body, D as the actual diameter:

$$|\boldsymbol{r}_{\mathcal{BN}}| = \frac{1}{2}\frac{D}{\sin\left(\frac{1}{2}A\right)} \tag{4.12}$$

$$\frac{1}{r_3}{}^{\mathcal{C}}\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \frac{1}{r_3}{}^{\mathcal{C}}\tilde{\boldsymbol{r}}_{\mathcal{BN}} = \frac{1}{f}{}^{\mathcal{C}}\begin{bmatrix} x \\ y \end{bmatrix} \tag{4.13}$$

These equations have been used in multiple instances in studies by Battin or Owen.[13,167] The third component of $\boldsymbol{r}_{\mathcal{BN}}$ provides the range measurement to the body which can be extracted using the apparent diameter measurements. Hence the definition of $\tilde{\boldsymbol{r}}_{\mathcal{BN}}$ which only contains the first two components of $\boldsymbol{r}_{\mathcal{BN}}$. The vector components of $\boldsymbol{r}_{\mathcal{BN}}$ are expressed relative to the inertial frame assuming inertial attitude knowledge from other instruments. Using the position of the camera on the spacecraft, this provides the measurement value for an orbit determination filter using a circle-finding algorithm.

In the case of the geometric formula, the partials allow to quantify error due to the camera specifications. Indeed, if X, Y are the pixel sizes (in their respective directions), x, y are the position

on the camera, and x_i, y_i, ρ_i are the pixel values for the CAD measurements:

$$ {}^{\mathcal{C}}\tilde{\boldsymbol{r}}_{\mathcal{BN}} = \frac{r_3}{f} {}^{\mathcal{C}}\begin{bmatrix} x \\ y \end{bmatrix} = \frac{r_3}{f} {}^{\mathcal{C}}\begin{bmatrix} x_i X \\ y_i Y \end{bmatrix} \tag{4.14} $$

$$ |\boldsymbol{r}_{\mathcal{BN}}| = \frac{1}{2}\frac{D}{\sin\left(\frac{1}{2}A\right)} \tag{4.15} $$

$$ = \frac{1}{2}\frac{D}{\sin\left(\arctan\left(\frac{\rho}{f}\right)\right)} \tag{4.16} $$

$$ = \frac{1}{2}\frac{D}{\sin\left(\arctan\left(\frac{\rho_i X}{f}\right)\right)} \tag{4.17} $$

Eq. (4.14) provides a simple partial with respect to the center measurement ${}^{\mathcal{C}}\boldsymbol{c}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^T$

$$ \frac{\partial \tilde{\boldsymbol{r}}_{\mathcal{BN}}}{\partial \boldsymbol{c}_i} = r_3 \begin{bmatrix} \frac{X}{f} & 0 \\ 0 & \frac{Y}{f} \end{bmatrix} \tag{4.18} $$

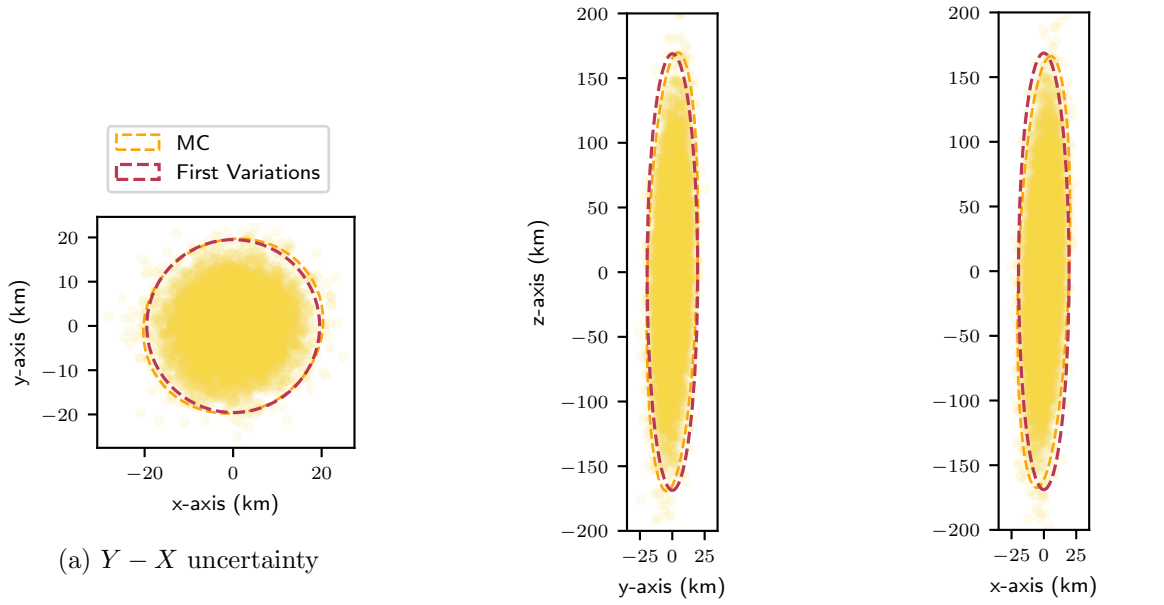
$$ \Rightarrow \mathbb{E}\left[\delta\tilde{\boldsymbol{r}}_{\mathcal{BN}}\delta\tilde{\boldsymbol{r}}_{\mathcal{BN}}^T\right] = r_3^2 \begin{bmatrix} \frac{X}{f} & 0 \\ 0 & \frac{Y}{f} \end{bmatrix} \left[\delta\boldsymbol{c}_i\delta\boldsymbol{c}_i^T\right] \begin{bmatrix} \frac{X}{f} & 0 \\ 0 & \frac{Y}{f} \end{bmatrix} \tag{4.19} $$

The partial for Eq. (4.15) is:

$$ \frac{\partial|\boldsymbol{r}_{\mathcal{BN}}|}{\partial\rho_i} = \frac{Dd_x}{2}\sqrt{f^2+\rho^2 d_x^2}\left(\frac{1}{f^2+\rho^2 d_x^2} - \frac{1}{\rho^2 d_x^2}\right) \tag{4.20} $$

Equation 4.20 is validated by Monte-Carlo analysis and compared as well to an unscented transform in Figure 4.9. This shows $10,000$ points propagated through Equation 4.12 using the camera parameters in Table 5.3 for a range of $18,000$km with Mars offset from the image center by $(23, 19)$ pixels. The pixel standard deviations used are $\sigma_x = \sigma_y = 0.5$ and $\sigma_\rho = 2$. Figure 4.9 shows good accordance of the first variations to the Monte Carlos. The image processing methods used here for center and apparent diameter are *Hough Circle* transforms[171] instantiated with the open-source computer vision library *OpenCV*. Given the scenario in which it is applied — orbit around a known spherical celestial body — the *Hough Circle* Transform provides a robust solution.

Figure 4.7 shows every 30^{th} circles found in the scenario using *Hough Cirlces* to fit Mars. Similarly to Figure 4.5, the x an y axes are pixel numbers in the focal plane, and the colors illustrate



(a) $Y - X$ uncertainty

(b) $Z - Y$ uncertainty

(c) $Z - X$ uncertainty

Figure 4.6: Partial Validation

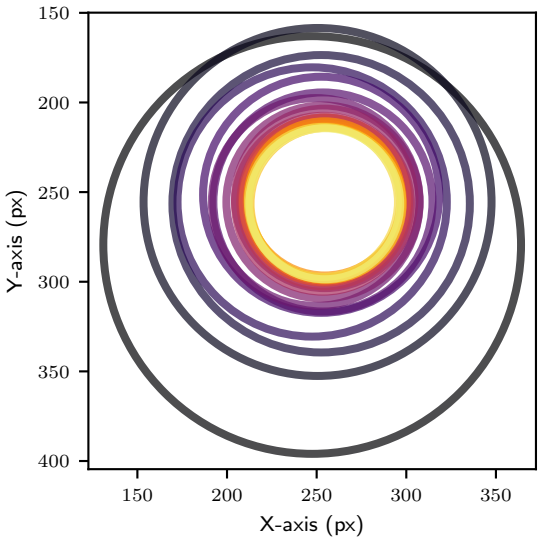


Figure 4.7: Every 30^{th} Mars Circle Fit

the order of the image capture: from dark to light shades. It is immediately seen that the variably of this method is greater than that of the limb extraction. Yet under the assumption of frequent

images, when the variations are mostly Gaussian in nature they can be handled through filtering. In practice, the circles are given in a frame centered at the top-right corner of the frame, and the pixel size isn't directly given. Therefore the computation for the planet direction is given using $X = \frac{\text{SensorSize}_x}{\text{Resolution}_x}$ and $Y = \frac{\text{SensorSize}_y}{\text{Resolution}_y}$ in mm/pixel:

$$^{\mathcal{C}}\hat{\boldsymbol{r}}_{\mathcal{BN}} = -\begin{bmatrix} \frac{X}{f} \cdot (\mathrm{x}_i - \frac{\text{Resolution}_x}{2} + \frac{1}{2}) \\ \frac{Y}{f} \cdot (\mathrm{y}_i - \frac{\text{Resolution}_y}{2} + \frac{1}{2}) \\ 1 \end{bmatrix} \tag{4.21}$$

Where the $\frac{1}{2}$ centers the point on the activated pixel. The normalization by the focal length f functionally brings the pictured spacecraft onto the image plane. This value is then scaled by $|\boldsymbol{r}_{\mathcal{BN}}|$ computed in Equation 4.15, and rotated into the desired frames. This is done using the star-tracker estimate of $[\mathcal{BN}]$ and the known camera frame $[\mathcal{CB}]$.

4.5 Coupled Pointing Guidance and Orbit Determination

This section analyses the performance of a spacecraft on an elliptical orbit around Mars. The initial conditions and simulations parameters are described and discussed in Tables 5.2-4.6. This section shows the orbit error solution convergence, as well as nominal performance of other flight-software algorithms.

The results showcase the possible results for autonomous OpNav with a spacecraft taking numerous pictures: 1 image per minute, using Hough-Cirlces. This approach also enables coupled pointing on a wide range of orbits and permits less accurate methods to perform despite noisy measurements. All of the results shown have the spacecraft perform both duties, given initial conditions that provide the planet in the field of view.

First the measurements are analyzed, then the filter parameters are listed, and finally the results are presented.

4.5.1 Measurements for Orbit Determination

The measurement noise is extracted from the images and rotated into the proper frame. Figure 4.8 shows the measurement quality for each of the methods by comparing the measurements directly to the truth value of the spacecraft position in the camera frame. Both of these methods show that the error in the measurements stem primarily from the ranging problem. The Z direction in the camera frame is the direction that suffers from the most errors in both methods as both algorithms present more sensitivity to a slight variation in apparent planet size. It should be noted that although the *Hough Circle* measurements provide more noise (seen in Figure 4.8a), it is not variable over the orbit and can be handled well by the filter. Figure 4.8b shows the errors from the limb-fitting method. A signal appears in these measurement residuals — which can also be seen in the Hough transformation. This signal is periodic with the orbit and changes only with the lighting conditions: if the orbit is exactly the same but the light source is displaced, the signal changes periodically.

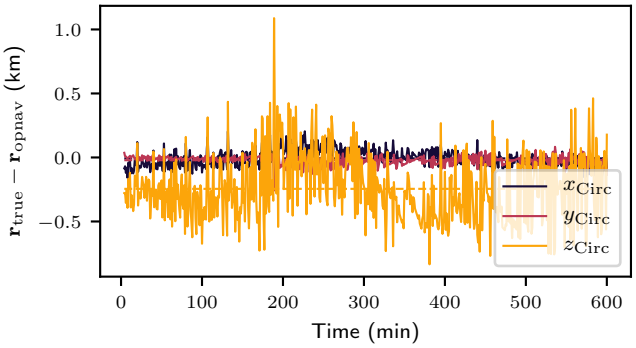
As the limbs and the circles are found using only the illuminated pixels, if these are scarce and only on a side of the planet, the measurement can suffer from a slight offset. The errors are, in both cases not necessarily due to the OpNav transforms used, but rather to the image processing method implemented.

The filter also overestimates the noise on the measurements by a factor of 5 in order to get the desired results. This is equivalent to modifying the scale factors on the following measurement noises, but provides the filter with noise scaling control. These noise values are selected accordingly:

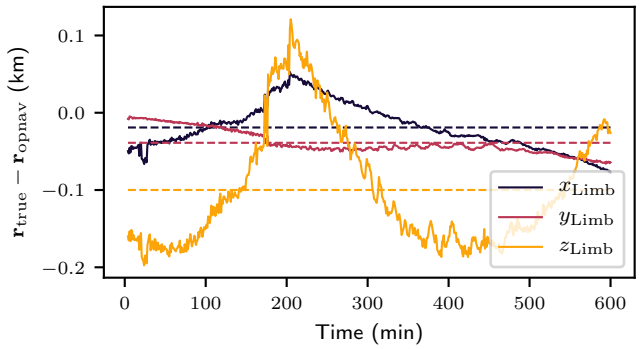
- The limb-finding algorithm takes as its pixel uncertainty the ratio of the camera width to the number of limb-points found (N), multiplied by 70. The scaling is an empirical value in order to get the correct order of magnitude for the noise:

$$\sigma_{\text{pix}} = 70\frac{\text{Resolution}_x}{N}$$

- The *Hough Circle* algorithm uses the voting system inherent to the Hough transform. It



(a) *Hough Circle* method



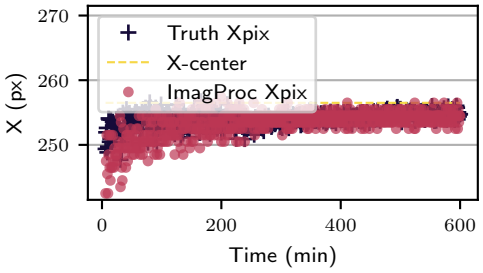
(b) Limbs and NIH-SVD

Figure 4.8: Measurements from Both Methods in Camera Frame

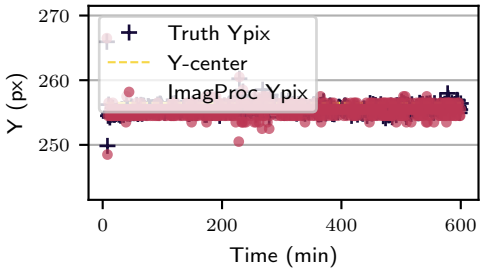
takes the ratio of the votes accumulated for the circle by the vote threshold:

$$\sigma_{\text{pix}} = \frac{\text{votes}}{\text{voteThresh}}$$

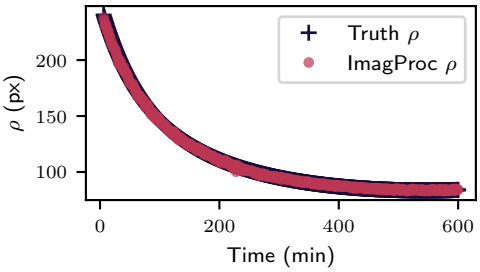
The *HoughCircle* algorithm, is compared to the expected center pair, and radius of the planet in pixels. These truth values are computed with the camera parameters and true spacecraft position and attitude. These results show once again how the data, despite the fact that it is noisy, provides good results next to the truth values. The limb-finding algorithm, generates a noise given the number of limb points found, which are plotted in Figure 4.10. These show the changing lighting conditions. These show that there is not a direct correlation between the number of pixels found and the measurement accuracy. If the planet is closer (as seen in the start of the scenario) there



(a) X-pixels



(b) Y-pixels



(c) Planet radius

Figure 4.9: Pixels Found Compared to Expected Measurements

are more limb-pixels found, whereas a more distant full disk might provide a better measurement with less pixels. The variations do, nonetheless provide information on the quality of the image detection, and a threshold can be applied before which the data is not trusted.

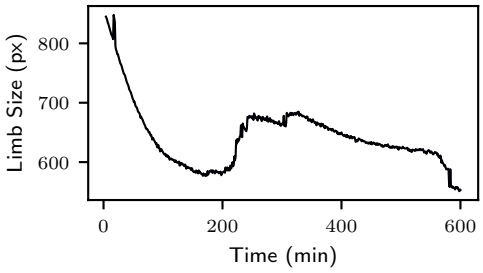


Figure 4.10: Limb Points Found in Images

4.5.2 Filter Implementation

Now that two methods have been developed to extract measurements from images, they can be filtered in a relative OD estimator. The field of statistical orbit determination provides a series of potential solutions for on-board, sequential, state determination.[12] In this thesis, the same filter implementation is used for both methods, with the same values for process noise:

$$[Q] = \text{diag}(10^{-6}\text{m}^2, 10^{-6}\text{m}^2, 10^{-6}\text{m}^2, 10^{-8}\text{m}^2/\text{s}^2, 10^{-8}\text{m}^2/\text{s}^2, 10^{-8}\text{m}^2/\text{s}^2) \tag{4.22}$$

Both the methods described provided relative position measurements for the orbit determination filter. This filter which estimates spacecraft position and velocity in the inertial frame is implemented as a square-root unscented Kalman filter.[210] This filter is used both for alongside star tracker measurements for inertial attitude, and in conjunction with heading determination filters using the planet's centroid for relative attitude. The filter state is

$$\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{N}}\boldsymbol{r}_{\mathcal{BN}} \\ {}^{\mathcal{N}}\dot{\boldsymbol{r}}_{\mathcal{BN}} \end{bmatrix} \tag{4.23}$$

Where ${}^{\mathcal{N}}\boldsymbol{r}_{\mathcal{BN}}$ is the spacecraft position relative to the celestial body (Mars). The 'dot' represents a derivative as seen by the inertial frame.[184] This keeps the estimation process minimal, though other states could be added if onboard applications allow.

$$\dot{\boldsymbol{X}} = F(\boldsymbol{X}) = \begin{bmatrix} \dot{\boldsymbol{r}}_{\mathcal{BN}} \\ \ddot{\boldsymbol{r}}_{\mathcal{BN}} \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{r}}_{\mathcal{BN}} \\ -\frac{\mu}{|\boldsymbol{r}_{\mathcal{BN}}|^3}\boldsymbol{r}_{\mathcal{BN}} \end{bmatrix} \tag{4.24}$$

The dynamics of the filter are given in Equation (4.24). The state propagation is done using an RK4 integrator. The following square-root uKF coefficients are used: $\alpha = 0.02$, and $\beta = 2$. These allow to vary the Gaussian nature of the noise. The filter does not know about the influence of other gravitational bodies. These only represent slight perturbations[202] which are not perceived given the measurement errors as well as the measurement density.

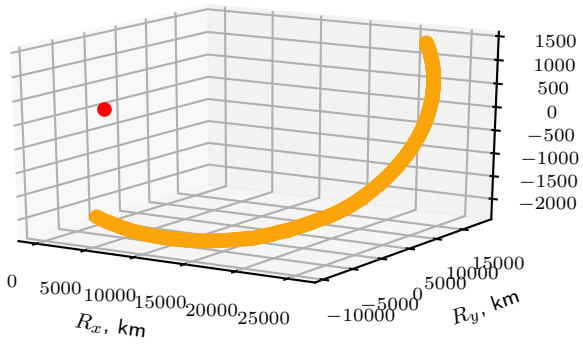


Figure 4.11: Orbit Visualized with Measurements

The measurement model is simple given the pre-processing done by the two methods described. This is in practice, equivalent to extracting the measurement model from the filter codebase. Therefore the measurements model in the filter is:

$$G(\boldsymbol{X}) = {}^{\mathcal{N}}\boldsymbol{r}_{\mathcal{BN}} \tag{4.25}$$

This is done to simplify the upkeep and modularity of the filter for OpNav, but also to be able to use different types of measurements for one same OD estimate, as was explained in prior sections.

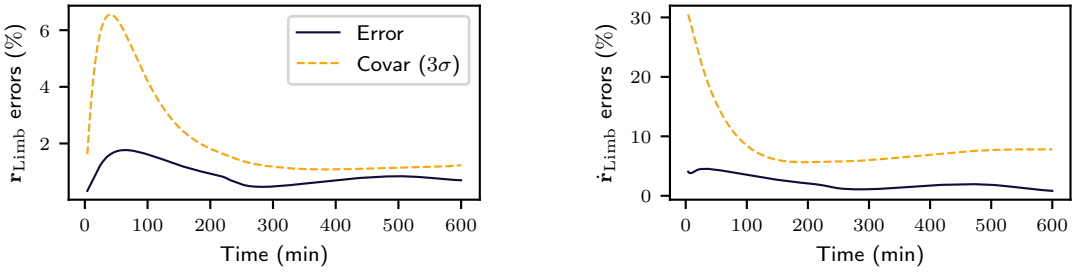
4.5.3 Filter Results

Figures 4.12 and 4.14 show the filter results for the limb fitting method coupled with the NIH-SVD algorithm. All percentages are computed as the norm of the difference between the estimate and the truth, normalized by the truth norm, multiplied by 100. The covariance on the $x-y$ components oscillates around 200km ($\sim 1.3\%$) error on position and 0.05km/s ($\sim 7\%$) error on velocity, with slightly better performance in the out-of-plane direction. The state estimates have errors in percentages below 0.9% on position and below 2% on velocity. These errors oscillate depending on the lighting conditions, which is seen as well in the results for the *Hough Circles*. In both of these results, the rising covariance on the velocity and position are driven by the elliptical nature of the orbit (true velocity decreases and relative distance increases).

The *Hough Circle* method performs well overall, as seen in Figures 4.13 and 4.15, given

the simplicity of the algorithm and the sensitivity of the geometrical position computation. This scenario, presents an ideal case for both methods, as the camera doesn't contain any artifacts, noise, or errors. Robustness to such corruptions will be seen in the next chapter.

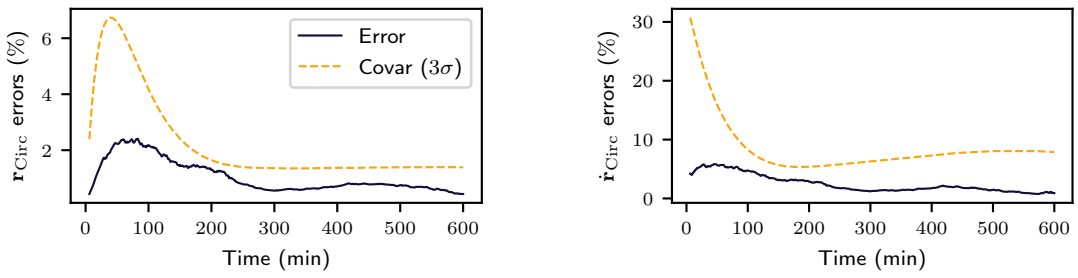
The limb extraction paired with NIH-SVD barely outperform it, given that the *Hough Circle* covariances range between 200 and 250km ($\sim$ 1.8%) error on position and 0.05km/s ($\sim$ 8%) on velocity. The state estimates have errors in percentages below 1% on position and below 2% on velocity. These errors are comparable and nearly identical to those of the more sophisticated method. Furthermore, *Hough Circles* does not require a lot of memory thanks to the reduced search space described using edge detection, and is easy to implement on smaller spacecraft that may not need the highest levels of fidelity on the covariance values. The main way to simplify the *NIH-SVD* method would be to not compute the full covariance, which is often not preferred. Furthermore, if the interest is solely on autonomous pointing using images, *Hough Circles* provide a more lightweight algorithm which performs as well as the state-of-the-art algorithm.



(a) Inertial Position Error Norm

(b) Inertial Velocity Error Norm

Figure 4.12: Relative Errors — *NIH-SVD*



(a) Inertial Position Error Norm

(b) Inertial Velocity Error Norm

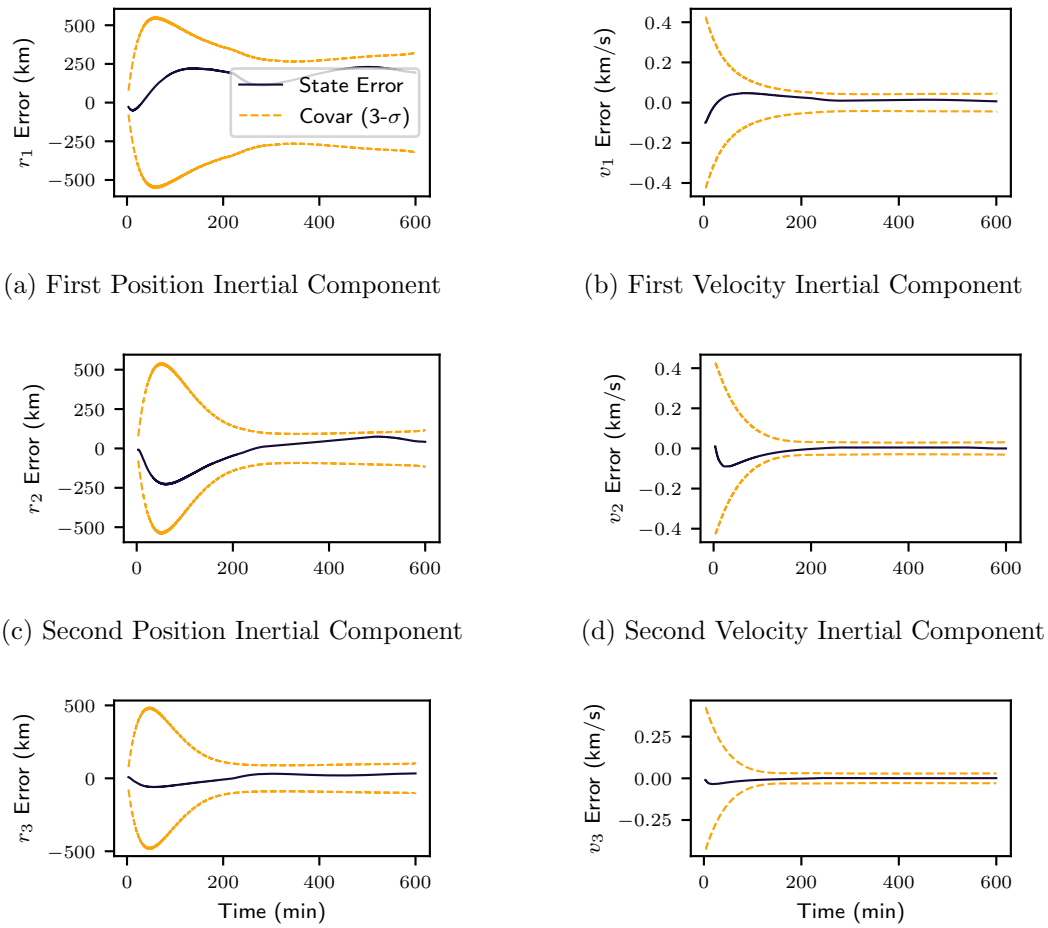
Figure 4.13: Relative Errors — *Hough Cirlces*

It is important to remember that the use-case of *NIH-SVD* is broader than the *Hough Circle* algorithm in that it generalizes to oblate spheroids directly. Therefore navigation about gas giants or other oblate planets with OpNav would certainly push the results in favor of *NIH-SVD* with a high-enough resolution camera. Nonetheless, in the case of telluric planets or moons, the novel *Hough Circle* results present promising results.

Both the methods show a slip outside of the covariance bounds in the early stages of the imaging process. This is seen on the second position component, and is due to the lighting conditions and the spacecraft position at that time. Overall, the results of both methods are promising, especially given the simplicity of the limb finding algorithm. Indeed, sub-pixel edge localization algorithm using Zernike moments[43] would allow for better results with the same images.

The post fit residuals for the *HoughCircle* method are seen in Figure 4.16. The filter goes through a transient paired with difficult lighting conditions, but the residuals are brought back to noise once enough measurements are processed. These show the expected performance from the filter, and the quality of the noise estimates from the images. This is seen notably by the oscillations in the noise which match with the noise extracted from the actual measurements.

The aggregate of these results show that both methods are implemented with success, and presents the *Hough Circle* algorithm as a viable navigation method given the camera, planet of interest, and orbit. This is notably achieved through a relatively fast imaging rate, which provides



(a) First Position Inertial Component

(b) First Velocity Inertial Component

(c) Second Position Inertial Component

(d) Second Velocity Inertial Component

(e) Third Position Inertial Component

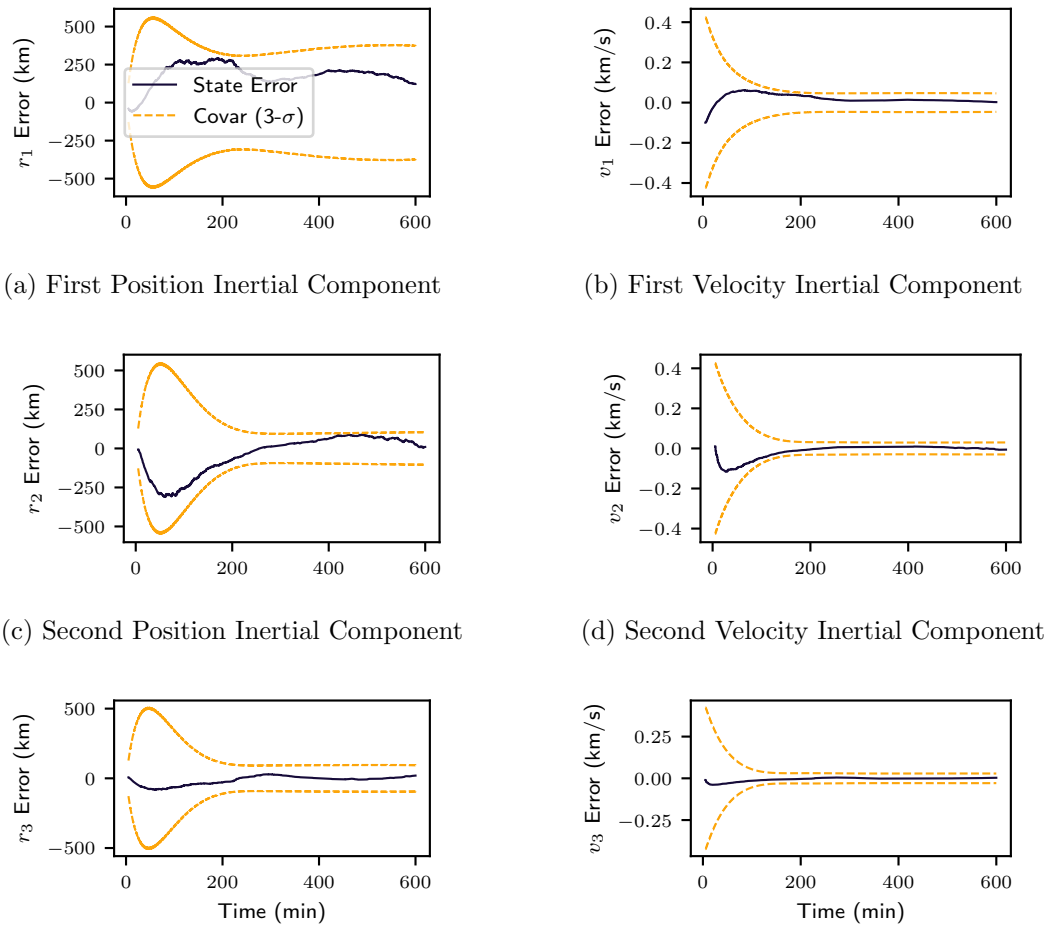
(f) Third Velocity Inertial Component

Figure 4.14: State Error and Covariance Plots — *NIH-SVD*

sufficient measurements to filter the noise that can be found in the processing step.

4.6 Monte-Carlo Analysis

Part of the effort put into the development of the software package to simulate complex dynamics was driven by the desire to create a general and fast simulation. The need arises from mission design and analysis. Indeed, most mission choices are tested and often designed through the use of Monte-Carlo analysis. Uses include but are not limited to: pointing analysis,[103] thermal analysis,[33] debris propagation,[176] etc. Monte-Carlo capability may also permit higher level analysis,



(a) First Position Inertial Component

(b) First Velocity Inertial Component

(c) Second Position Inertial Component

(d) Second Velocity Inertial Component

(e) Third Position Inertial Component

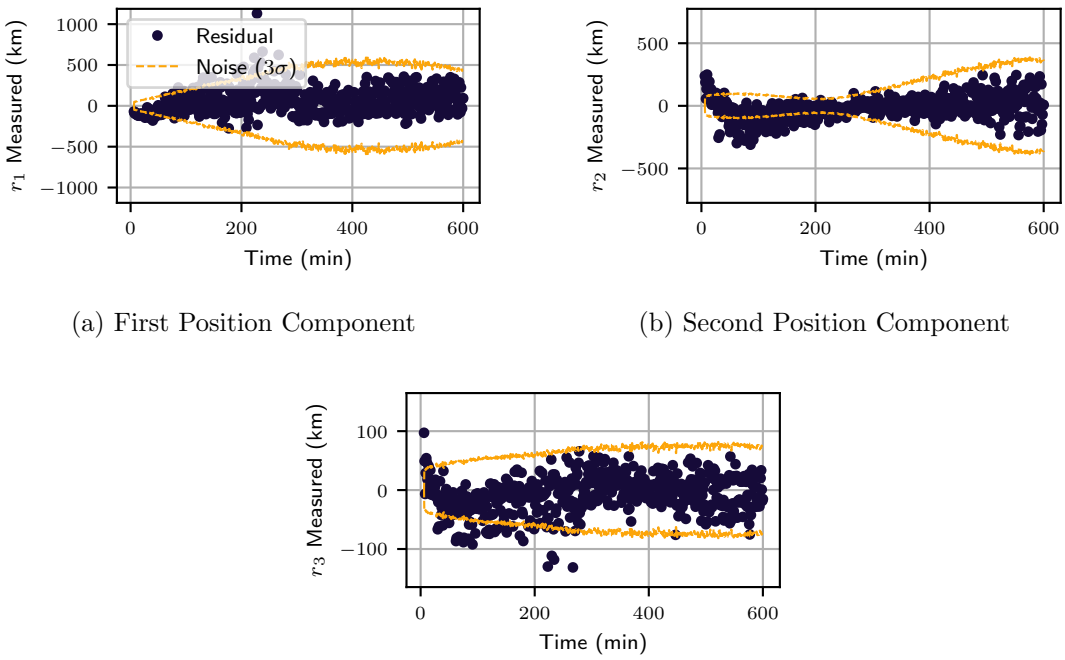
(f) Third Velocity Inertial Component

Figure 4.15: State Error and Covariance Plots — *Hough Cirlces*

notably the analysis of sensitive parameters[153] through meta-model design.[192]

Reference 192 explains the applicability of Design and Analysis of Computer Experiments Techniques (DACE) to aerospace. DACE is an adaptation for simulations of the DOE approach (Design of Experiments) used in physical experiments on industrial processes. DOE is a statistical method of experimentation that varies all inputs in a simulation simultaneously (not one factor at a time) and achieves the following:

- Determines the critical inputs (those with biggest effect on output of interest)
- Quantifies the input/output relationships in an analytical form in the experimental range



(a) First Position Component

(b) Second Position Component

(c) Third Position Component

Figure 4.16: Post Fit Residuals for the *Hough Cirlces*

- Shows interactions between inputs

The method of DACE[84] has been used extensively in the automotive and other manufacturing sectors, but so far not in astrodynamics. DACE is used to augment the limited number of runs of a simulation by fitting an approximate statistical model, a surrogate or Meta-Model, based on a set of limited observation data acquired by running the simulation at carefully selected design points, generated from a Space Filling design. These developments illustrate the ongoing work on Monte-Carlo improvements and the need for faster simulations to produce required data. The following simulations display capabilities that could harness such methods in the future, though do not currently use it for sensitivity analysis.

Since there is no direct requirement to be met in this scenario, no direct validation can be produced. Despite this boundless analysis, the sensitivity of parameters can be explored. The notable topics of importance in this section are: the orbit initial conditions (applying the solutions

to a wide range of orbits), filter parameter modifications (noise scaling applied to the relative OD SRuKF), errors in the camera parameters (errors in focal lengths or field of views). In the following subsections, the orbit parameters, camera field of view, and filter noise scale factor are always dispersed. The values of the dispersions are detailed in the specific subsections as they focus on one fo the specific studies.

4.6.1 Dispersions on Orbital Parameters

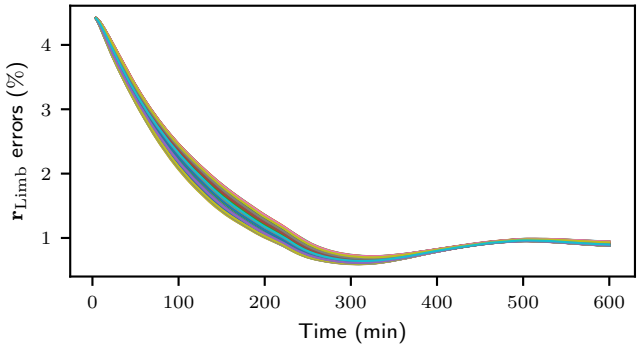
The first sensitivity analysis examines the orbit sensitivities of the scenario. This is done by changing the orbit by adding dispersions to the orbital parameters. These dispersed values are then turned back to position and velocity components in order to initialize the simulation. The goal of this analysis is to ensure that there is not an inherent tie between the orbit chosen and the results. In this regard, a large orbit variation of $3,000$km around the $20,000$km mean is set which will test a variety of orbits. The eccentricity sweeps a range of 0.1 to 0.6 uniformly to keep varying the planet size in the image. Inclination, Right Ascension of the Ascending Node (RAAN) and true anomaly are also dispersed strongly. The filter noise stays within values that are known to work, this ensures that covariance variations are not due to the filter, but rather to the changing OpNav measurements. The camera field of view barely varies by $\pm 0.001^\circ$ in these runs. All the parameters for the 100 Monte-Carlo runs executed are summarized in Table 4.7. Although the parameters provide a wider set of variability than the Monte-Carlo number can validate, it provides an illustration to the capabilities and some insight in the overall performance of the FSW stack. Both the *NIH-SVD* and *Hough Circles* both run in roughly $45s$ per simulation.

Figure 4.17 shows the state error percentage for the Monte-Carlo analysis. Variations are seen in the filter performance (notably along the velocity lines) mostly during the transient convergent phase. Nonetheless all of the solutions come to a sub-percent error on position and below two percent errors on velocity. This does show the variability induced by the orbit, yet also shows that the stability of the problem remains intact for a broad set of images.

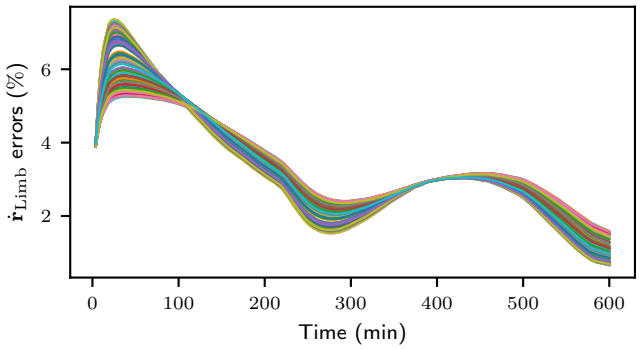
Another interesting study is the covariance analysis that can come following these Monte-

Table 4.7: Orbital Monte-Carlo Dispersions

Parameter	Dispersion
Semi-Major Axis (km)	$\mathcal{N}[20,000, 3,000]$
Eccentricity	$\mathcal{U}[0.1, 0.6]$
Inclination (°)	$\mathcal{U}[-60, 60]$
True Anomaly (°)	$\mathcal{U}[0, 359]$
RAAN (°)	$\mathcal{U}[-60, 60]$
Argument of Periapses (°)	190
Filter Noise	$\mathcal{U}[5, 7]$
Camera FOV	$\mathcal{U}[39.999, 40.001]$



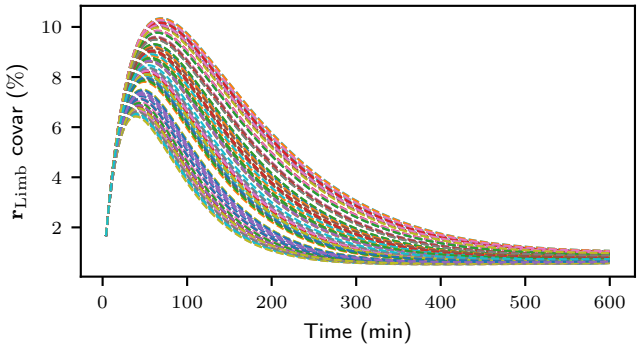
(a) Inertial Position Error Norm



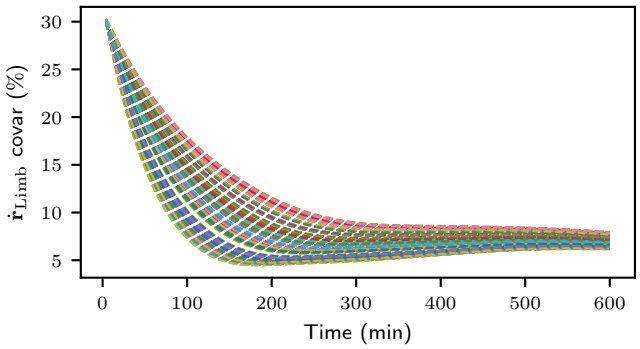
(b) Inertial Velocity Error Norm

Figure 4.17: Monte-Carlo Varying orbits— *NIH-SVD*

Carlos. Figure 4.18 shows that the covariance stays coherent with regard to the state errors above it, and show variations in the solution as well. A mission trade-study could provide requirements on either state error or covariance bounds and showing variability in these values according to the



(a) Inertial Position Covariance



(b) Inertial Velocity Error Norm

Figure 4.18: Monte-Carlo Orbit Covariances— *NIH-SVD*

orbit provides deep insight into the full system.

4.6.2 Dispersions on Filter Noise

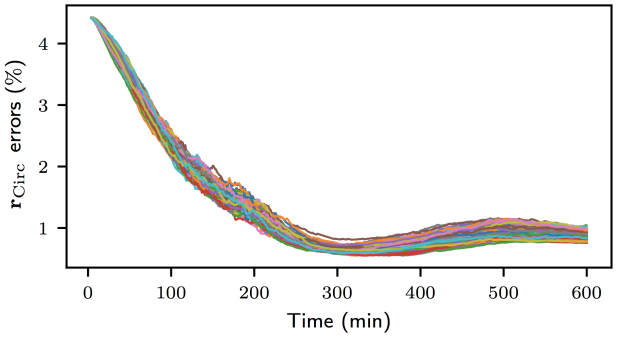
A second parameter that tends to be tuned ad hoc would be the noise scale factor on the filter measurements. Due to the difficult nature of the uncertainty extraction from the image processing, measurement noises tend to be over-estimated.[27] A thorough analysis can provide insight on the effect of such parameters on the state estimate of the filter. Table 4.8 summarizes the dispersions for this case. The orbital variations have been reduced, while the filter now spans a greater number of scale factors. Once more, 100 simulations are run in order to analyze the sensitivity.

The results in Figure 4.19 displays the results for variations in filter noise. These show that

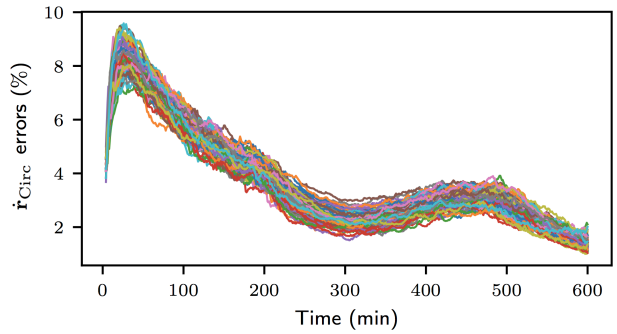
Table 4.8: Filter Noise Monte-Carlo Dispersions

Parameter	Dispersion
Semi-Major Axis (km)	$\mathcal{N}[22,000, 3,000]$
Eccentricity	$\mathcal{U}[0.2, 0.4]$
Inclination (°)	$\mathcal{U}[-20, 20]$
True Anomaly (°)	$\mathcal{U}[0, 180]$
RAAN (°)	25
Argument of Periapses (°)	190
Filter Noise	$\mathcal{U}[5, 9]$
Camera FOV	$\mathcal{U}[39.9, 40.1]$

the tuning of this parameter doesn't change the results to an important extent. This justifies the choice of parameters and provides a range of applications for the orbit.



(a) Inertial Position Error Norm



(b) Inertial Velocity Error Norm

Figure 4.19: Monte-Carlo Varying Filter Noise— *Hough Cirlces*

4.6.3 Dispersions on Camera Parameters

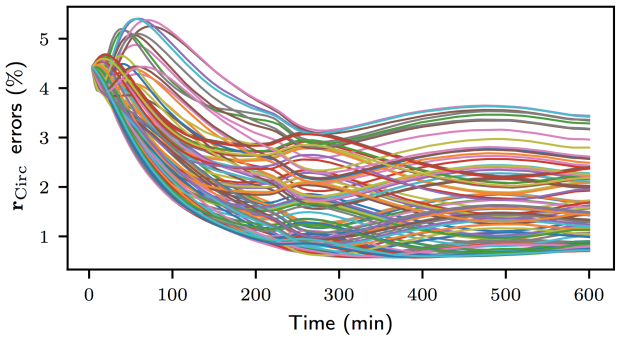
The last sensitivity analysis studies the camera field of view. Although many other parameters can be analyzed, in the interest of seeing specific effects of single parameters, only field of view variations are added. This is also analogous to changing the focal length as the sensor size stays unchanged. The goal of this analysis is to test how sensitive our choice of parameters is to the end goal of navigating the spacecraft. In this regard, a relatively high orbit is chosen in order to stay relatively insensitive to the size of the planet and eclipses. The eccentricity is kept below 0.4, but over 0.2 to keep varying the planet size in the image. Similarly the inclination varies around the zero value, whereas the anomaly stays in the first half of the orbit.

The filter noise stays within values that are known to work, which primarily inflates the covariance and therefore changes the state estimate. The camera field of view however varies by $\pm 1^\circ$ in this run. All the parameters for the 100 Monte-Carlo runs executed are summarized in Table 4.9.

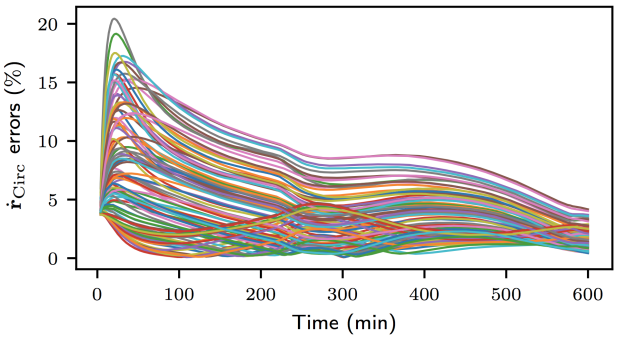
Table 4.9: Camera Monte-Carlo Dispersions

Parameter	Dispersion
Semi-Major Axis (km)	$\mathcal{N}[22,000, 3,000]$
Eccentricity	$\mathcal{U}[0.2, 0.4]$
Inclination (°)	$\mathcal{U}[-20, 20]$
True Anomaly (°)	$\mathcal{U}[0, 180]$
RAAN (°)	25
Argument of Periapses (°)	190
Filter Noise	$\mathcal{U}[4, 6]$
Camera FOV	$\mathcal{U}[39, 41]$

Figure 4.20 displays the results for the Monte-Carlo analysis. Although the variations can be large, it's because they compound with the ones shown previously, and the results remain positive regarding filter convergence. This type of analysis can be key in order to understand the pixel accuracy and the results that the OpNav suite can provide. The speed of execution and the ease with which variability can be added to the simulation provides large mission support capabilities.



(a) Inertial Position Error Norm



(b) Inertial Velocity Error Norm

Figure 4.20: Monte-Carlo Varying Camera Errors— *Hough Circles*

4.7 Conclusions

This chapter presents novel developments on several fronts. It provides estimation results using solely autonomous OpNav on orbit about Mars, in order to quantify the accuracy achieved with different methods. It also introduces a *Hough Circle* based navigation method. Although this method doesn't outperform current state-of-the-art algorithms, it provides a robust alternative with a simple implementation. The difference between state-of-the-art algorithms and the new *Hough Circles* method lies in the choice of fitting ellipses or circles: yet a simple circle fit with a spacecraft aiming to center the planet on the camera plane has shown good results. Finally the entire study is done with an underlying pointing coupling with the orbit determination. Doing simultaneous attitude and OD adds a level of fidelity to the simulation presented. Furthermore, it supports the development of a high-image count OpNav framework. By taking pictures frequently

(every minute) even a less accurate method like *Hough Circles* show valid results for autonomous navigation around Mars.

The Monte-Carlo analysis for sensitivity testing, and design choices has shown how useful these tools can be in order to evaluate methods in use. Though not applied to a direct mission — and therefore in the abscence a set of specific requirements to verify — this analysis aims to present which parameters are most likely to change the OD solution, as well as confirm the robustness of the algorithms to a wide set of situations. Indeed, as any image processing algorithm and control law is tuned, the question of hyper-parameter choices are raised. This study allows to validate these choices and in other cases could provide specific insight on what needs to be changed to achieve a mission target performance. Some lessons learned are given in this section: the camera parameters strongly influence the orbit solution, while the orbit variability and filter parameters are not as influential to the estimate.

Chapter 5

Robust Optical Navigation for Spacecraft Autonomy

The speed and modularity of the simulation presented in the first chapter allows for in-depth fault studies. With the ability to add faults to the camera, image processing algorithms can be tested in harsh, off-nominal conditions. Furthermore, the ability to run multiple image processing algorithms side by side allows for on-line comparisons between methods.

This chapter presents a robust navigation framework by opposing optical navigation algorithms which extract different features from images. This is seen notably by using a limb detection method versus a circle finding method. Although the edge quality will affect both, the circle finding algorithm will be naturally more robust as it will not detect any other structures in the image. The work allows the testing novel image processing methods in flight. With a comparative framework in place, image processing through neural networks or other novel methods can be run in the background and tested directly with more reliable methods.

5.1 Overview

The overarching task in this chapter is to enhance the developments of previous results with faults and robustness. The principle scenario remains an elliptical orbit around Mars, but with camera errors and faults. Once more, no ground-based measurements (range or range-rate data) will be used for navigation. The elliptical orbit will facilitate the testing of the image processing algorithms with a variety of apparent diameters, and the determination and control interaction with both slow and fast orbital dynamics.

This work presents fully coupled pointing-OD OpNav on orbit, as well as comparative capabilities for different methods in a flight-like simulated environment. Realistic simulations help define what level of complexity is needed on board for a specific requirement to be satisfied. The framework defined and developed in previous chapters provides the ability to run several FSW algorithms in parallel, which is useful to develop technology tests in flight where a trusted method oversees the performance of the other. It also provides a framework for fault detection analysis in which several methods run simultaneously in order to harness each other's strengths.

In order to prepare and validate an autonomous OpNav algorithm, analysis of outliers and their potential impact on the state estimate must be well understood. This is notably used to assess the feasibility of missions utilizing autonomous or ground-based OpNav in order to simulate hardware and software limitations. When testing and validating autonomous systems, testing for the "off-nominal" case is crucial to understanding the limits of the performances and mission feasibility. These situations can also arise from sensitivity analysis through Monte-Carlos allows the comparison of expected algorithm performance against mission requirements.

5.2 Image Corruptions for Increased Realism

In order to push the envelope in autonomous algorithms, a realistic spacecraft environment must be generated. This includes the ability to toggle and manipulate the sensors and simulation, and the ability to inject faults into the system. In OpNav, the modeling of the environment and faults happen primarily through the camera model.

Modeling cameras and their corruptions is a longstanding task in fields that require extracting data from images. In fields like solar physics, the removal of both known physical phenomenon and camera artifacts to extract information from instruments,[93] while robotics has greatly pushed the field of computer vision with stereo-vision[197] modeling for legged robots[120] or open-source camera calibration software such as *Calibu*[112] developed at University of Colorado Boulder. Some recent navigation work even involves using camera distortions in order to improve navigation:[154] by determining the attitude of the spacecraft by exploiting the offsets in the Earth observational

imagery.

This section focuses on the types of corruptions that are predominant in the space sector. Some basic modeling tools are implemented in order to generate noisy and imperfect data. The goal of this chapter is not to push the state-of-the-art in camera modeling and calibration, but rather to provide tools to test the robustness of algorithms in place. Artifacts that target a specific algorithm's weakness are key to developing the necessary techniques to switch between methods and ensures a continuous and robust navigation solution.

5.2.1 Camera Modeling for OpNav

Space imaging is traditionally done using a Charged Coupled Device (CCD) sensor to capture images. CCD technology has been able to provide high-level performance for detection and remains the first-choice technology for high-end applications. On the other hand, Complimentary Metal Oxide Semiconductor (CMOS) chips have intrinsic advantages — low power consumption, readout rate, noise, radiation hardness, integration capability — that make them well suited for space applications.[139] Reference 139 provides an in-depth comparison of the two sensors. The work presented in this thesis does not make an explicit choice on which sensor is preferable or implemented. Each provides advantages and disadvantages, and is susceptible to different types of faults.

Different space environments require different corruption models. An asteroid may spew dust and occult a camera as a function of particle weights, electrostatic and solar radiation pressure forces;[185] high energy particles in magnetic fields can also induce frequent Single Event Upsets (SEU);[32] while cosmic rays[174] are high energy particles hit the image plane and saturate a streak of pixels. These can happen at various frequencies depending on the shutter speed and the local environment.

Some common camera artifacts or errors that need to be modeled in simulations (or removed in real data) are: dark current, hot and dark pixels, colored noise, spectral variability, radial distortions, and blur. Lenses or camera imperfections during manufacturing process can also lead

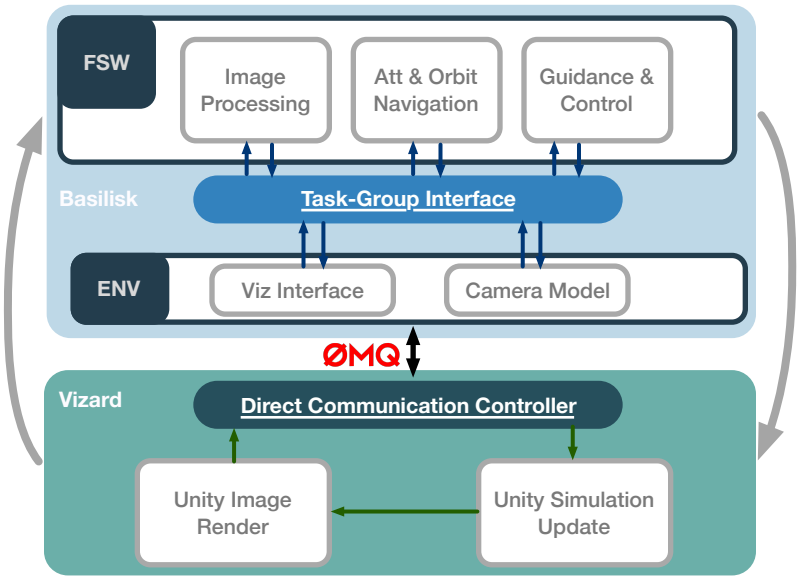


Figure 5.1: Information flow with Camera Model

to aberrations and static artifacts that can be modeled such as small scratches or dents on the lens. The color characteristics of a camera sensor also vary widely depending on the instrument. If a navigation algorithm requires contrast between features, the relative sensitivity to the color spectrum is important to model.

This subsection focuses on some attributes that can be added to the simulated camera for realism. The schematic in Chapter 2 for the information flow is reminded in Figure 5.1. This shows a camera model which can harness the *OpenCV* library in order to alter the images. In this work, dark current, stuck an dark pixels, noise, blur, and cosmic rays are implemented. Every image processing algorithm will present a weakness. For instance, a limb finding algorithm either has to ensure there are no artifacts or sharp gradients in the image outside of the limb in order to not misplace points. The goal in this thesis is not to model cameras to the highest physical accuracy, but rather to provide the tools in order to trip imaging processing algorithms.

Cameras developed for navigation are usually high-resolution, low noise, and well characterized such as the AMICA instrument on Hayabusa,[70] HRSC on ESA's Mars Express,[69] or the Orion EM-1 flight camera.[99] Previous simulation work has combined also hardware with synthetic

images in the loop in order to test EDL capabilities.[67] This section will provide a more conservative model of the camera being used, and environmental perturbations in order to display fault detection capabilities.

5.2.2 *Basilisk* Modeling Capabilities

In order to provide the necessary capabilities for fault detection, several perturbations were added to the camera module. Gaussian noise generates color noise in the image which adds errors to the features on the planet map. Corrupting details on known textures provides an interesting tool in order to challenge feature detection and tracking. This is done by using the `addWeighted` *OpenCV* method and scaling the noise according to the input parameter. The noise is zero-mean with standard deviation equal to twice the gaussian noise parameter. The image is then thresholded at the 6-σ value of the noise parameter in order to keep the background dark. In summary:

- Gaussian noise is added with a mean of zero, and standard deviation of $2 \times$ G, where G is the input parameter to the *Basilisk* module.
- The image thresholds down all pixels below the 6-σ values of the noise added

A similar effect is achieved with the implemented dark current model. Dark current is due to thermal properties of the CCD or CMOS sensor in use: as electrons are created independently of incoming light, they are captured in the pixel potential wells and appear to be a signal. Ways to mitigate thermal electron creation is in fact to cool actively or passively[68] the sensor. Dark current noise is the statistical variation of this phenomenon which can be modeled[75] and methods exist for it's extraction from images.[85] In *Basilisk*, dark current noise is added with a Gaussian noise model with zero standard deviation and mean of $15 \times$ D, where D is another input to the module.

A blur is added to the image as well. Softening edges is crucial to any kind of Limb based navigation as it allows to create a comparable limb to a specific camera. The size of the blur can be toggled from the module and Figure 5.3 shows how it can be used to replicate a specific camera quality. The blur is programmed in *OpenCV* by convolving the image with the filter in

Equation (5.1) in the case of a blur parameter set to 3.

$$[Bl] = \frac{1}{3 \times 3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5.1}$$

Equation (5.1) only requires that the parameter be an odd number in order to have a center pixel in the $[Bl]$ matrix. The size of $[Bl]$ is the only variable for blurring in the *Basilisk* module.

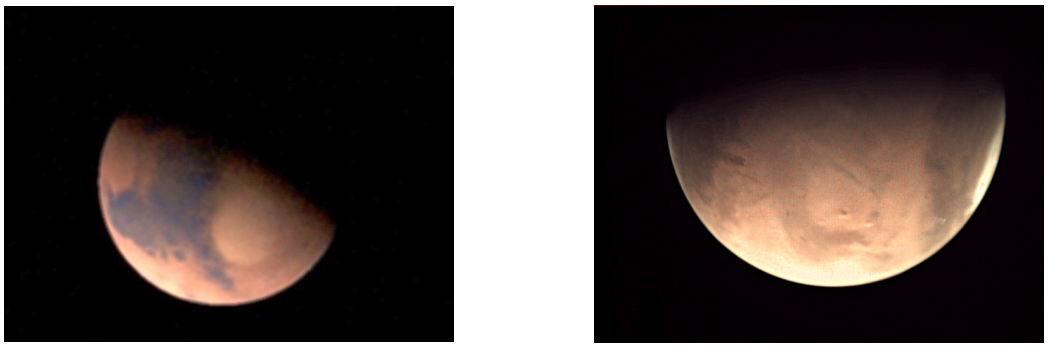
Hot and dead pixels are also implemented as a static perturbation. At the initialization of the run, a random number generates arbitrary pixel coordinates and either saturates them dark or light. The corrupted pixels stay the same throughout the simulation and provide an example of a static artifact. They are chosen by uniform distrubtions of the size of the images and values are either maximized or minimized.

Finally, cosmic rays are also implemented. Cosmic rays are ionized nuclei: 90% being protons, 9% alpha particles, and the rest are heavier nuclei. They hit the Earth's atmosphere approximately 1000 times per square meter per second.[75] Although their origins and energies are commonly studied in astrophysics, their effect on camera sensors is also of interest. Indeed by hitting the camera sensor with relatively high energies, they can saturate a line of pixels and corrupt the image.[216] Cosmic rays are modeled in *Basilisk* by randomly choosing a point on the sensor as well as second point within a maximal distance of the first one. The abundance of cosmic rays on an image depend on the shutter speed amongst other parameters, and the module allows to toggle the frequency and quantity of such events. The outline of the model is described as follows and only depends on a single variable CR:

- Cosmic rays are added with a probability of $p(\mathrm{CR}) = 1 - \frac{1}{(\mathrm{CR}^2 + 0.02)}$, where CR is the input parameter. The threshold is defined such that CR= 1 provides roughly a $\frac{1}{10}$ chance of getting a ray, while CR= 10 will generate roughly 10 rays per image.

- The length of each ray is bounded by a 50×50 pixel box, and can be modified.

- Each call to the method adds one ray, and the model attempts to add CR cosmic rays (one is only added with probability $p(\text{CR})$).



(a) *Basilisk*-*Vizard* Image (b) Mars Express VMC Image

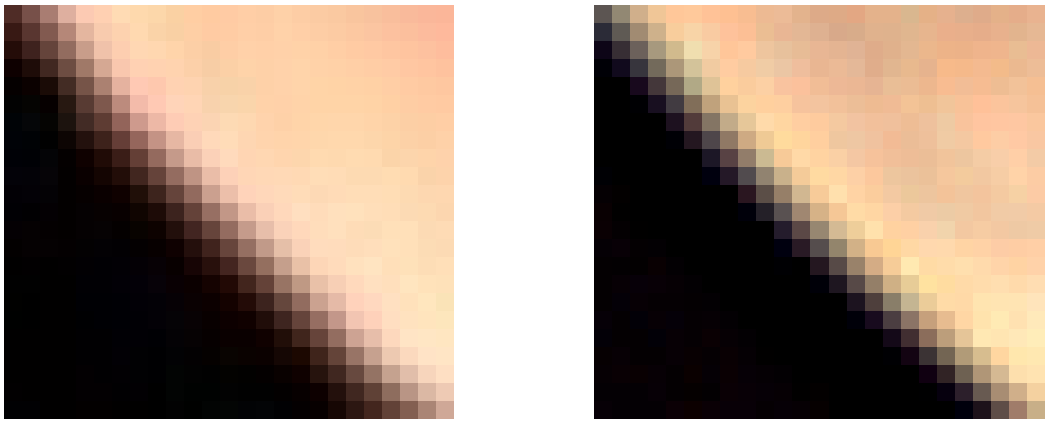
Figure 5.2: A Comparison of Mars Express VMC Image Versus a Synthetic Image

The Visual Monitoring Camera (VMC) is mounted on Mars Express, ESA's deep-space probe orbiting Mars[1]. It provided low-resolution images of the *Beagle* lander after separation, and has been converted to a imager for the rest of the mission duration. It is neither a scientific nor navigation instrument, but provides some examples of coarse planet images. Although it is clear that VMC does not provide that image quality expected from a navigation camera, it does stand as an example of low-resolution images for OpNav. Indeed, if an image processing algorithm can be robust to such images, it not only opens the door to OpNav for smaller less-expensive spacecraft (namely cubeSats and small-sats), but also provides a robust alternative or check for other state-of-the-art methods.

Figure 5.2 displays some comparative images from the VMC camera and synthetic images generated by *Vizard* and modified by *Basilisk*. With just a few of these corruptions, generated images take on far more realistic appearances: these images are not intended to provide identical results, but rather to show comparable features. Looking more closely, Figure 5.3 shows how the limbs can be set to present similar gradients going from the planet to the surrounding darkness. Other corruptions such as radial distortion are also potential candidates to add as errors, though

[1] http://blogs.esa.int/vmc/

they do not provide errors specific to these methods and are more applicable to EDL and terrain navigation.



(a) *Basilisk*-*Vizard* Limb (b) Mars Express VMC Limb

Figure 5.3: Limb Comparisons for VMC Image Versus a Synthetic Image

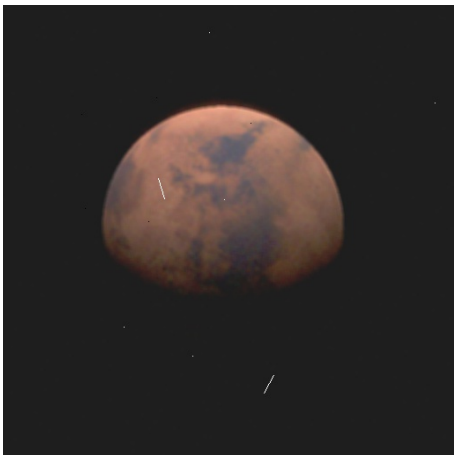
These images show the potential that small error cumulation has to make images more difficult to process. In *Basilisk* are modeled stochastically in order to provide sufficient simulation speed. Indeed, the caveat that the derived corruptions present is that they may not be physically representative of the phenomena that generate them. Better models for such errors is in progress and requires developments on both the visualization and simulation. The primary goal of this work is nonetheless to navigate under uncertain measurements, and from that perspective the implemented models provide sufficient accuracy.

5.2.3 Camera Corruption Modes

In the following simulations, a subset of corrupted modes are used. This is done to focus the attention on certain facets of the navigation problem. Indeed, with just the few camera errors implemented, a wide range of different cameras and faults can be simulated. The rest of the chapter focuses on how to navigate in the presence of the uncertainty generated by the camera model.

The simulation can generate multiple errors, unexpected objects or artifacts, and other corruptions in the images. Figure 5.4 shows an array of different images that can be generated.

(a) Low-noise Image



(b) Med-noise Image

(c) High-noise Image

Figure 5.4: Image with Added Corruptions

Subfigure 5.8a shows an image with only a few Hot-Dead pixels, while Figure 5.8b shows a blur value of 2, a cosmic ray value of 2 and a gaussian noise value of 2. Finally the last image in Figure 5.4c shows the simulation with Gaussian noise set to 6, dark current set to 5, hot-dead pixels set to 5, Cosmic rays set to 5 and blur set to 6. The last image is very difficult to use for navigation as the planet doesn't stick out of the background, all the while presenting many false edges to account for. Images with this amount of noise are not good candidates for spacecraft navigation, but rather display the capabilities of the simulation. Once again, color images are produced though the additional information is not used.

In order to challenge image processing methods and remain realistic, three main levels of corruption are added. The first two revolve around the addition of Cosmic Rays (from now on noted CRs). As seen previously these are modeled as random streaks across the sensor. They provide are expected to strongly corrupt the *NIH-SVD* performance. Therefore it is studied in two main modes which only add CRs as error sources.

Other artifacts can be added to the image. Two other cameras are added called "Noise" and "NoiseHigh" in which other parameters are turned on to further hurt the image quality. The modes used are summarized in Table 5.1 and are reference in the rest of this section.

Table 5.1: Camera Corruption Modes

Mode Name	**Gaussian**	**Dark Current**	**Hot-Dead**	**CRs**	**Blur**
Nominal	0	0	0	0	0
CR1	0	0	0	1	0
CR2	0	0	0	2	0
Noise	3	0	0.5	1	3
NoiseHigh	5	1	1	2	5

NoiseHigh is solely used to show the boundaries of the algorithm performances. The primary errors are found and use in the Noise and CR cases as seen in the following sections.

5.3 Navigation Results with Corrupted Images

In order to provide consistent results, the same simulation as Chapter 4 and can be found in Tables 4.4, 4.5, 4.6. The orbital parameters and initial conditions are reminded in Table 5.2, while the ideal camera parameters are reminded in Table 5.3. Alongside the noise parameters in Table 5.1, all the information defining the simulations is provided.

Table 5.2: Spacecraft Initial States

$\boldsymbol{\sigma}_{\mathcal{BN}}$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$
$\boldsymbol{\omega}_{\mathcal{BN}}$[rad/s]	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$
Orbital Elements	(18000km, 0.6, 10°
$(a, e, i, \Omega, \omega, f)$	25°, 190°, 80°)

Table 5.3: Camera Parameters

$\boldsymbol{\sigma}_{\mathcal{CB}}$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$
${}^{\mathcal{B}}\boldsymbol{r}_{\mathcal{CB}}$[m]	$\begin{bmatrix}0 & 0.2 & 0.2\end{bmatrix}^T$
Resolution (pixels)	$\begin{bmatrix}512 & 512\end{bmatrix}^T$
Sensor Size (mm)	$\begin{bmatrix}10 & 10\end{bmatrix}^T$
Field of View (°)	$\begin{bmatrix}40 & 40\end{bmatrix}^T$

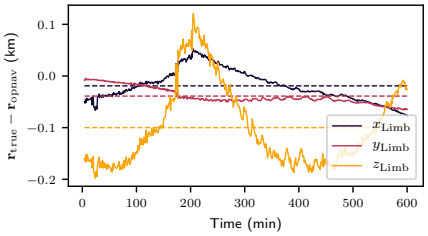
The *NIH-SVD* method relies on finding a unambiguous limb. Models which aim to understand the way limbs are casted upon camera sensors such as the Kasseleinen-Shkuratov model[191] and the Hapke model[88] have been utilized in the literature.[43] Fast subpixel edge detection methods such as Sobel-Zernike[222] moments or Zernike moments[80] have also been developed to better fit the detected limb. In the interest of introducing faults for one of the two methods, the limb finding algorithm used remains the *Canny* transform. This will allow some of the perturbations discussed in the previous section to provide faults. The *Canny* transform is still a reliable method and provides good results in a clean image as seen in the previous chapter.

In this section both methods are run with identical errors and environments. The results show how each method operates at the baseline when dealing with more difficult camera models. The study of the stand-alone methods allows to instruct the fault detection algorithms that follow by identifying the specific behaviors of each method.

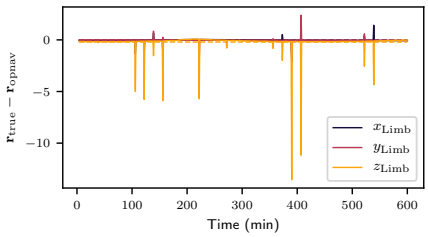
5.3.1 Adding Cosmic Rays

The results show the performance of the CR1 and CR2 modes. This is shown by the features found in the images, the measurement quality (as compared to the truth), and the filter results. The CR1 mode provides roughly a cosmic ray ever ten images, while CR2 generates one to two rays every two images.

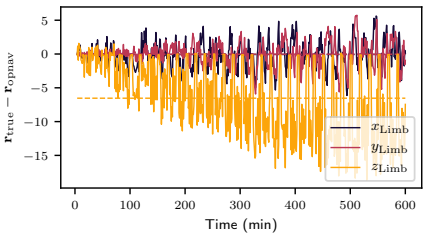
Figure 5.5 shows the measurement quality degrading for the *NIH-SVD* method. Without any guard against false-limb identification, poor results are not surprising, but show the sensitivity



(a) *NIH-SVD* Nominal



(b) *NIH-SVD* CR1-mode



(c) *NIH-SVD* CR2-mode

Figure 5.5: Cosmic Ray Effects on the *NIH-SVD* Measurement

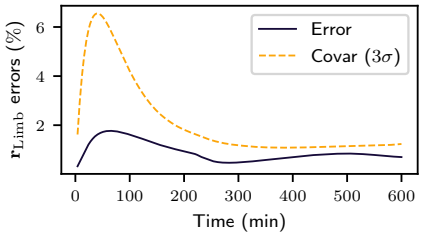
of the least-squares estimate to false points in the $[H]$ matrix:

$$[H] = \begin{bmatrix} \bar{\boldsymbol{s}}_0'^T \\ \vdots \\ \bar{\boldsymbol{s}}_N'^T \end{bmatrix} \tag{5.2}$$

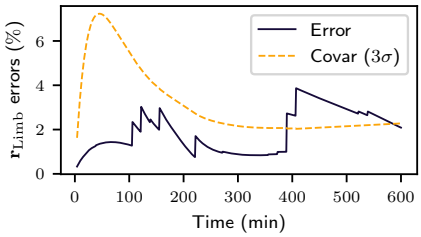
$$[H]\boldsymbol{n} = \mathbf{1}_{N\times 1} \tag{5.3}$$

Equation (5.2) reminds the construction of the $[H]$ matrix which needs to be inverted to solve for $\boldsymbol{n}$. Indeed, in the CR1-mode measurement errors can spike up to an order of magnitude above nominal errors, seen by comparing Figures 5.5a and 5.5b. Figure 5.5 also displays the statistics designed in the previous section: the spike count in the measurements are close to expected numbers though slightly below. Finding fewer CRs in the data is due to the fact that some of them appear on the planet disk and are not seen by the FSW algorithms. Figure 5.5c displays how harmful frequent artifacts can be: with frequent cosmic rays hitting the sensor, the measurement quality degrades greatly.

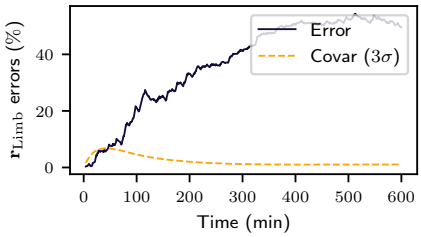
The impacts on the OpNav solution are notably seen in subfigures 5.5b and 5.5c of Figure 5.6



(a) *NIH-SVD* nominal



(b) *NIH-SVD* in CR1-mode



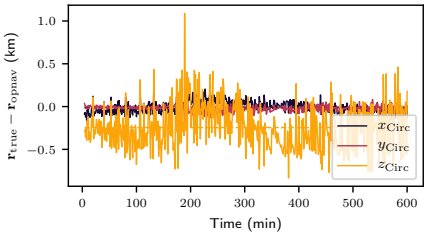
(c) *NIH-SVD* in CR2-mode

Figure 5.6: Cosmic Ray Effects on the *NIH-SVD* Filters

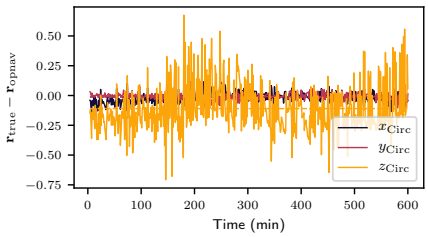
where the filter is frequently offset by poor measurements which are difficult to identify at the time of processing. It should be noted that the filter performs well for CR1 mode, aside from the erratic state estimates. Figure 5.6c echoes what was seen in Figure 5.5c: the filter cannot converge with erroneous measurements at such a high frequency. CR2 mode therefore displays a limit to the current implementation of the limb finding algorithm.

Figure 5.7 shows the measurements achieved by the *Hough Circle* implementation. Once more, it can be seen that in the Nominal case, these measurements are more noisy than ones produced by *NIH-SVD*. Nevertheless, as the CR value increases, little to no impact is seen on the measurements. This is because the circle finding method expects to find circles and not gradients directly. Lines are therefore poorly represented in the parameter space and do not affect estimates strongly.

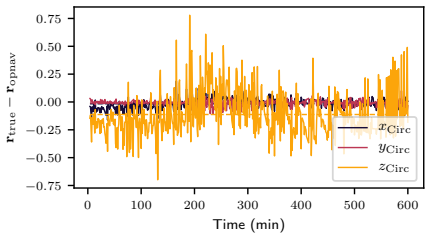
The robust performance of the *Hough* method is confirmed in Figure 5.8 which displays the filter results. Although there are slight changes in the fluctuations during the first hundred minutes, and a slight error increase at the end of the simulation, there is almost no difference between the



(a) *Hough* nominal

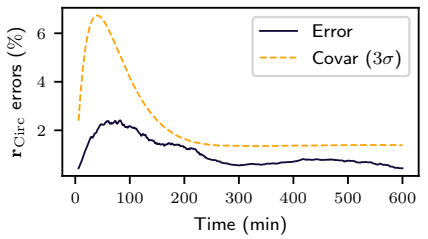


(b) *Hough* in CR1-mode

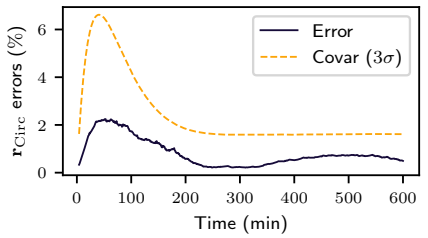


(c) *Hough* in CR2-mode

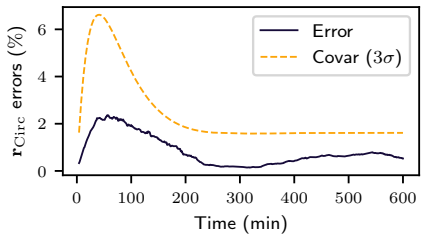
Figure 5.7: Cosmic Ray Effects on the *Hough* Measurement



(a) *Hough* nominal



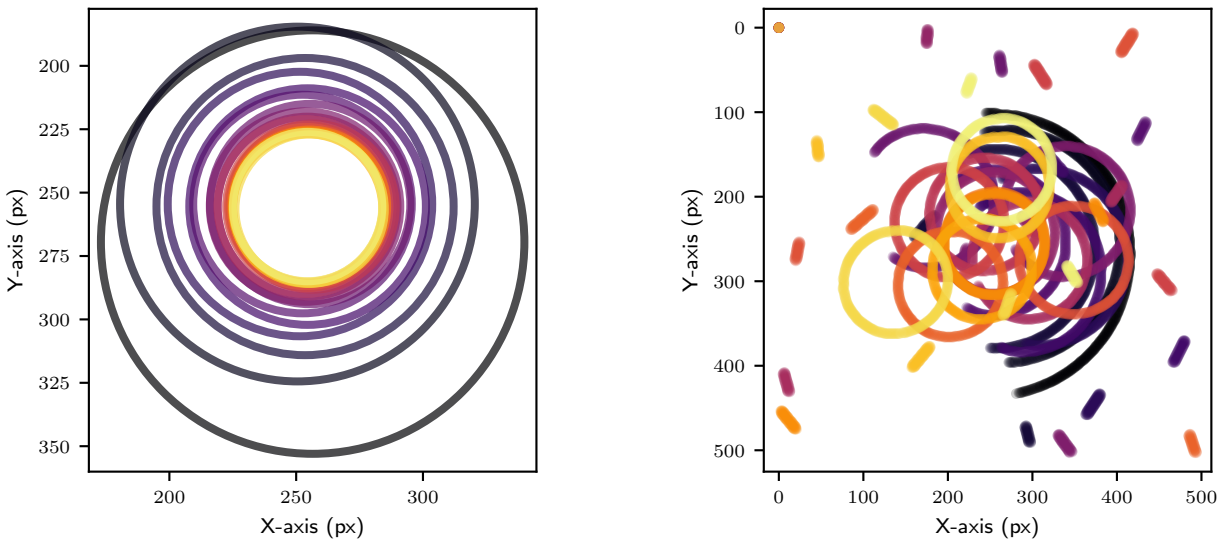
(b) *Hough* in CR1-mode



(c) *Hough* in CR2-mode

Figure 5.8: Cosmic Ray Effects on the *Hough* Filters

performance with and without cosmic rays present.



(a) *Hough Circles* with perturbations

(b) Limbs and *NIH-SVD*

Figure 5.9: Image Processing in CR2-mode

Finally, the different performances can also be illustrated by observing the output of the image processing modules. Indeed, Figure 5.9 shows on the left the fact that the circles found are centered and consistent with the nominal case. On the other hand, the edges found with the *Canny* transform are erroneous, which also leads to the pointing algorithm's performance to degrade. Furthermore the planet can be seen off center because of less accurate estimates which in turn affect the pointing.

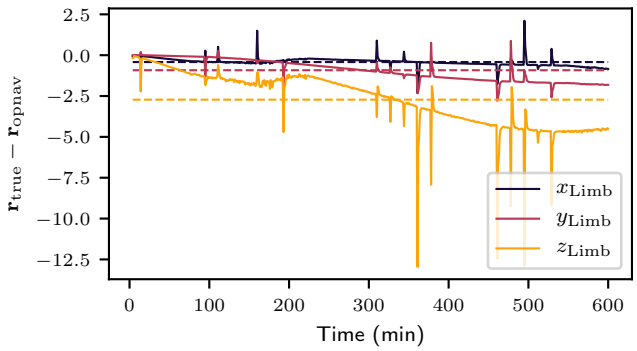
The results shown in this subsection display how the CR modes test the limits of the Limb-finding method, while displaying the robustness of *Hough Circles*. These simulations alone provide a basis for fault detection and mitigation by using two different image processing methods.

5.3.2 Compounding Image Errors

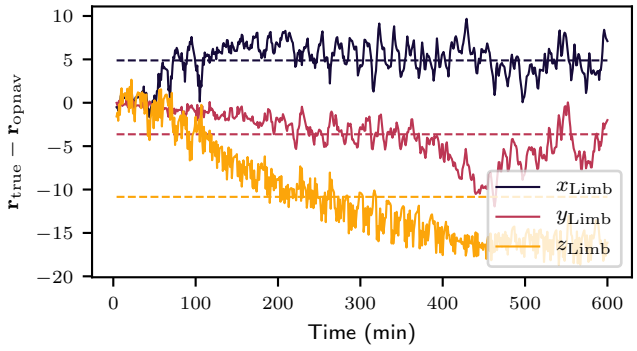
This subsection illustrates the performances of the OpNav algorithms used on the Noise and NoiseHigh modes. Although the CR modes generate a sufficiently rich environment for fault detection, the two Noise modes add complexity and realism to the scenarios. This section also serves

to show how robust a basic implementation of *Hough Circles* can be. Indeed, with no change to the values of the image processing implementation in Table 4.5, the circle finding is still performant despite being subject to wide variety of images.

Compounding small image errors interestingly shows that both methods are robust to smaller changes, but when adding considerable noise as in cases studied in this section, only *Hough Circles* provide good estimates. These results justify the assumption made in the visualization design constraints: the limb is the primary feature of importance. It does show that the cumulative error sources can lead to large errors with both methods, including loss of tracking in the case of the Limb-based method.



(a) *NIH-SVD* measurements in Noise-mode

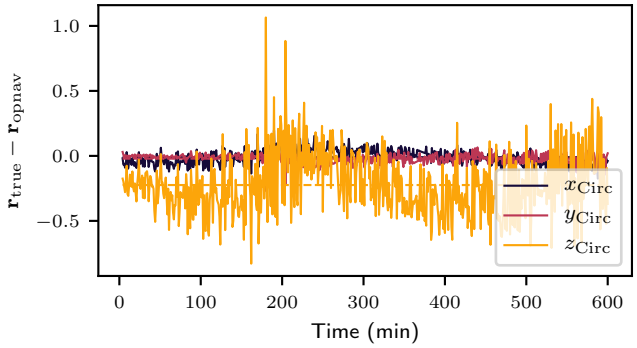


(b) *NIH-SVD* measurements in NoiseHigh-mode

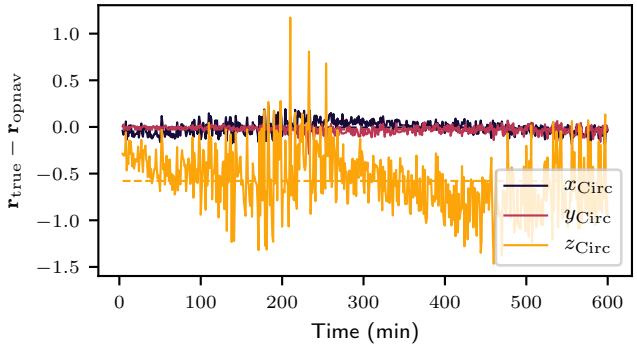
Figure 5.10: Noise and NoiseHigh Measurements (km) - *NIH-SVD*

The Limb-based method suffers once again from consistent measurement errors seen in Fig-

ure 5.10. Notably, Figure 5.10b displays the measurements in the NoiseHigh mode which not only have large static noise but also a walking bias.



(a) *Hough* measurements in Noise-mode

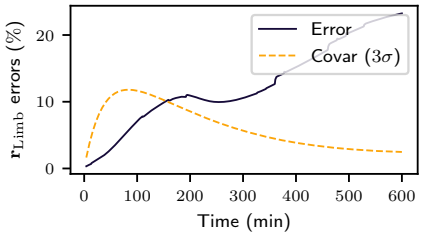


(b) *Hough* measurements in NoiseHigh-mode

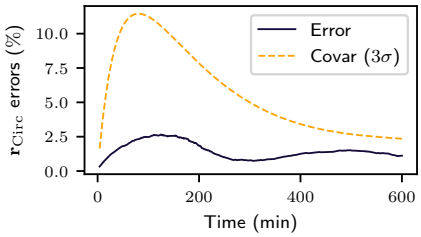
Figure 5.11: Noise and NoiseHigh Measurements (km) - *Hough Circles*

The *Hough* measurements are once again fairly consistent. Figure 5.11 shows that the quality of the measurements decrease (nearly multiplied by three in the NoiseHigh case) and display similar biases than in the Limb finding. Nevertheless, the impact of such drastic artifacts is not seen as clearly.

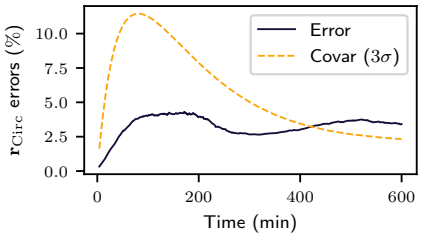
The measurement results are corroborated by filter results in Figure 5.12. The top figure shows that the Limb-based method once again diverges. On the other hand, subfigures 5.12b and 5.12c show better performance of the circle-based method. Although the NoiseHigh mode pushes the position estimate outside of the covariance bounds, error percentages are still below 5%.



(a) *NIH-SVD* in Noise-mode



(b) *Hough* in Noise-mode



(c) *Hough* in NoiseHigh-mode

Figure 5.12: Position Errors for Noise and NoiseHigh Cases

This section has shown the available perturbations that are added to images in a set of fault modes. The *Hough Cirlces* are a good candidate for robust fault detection and mitigation as results degrade much more slowly than in the *NIH-SVD* with the *Canny* transform.

5.4 Outlier Detection and Mitigation

The previous section has shown the strength of a direct implementation of *Hough Circles*. As stated previously, although several additions can be added to the Limb detection algorithm in order to increase the limb finding fidelity, these require image treatment and layer algorithms and checks which all need to be validated and tested. Generally speaking, this narrows the use cases for such methods: every subcomponent of the artifact removal and limb finding needs to be tested and tuned to a specific application. Furthermore, the overall complexity makes it more difficult to implement such methods on small-sats and nano-sats.

The research framework developed throughout this thesis permits the addition of faulty measurements to the simulation and identification of poorly processed measurements. In a real

mission scenario, corrupted measurements could be weighted less favorably than trust-worthy ones when identified. The difference in the two image processing methods is pictured in plots such as in Figure 5.9. The Limb-finding method has no prior on the fact that a spheroid must be found, whereas the circle-fitting does not allow for broader shapes.

The limb processing module has room for improvement in order to protect against faulty limbs. If the slightly weaker performance of *Hough Circles* has been noted in Chapter 4, it at least provides robustness. This framework not only generates comparative results between two identical methods, but also runs the two methods side-by-side in one simulation. This section attempts to heighten the ability of the simulation to highlight the differences between the two primarily methods.

5.4.1 Developing a Fault Detection Algorithm

The benefit of the *Basilisk*-*Vizard* tool beyond that of comparison, is to harness the strengths of each implementation in order to reject faults. In this case, a comparative module for image fault detection can reject the measurements from the limb finding method when not in accordance with the circle-finding algorithm. *Hough Circles*, in their simplicity provide assurance of a certain type of feature being processed, but remain noisy measurements. Limbs on the other hand provide much more refined measurements, but can be more susceptible to artifacts and lighting conditions.

In order to defend against the artifacts in the images — as seen in Figure 5.9 — the algorithm displayed in Alg. 2 is developed. Fault detection and diagnosis can take many forms as a function of the system.[62,106] In this scenario, the fault detection primarily acts as a monitoring agent for a Multi Input Single Output (MISO) system. A previously used technique for fault detection can be merging data from several measurements,[214] which ensures that there is no mismatch between different sensors or processing methods. This doesn't preclude the use of methods that use multiple different types of residuals processors to extract the probability of a fault,[215] many of these methods have been implemented on *NASA* programs such as *SPLICE*[28] or *DIMES*.[35]

The goal in this scenario is to detect when one of the two image processing methods fails and

Algorithm 3 OpNav Fault Detection

```
faultMode ← input(FaultMode)
prim ← read(PrimaryNav)
sec ← read(SecondaryNav)
if prim !valid and sec !valid then
    write(None)
else if prim valid and sec !valid then
    if faultMode < 2 then
        write(prim)
    else if faultMode == 2 then
        write(None)
else if prim !valid and sec valid then
    if faultMode > 0 then
        write(None)
    else if faultMode == 0 then
        write(sec)
else if prim valid and sec valid then
    if dissimilar(prim, sec) == False and faultMode == 0 then
        merged ← Merge(prim, sec)
        write(merged)
    else if dissimilar(prim, sec) == False and faultMode > 0 then
        write(prim)
    else if dissimilar(prim, sec) == True and faultMode == 0 then
        write(sec)
    else if dissimilar(prim, sec) == True and faultMode > 0 then
        write(prim)
```

is therefore applied prior to being ingested by the filter. Analyzing raw measurements keeps the detection algorithm agnostic to any dynamic models, and focuses solely on the image processing reliability. The difficulty of identifying faults post-filtering relies in the post-fit residual analysis. If the measurement noise is not well calibrated or indicative of the true measurement probability density function, finding outliers will be difficult. Furthermore, outliers in post-fit residuals might be indicative of a change in the dynamics, physical event, or highly uncommon measurement. Therefore performing outlier detection and mitigation requires use of many measurements to provide good statistics as well as a rigorous framework to interpret them.[215] The benefit being that the detection is statistically sound, and provides both dynamics insight as well as knowledge of the previous best estimate.

The algorithm implemented provides a simple, easily enhanced fault detection method. It involves a similarity check between the two solutions given by the value of:

$$\text{sign}\left(\mathcal{L}_2(\boldsymbol{r}_{BN,2} - \boldsymbol{r}_{BN,1}) - \sigma_{\text{fault}}\sqrt{\mathcal{L}_2([R_1] + [R_2])}\right) \tag{5.4}$$

Equation (5.4) is what is coded in the `dissimilar` method used in Algorithm 3.

Because the image processing errors are highly dependent on the specific algorithms implemented, overarching methodology and logic where chosen over method-specific enhancements. Furthermore, allowing the filter to process a measurement that is faulty can effect the state estimate and therefore compromise the overall filter performance. By not adding the knowledge of the dynamics, this fault detection focuses on the quality of the processed image alone. The possible issue with such a comparison only resides in the merging good measurements: the formulas used assume independent measurements which is not provably verified. Nevertheless, this is a improvement that only applies to one mode and isn't related to the detection of the fault itself. Alg. 3 is used in three possible fault detection setups. It inputs two measurements extracted from the images and outputs a single vector and noise matrix for the filter.

- The first mode (*faultMode*=0) is a merging method in which both measurements are trusted equally. If either is present without the other, it is passed on to the filter, and if they are both present they are merged:

$$[R] = \left([R_1]^{-1} + [R_2]^{-1}\right)^{-1} \tag{5.5}$$

$$\boldsymbol{r}_{BN} = [R]\left([R_1]^{-1}\boldsymbol{r}_{BN,1} + [R_2]^{-1}\boldsymbol{r}_{BN,2}\right) \tag{5.6}$$

  Where $[R_1]$ and $[R_2]$ are the noises associated to measurements $\boldsymbol{r}_{BN,1}$ and $\boldsymbol{r}_{BN,2}$ respectively, and $\boldsymbol{r}_{BN}$ and $[R]$ are the merged measurement and noise matrix.[12]

- The second mode (*faultMode*=1) inputs a primary method and secondary method. Since the secondary is not considered as accurate as the primary, they are not merged, but a dissimilar check is implemented. This is done by differencing the two vectors, and checking

that the norm for the difference is less than the sum of the $\mathcal{L}_2$ (or Frobenius) norms of the noise matrices scaled by a factor σ_{fault} to control sensitivity seen in Equation (5.4).

- Finally, the last mode (*faultMode*=2) only uses the primary method as a measurement, but only does so when both measurements are present and similar. Although restrictive given the measurement input, this does ensure that no erroneous methods can be given to the filter.

5.4.2 Results in Specific Simulation Case

In order to get a clear idea of the performance of the fault detection, a specific case is shown in the CR1-mode. The error mode signifies that the only source of corruption comes from cosmic rays as defined in previous sections and Table 5.1. Furthermore, this mode has been shown — in Figures 5.5 and 5.6 — to greatly affect the state estimate of the *NIH-SVD* method.

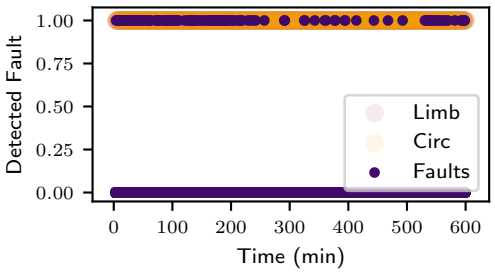
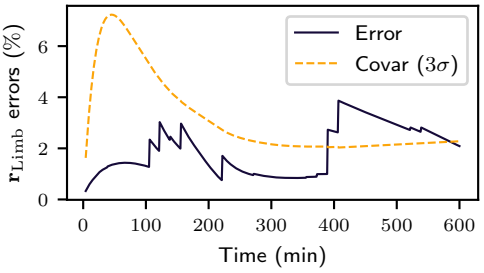


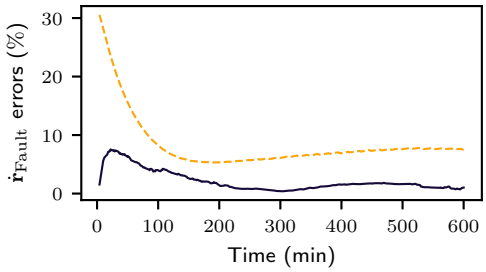
Figure 5.13: Faults Detected during Scenario

Figure 5.13 shows the faults detected and current valid measurements (zero indicates no fault or invalid measurements, while one indicates valid or faultless updates). This is run with *faultMode*= 0, $\sigma_{\text{fault}} = 0.5$, and a filter noise noiseSF = 5. The plot shows that faults are frequently found (due to a relatively small σ_{fault} value). In this fault-mode measurements are merged together and so faults do not signify necessarily that a method has failed entirely, rather that it is too distant relative to the other one to be confidently merged.

Figure 5.14 shows the last mode is implemented with $\sigma_{\text{fault}} = 0.5$. It's clear that while the



(a) Without any mitigation



(b) With mitigation in fault mode 0

Figure 5.14: Position Errors with and without Fault Detection

first plot shows errors that grow greatly when the artifacts are detected, the second one discards those and waits for a proper measurement validated by both methods. The combination of both processing methods in the fault detection algorithm greatly increases the robustness of the OpNav stack to errors like cosmic rays.

The algorithm derived in Alg. 3 shows a solution to a specific problem that can occur during autonomous OpNav. Detecting and protecting from probable faults on orbit is a key enabler to routine navigation without the ground in the loop. Depending on the mission design, the specific camera, and environment, other faults would need to be accounted for as well. The flexibility of this framework and the ease with which faults and mitigation algorithms can be added provides enabling capabilities to the field of spacecraft autonomy.

5.4.3 Results for a Diverse Set of Scenarios

As stated in the previous subsection, one of the goals of the fault detection algorithm lies in the generality it can provide. Therefore it becomes important to test a set of different camera and FSW parameters in order to ensure the performance generalizes. This also provides a sensitivity analysis to the results given image errors.

In order to run such an analysis, a range of different cases were designed and tested. The cases still focus on the scenario where cosmic rays are the primary error source, but also extend to the Noise-mode described earlier in the chapter. The important parameters that can be toggled by an GNC engineer are primarily going to be :

- Which measurement source is primary, and which is secondary
- Which fault detection mode is in use
- What is the sensitivity to fault detection, ie the value fo σ_{Fault} that defines the necessary overlap of measurement uncertainties
- What is the filter measurement noise scaling

The first three points directly affect the fault detection implementation. Depending on the methods used and their sensitivities, a primary and secondary measurement type is likely to be self-evident. The choice of primary and secondary navigation sources also ties in with the fault detection type in use. The sensitivity is less intuitive to set as it scales the $\mathcal{L}_2$ norm of the noise on the measurements. The validity of the estimated uncertainty is done by the image processing algorithms (which are here assumed to be faulty at times) and can be implemented in a variety of ways. Finally, the noise scaling on the filters, which also ties in to the measurement uncertainties, changes the state estimate. It must also be varied to ensure that the filter is performing as expected.

Table 5.4 provides the list of all the different cases tested. The last six cases are of interest in order to test the most difficult cases in a decoupled environment: if the pointing also depends

Table 5.4: Fault Cases

Case Name	Primary	Secondary	Camera	σ_{Fault}	Filter Noise	Auto-Point
Test-Hough	*Hough*	None	Clean	N/A	5	On
Test-*NIH-SVD*	*NIH-SVD*	None	Clean	N/A	5	On
Test-Merge	*NIH-SVD*	*Hough*	Clean	3	5	On
CR1-Case 1	*NIH-SVD*	*Hough*	Cos-Ray 1	$\frac{1}{3}$	5	On
CR1-Case 2	*NIH-SVD*	*Hough*	Cos-Ray 1	$\frac{1}{3}$	6	On
CR1-Case 3	*NIH-SVD*	*Hough*	Cos-Ray 1	$\frac{1}{3}$	7	On
CR1-Case 4	*NIH-SVD*	*Hough*	Cos-Ray 1	1	5	On
CR1-Case 5	*NIH-SVD*	*Hough*	Cos-Ray 1	1	6	On
CR1-Case 6	*NIH-SVD*	*Hough*	Cos-Ray 1	1	7	On
CR1-Case 7	*NIH-SVD*	*Hough*	Cos-Ray 1	3	5	On
CR1-Case 8	*NIH-SVD*	*Hough*	Cos-Ray 1	3	6	On
CR1-Case 9	*NIH-SVD*	*Hough*	Cos-Ray 1	3	7	On
CR2-Case 1	*NIH-SVD*	*Hough*	Cos-Ray 2	$\frac{1}{3}$	5	On
CR2-Case 2	*NIH-SVD*	*Hough*	Cos-Ray 2	$\frac{1}{3}$	7	On
CR2-Case 3	*NIH-SVD*	*Hough*	Cos-Ray 2	1	5	On
CR2-Case 4	*NIH-SVD*	*Hough*	Cos-Ray 2	1	7	On
CR2-Case 5	*NIH-SVD*	*Hough*	Cos-Ray 2	3	5	On
CR2-Case 6	*NIH-SVD*	*Hough*	Cos-Ray 2	3	7	On
Noise-Case 1	*Hough*	*NIH-SVD*	Noise On	3	6	On
Noise-Case 2	*Hough*	*NIH-SVD*	Noise On	3	7	On
Noise-Case 3	*Hough*	*NIH-SVD*	Noise On	3	8	On
Noise-Case 4	*Hough*	*NIH-SVD*	Noise On	4	6	On
Noise-Case 5	*Hough*	*NIH-SVD*	Noise On	4	7	On
Noise-Case 6	*Hough*	*NIH-SVD*	Noise On	4	8	On
OD-Case 1	*Hough*	*NIH-SVD*	Noise On	3	6	Off
OD-Case 2	*Hough*	*NIH-SVD*	Noise On	3	7	Off
OD-Case 3	*Hough*	*NIH-SVD*	Noise On	3	8	Off
OD-Case 4	*Hough*	*NIH-SVD*	Noise On	4	6	Off
OD-Case 5	*Hough*	*NIH-SVD*	Noise On	4	7	Off
OD-Case 6	*Hough*	*NIH-SVD*	Noise On	4	8	Off

on the image processing as done in all other cases, the spacecraft can lose track of the target and not perform orbit determination any longer.

The results of each case is summarized in Table 5.5. In order to condense the results, only

Table 5.5: Fault Case Root Mean Square Errors

Case Name	Position (%)	Position Covar (%)	Velocity (%)	Velocity Covar (%)
Test-Hough	1.003	2.833	2.605	9.884
Test-*NIH-SVD*	0.779	3.554	1.941	11.29
Test-Merge	0.943	2.286	2.597	8.242
CR1-Case 1	1.111	2.436	3.048	8.611
CR1-Case 2	1.215	3.718	3.023	10.91
CR1-Case 3	1.217	5.381	2.877	13.80
CR1-Case 4	0.994	2.299	2.720	8.282
CR1-Case 5	1.013	3.610	2.504	10.65
CR1-Case 6	1.125	5.308	2.647	13.63
CR1-Case 7	0.967	2.290	2.678	8.263
CR1-Case 8	0.996	3.595	2.429	10.62
CR1-Case 9	1.065	5.279	2.479	13.57
CR2-Case 1	1.138	2.694	3.003	9.559
CR2-Case 2	1.183	5.557	2.787	14.27
CR2-Case 3	1.133	2.647	2.931	9.458
CR2-Case 4	1.122	5.533	2.643	14.21
CR2-Case 5	1.299	2.586	3.348	9.317
CR2-Case 6	1.275	5.493	2.992	14.14
Noise-Case 1	9.129	3.940	26.92	11.54
Noise-Case 2	3.890	5.597	9.789	14.43
Noise-Case 3	6.799	7.684	18.29	18.18
Noise-Case 4	2.487	3.993	5.718	11.75
Noise-Case 5	2.388	5.606	5.750	14.43
Noise-Case 6	2.671	7.705	6.220	18.20
OD-Case 1	2.463	3.996	5.888	11.76
OD-Case 2	2.683	5.604	6.266	14.44
OD-Case 3	2.346	7.703	5.328	18.18
OD-Case 4	2.793	3.993	6.694	11.75
OD-Case 5	2.615	5.604	6.191	14.43
OD-Case 6	2.432	7.695	5.696	18.15

Root-Mean-Square (RMS) errors of the position and velocity estimates are noted:

$$\mathrm{RMS}(\boldsymbol{r}_{\mathrm{meas}}) = \sqrt{\sum_{i}^{N} 100 \frac{|\boldsymbol{r}_{\mathrm{meas}}(t_i) - \boldsymbol{r}_{\mathrm{truth}}(t_i)|}{|\boldsymbol{r}_{\mathrm{truth}}(t_i)|}} \tag{5.7}$$

$$\mathrm{RMS}([P]) = \sqrt{\sum_{i}^{N} 100 \frac{\sqrt{[P]_{0,0}(t_i) + [P]_{1,1}(t_i) + [P]_{2,2}(t_i)}}{|\boldsymbol{r}_{\mathrm{truth}}(t_i)|}} \tag{5.8}$$

where $\boldsymbol{r}$ is a studied vector value with a measured and true value (indicated by subscripts) and $[P]$ is the covariance matrix with subscripts indexing into it (starting with index zero). Equation (5.7)

assumes there are N time steps in the simulation, and takes into account estimates at every time of the simulation.

Table 5.5 also colors the results on two scales: a position scale (green at 0.7% and red at 10%) and a velocity scale (green at 1.9% and red at 27%). The results show that every CR case run has errors below 1.3% on position and several cases dropping below 1%. Velocity estimates are commensurate though consistently higher as seen in Chapter 4. These cases all have the *NIH-SVD* as a primary method and borrow from the circle-finding method when errors are triggered. Choosing the primary revolves around some prior knowledge of the instrument and it's sensitivities. The algorithm virtually lifts all faulty estimation errors from the scenario. The Noise case can prove to be more troublesome for *NIH-SVD* (Figure 5.10a), but when switching the *Hough* estimates to the primary navigation method and relaxing the fault criteria, good performance is achieved.

A few important comments can be made about Table 5.5. The first is that the covariance values are very comparable are similar but generally not comparable one-to-one. This is mostly because of the covariance scaling that is added to enhance the filter performance. Furthermore, Noise-Case 1 and Noise-Case 3 show poorer performance than neighboring cases. In these high noise cases, this does show a sensitivity to this noise scaling.

5.5 Conclusion

This section covers a wide range of faults that can be added to the camera module, tested the performance of different OpNav measurements, and proposed a fault detection algorithm for FSW. The section shows the robustness of the *Hough*-based image processing stack. Without changing any parameters, a large variety of images can be processed successfully. The strength of this analysis is still in it's generality: different faults and algorithms can be integrated and tested in the simulation readily.

The power of the *Hough Circle* transform lies in the ubiquitous and strongly general results it provides. It should be noted that weakness do occur when very high background noise is present. This is due to the circles in the image having cumulative votes which become drowned out by the

many other points. This can also be re-adjusted and would signify a very large instrument change which would certainly impact other methods as well.

The fundamental limitation of the *Hough Circle* algorithm, is that it is constrained to finding circles. Therefore navigating oblate objects will bring intrinsic errors. This can be overcome by using the *Hough* transform for ellipse finding, which leads to increase computational complexity since the parameter space goes from three-dimensional to six-dimensional in general.

This chapter provides new insight on the robustness of guidance methods for spacecraft autonomy. After more than a decade, AutoNav is still at the cutting edge of high-technology-readiness-level (TRL) spacecraft autonomy. The proposed research harnesses the derived simulation capabilities and presents new developments and analysis.

Chapter 6

Machine Learning for Autonomy and Optical Navigation

Previous developments in Chapter 5 have borrowed from a well developed literature and number of pre-existing techniques. The analysis of fault cases is a key component to enhancing autonomous OpNav and spacecraft autonomy in general. Similar methods have been used successfully in space exploration and autonomy generally, and are therefore prime candidates for continuous developed.

Nevertheless, recent years have seen the exponential development of Machine Learning (ML) for robotic and everyday applications. This section explores how ML can be applied to spacecraft autonomy in two different regards. The first study relates to high-level or decision-making autonomy and rely on Reinforcement Learning (RL) while the second subsection will focus on CNNs for image processing for OpNav.

6.1 Overview

This final technical contribution develops novel capabilities that the simulation explores. Providing capabilities to integrate certain optimization methods that borrow from ML methods opens new avenues in the field of aerospace. As ML continues to grow as a field, many different derivatives are developed and applied to different problems. In this chapter, two techniques are studied for spacecraft autonomy. The first is a high-level autonomy: learning spacecraft decision making for a set of tasks. This is done by using Reinforcement Learning on a set of examples: a proof of concept for an orbit insertion, and a pointing scenario using OpNav. The second is in

neural network development, and consists of using the simulation to train a Convolutional Neural Network (CNN) to find planet center and radius for OpNav.

With the previous developments in simulation capabilities, OpNav algorithms, and multi-method fault detection and navigation, ML can be be integrated in the greater context of spacecraft GNC. More notably, multi-method fault detection algorithms pave the way towards the use of novel Machine Learning algorithms in flight by simulating exactly the FSW environment that will be reproduced in flight. The implemented FSW can harness two methods: a trusted classical one, and a demonstration algorithm and change fault modes in order to dictate which method navigates.

Both the sections of this chapter are vast fields of research, and the results shown are preliminary. Ongoing work in the Autonomous Vehicle Systems lab at CU Boulder intends to continue pushing the state-of-the-art. Yet introductory results in both the RL application to high-level autonomy, and the CNN application to OpNav image processing are achieved and presented.

6.2 Introduction to Reinforcement Learning for Decision-Making

At present, examples of high-level spacecraft autonomy typically fall into two categories: rule-based autonomy and optimization-based autonomy. Rule-based autonomy treats a spacecraft as a state machine consisting of a set of mode behaviors and defined transitions between modes. Pioneered by missions like Deep Impact,[122] and currently used by missions such as the PlanetLabs constellation,[72] spacecraft using rule-based autonomy transition between operational and health-keeping modes (charging, momentum-exchange device desaturation) autonomously without ground contact. They require accurate understanding of the mission environment in order to prepare for unwanted behavior. Additionally, rule-based approaches do not readily support the integration of multiple competing mission objectives, and require that those trades be made on the ground with humans in the loop before mission sequences are uploaded.

This section explores an approach to fit spacecraft autonomy into the Partially-Observable Markov Decision Process (POMDP) framework and its general solution through model-free "Deep-Q" RL. In this section, an example is shown for proof-of-concept in which a spacecraft autonomously

times an orbit insertion maneuver (stationkeeping was also developed). This framework therefore provides a better solution to the autonomous orbit problem, all the while opening the door to novel fault detection methods and autonomous maneuvers. The work presented can extends to the OpNav problem and the integration of results displayed in this subsection into *Basilisk* will provide the ability to run previous OpNav simulations in a RL framework.

In contrast to rule-based autonomy, optimization-based autonomy typically requires large amounts of computing power that precludes their use on-board. This method also requires realistic models, and well developed testing environments. Examples of this work include the Applied Physics Laboratory's SciBox software library (used to generate *MESSENGER* mode sequences) and the *ASPEN* mission planning suite developed by the Jet Propulsion Laboratory and applied to the Earth Observing-1 mission.[39] The simulation developed in Chapter 2 paired with the developments in camera models of this chapter provide both the simulation efficiency and realism required to develop RL for spacecraft decision-making. This section aims to explore the applications and frameworks necessary to apply deep reinforcement learning to the autonomous navigation.

A small collection of other works in the application of machine learning techniques to spacecraft problems exists in the recent literature, mostly focusing on the application of learning approaches to control problems in uncertain environments. Several works such as References 91 and 51 have considered reinforcement learning in the context of autonomous aerobraking planners, with mixed results. Others explore machine learning techniques for asteroid proximity operations[79] or autonomous lunar landing.[179] Importantly, these approaches have focused on low-level control with reinforcement learning, an area that has been traditionally been covered by conventional estimation and control techniques with great success. In contrast, this work explicitly examines applications of reinforcement learning to high-level spacecraft planning and decision-making problems that have traditionally been the domain of rigid expert policies or optimization-focused strategies. Although not directly applied to the OpNav simulations of previous chapters, this work provides the framework for future RL developments to serve those mission concepts.

6.2.1 General Problem Statement

6.2.1.1 Spacecraft as a State Machine

POMDPs compactly represent the processes facing a software agent acting in an evolving environment according to some higher-level objective.[30] The mathematics of such processes, and challenges associated with them, are reviewed briefly here.

As in traditional Markov Decision Processes, the state in a POMDP is updated by a transition function $\mathcal{F}$, and at any given time can be computed as a function of the previous state and the most recent action taken by the considered agent:

$$s_k = \mathcal{F}(s_{k-1}, a_{k-1}) \tag{6.1}$$

This state s_k (e.g. spacecraft position, phase angle, filter covariance) is observed by the agent according to some observation function $\mathcal{H}$:

$$o_k = \mathcal{H}(s_k) \tag{6.2}$$

Given an observation o_k of the state, the agent then selects an action a_k to influence the future state according to some policy π:

$$a_k = \pi(o_k) \tag{6.3}$$

While these transition functions represent physical or software-defined process dynamics, the objective of an agent is ultimately motivated by a reward function $\mathcal{R}$:

$$r_k = \mathcal{R}(s_{k-1}, a_{k-1}, s_k) \tag{6.4}$$

The objective of a software agent within a POMDP is to select a policy π that maximizes its realized reward.

The "true" non-linear dynamics resulting from gravity interactions are taken to follow the two-body equations of motion in the presence of perturbing accelerations:

$$\ddot{\boldsymbol{r}} = \frac{-\mu}{r^3}\boldsymbol{r} + \boldsymbol{a}_p \tag{6.5}$$

At the same time, a pre-defined reference trajectory obeying two-body dynamics without perturbing accelerations is used to define the desired mission:

$$\ddot{\boldsymbol{r}}^* = \frac{-\mu}{r^{*3}}\boldsymbol{r}^* \tag{6.6}$$

The ignorant propagator in Equation (6.6) is also used to propagate forward the spacecraft's current orbital state estimate, $\hat{\boldsymbol{x}}$. The true state is noted $\boldsymbol{x}$ while the reference state is $\boldsymbol{x}^*$. The resulting state and estimate errors are defined as

$$\boldsymbol{e}_s = \boldsymbol{x} - \boldsymbol{x}^*,\ \boldsymbol{e}_{\text{est}} = \boldsymbol{x} - \hat{\boldsymbol{x}} \tag{6.7}$$

Likewise, the spacecraft-internal control error is defined as:

$$\boldsymbol{e}_c = \hat{\boldsymbol{x}} - \boldsymbol{x}^* \tag{6.8}$$

These models, alongside a family of deep reinforcement learning agents, are implemented in Python using the *OpenAI* `gym` framework[23] to represent the spacecraft-mode POMDP interface in a standardized manner. The deep learning agents are created using *Keras* [1] using *Tensorflow* [2] back-end.

6.2.1.2 Reinforcement Learning Methods

This subsection briefly describes the methods used for optimizing the RL agents. The two methods used in this chapter are "Deep-Q" and "PPO2".[187] In both of these methods, the agent begins with no knowledge of the environment. By taking actions and observing the state and rewards, an approximation of the "Q" function can be developed:

$$Q_\pi(s,a) = \mathbb{E}\left[R_{\text{total}}\right] = \mathbb{E}\left[\sum_t \gamma^t r_t\right] \tag{6.9}$$

where R_{total} represents the sum of all future rewards given the state and action, and $\gamma \in [0,1]$ is the discount-rate. The concept behind Q-learning lies in approximating the Q function in order to

[1] `https://keras.io`

[2] https://www.tensorflow.org/

find the optimal policy:

$$\pi^*(s) = \max_a Q(s,a) \tag{6.10}$$

The Bellman equation[130] then relates Q to consecutive time steps:

$$Q_{\pi^*}(s,a) = r + \gamma \max_{a'} Q(s',a') \tag{6.11}$$

Where prim values of actions and states relate to the next iteration. The loss function is the extracted by seeking to minimize the mean squared error of the prediction term

$$L = \frac{1}{2}\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right) \tag{6.12}$$

The "deep" component to neural networks reside in the fact that a neural network is used to approximate the Q function. Gradient decent or other optimization methods can then minimize the loss function. This then allows the policy to be backed out of the knowledge of the Q function by maximizing the reward through action. PPO2 or other policy gradients seek to directly optimize the policy π.

One form of the gradient estimator is given in Reference 187 is

$$\hat{\boldsymbol{g}} = \mathbb{E}_k \left[\nabla_\theta \log \pi_\theta(a_k|s_k)\boldsymbol{A}_k\right] \tag{6.13}$$

Where $\boldsymbol{A}_k$ is the advantage estimate: it represents the difference between R_t (discounted rewards) and the baseline estimate for the rewards.[187] In other words, the advantage estimate characterizes if a action provided better results than currently expected. The algorithm therefore learns the advantage function as well as the policy in two separate neural networks. Trusted region policy optimization, clipped surrogate objectives are other additions the enhance the performance of PPO2.

6.2.2 Proof of Concept-Mars Orbit Insertion

6.2.2.1 Scenario Description

The first environment simulates insertion into orbit about a planet, as is represented visually by Figure 6.1. This is an example of an interplanetary mission in which a spacecraft is flying towards

Mars, and needs to conduct an impulsive maneuver at the correct time to enter orbit about Mars under uncertain knowledge of its state. This type of decision is crucial to the success of many interplanetary missions, and therefore represents an important challenge for proposed autonomy systems. This is also an example of a timely maneuver in which any unexpected behavior could be lethal to the mission as ground teams could not react in time. All simulations are run with an unmodeled acceleration of Mars' second gravitational spherical harmonic J_2 for true propagation.

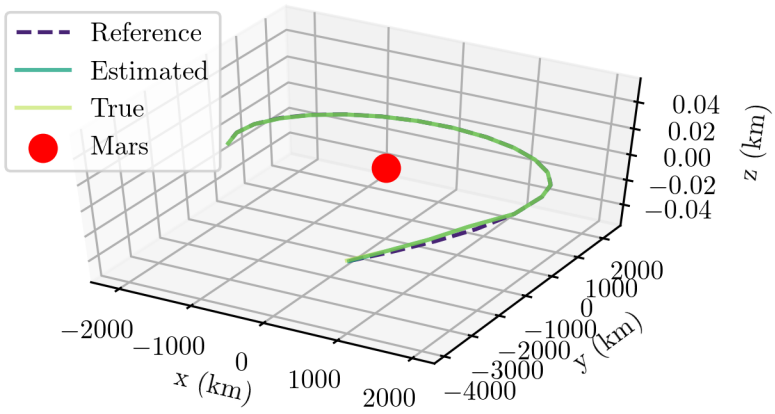


Figure 6.1: Trajectories during Insertion Maneuver

Three modes are available for the spacecraft to chose from, these modes define the action space. The first mode performs orbit estimation, which is general but could be replaced with the OpNav modes used in previous chapters of this thesis. Entering this mode reduces the uncertainty on the state, and brings the estimated position and velocities to the true position and velocities exponentially. The second mode is a control mode: it moves the true states of the spacecraft towards the estimated states. For MOI, the spacecraft considers an additional mode, "thrust," which applies an impulsive ΔV computed with it's estimated states at a specific time and reflects a major maneuvers to adjust its trajectory. The challenge of the orbit insertion scenario is therefore to ensure that this thrust is applied at the correct time to ensure orbit insertion, while at the same time maintaining accurate position estimates and controlling towards a defined reference trajectory. Furthermore, the knowledge of the true spacecraft states are paramount to the spacecraft thrusting

correctly.

Figure 6.1 pictures the reference trajectory. The initial orbit is a hyperbolic fly-by with a semi-major axis of $a = -2,000$km, eccentricity of $e = 2$, and initial true anomaly of $\nu = -1.5$ rad. The goal orbit has a semi-major axis of $a = 2,000$km, eccentricity of $e = 0.01$, inclination of 0rad, true anomaly of $\nu = 0.01$ rad. This scenario therefore brings a spacecraft from a hyperbolic fly-by orbit into a captured circular orbit around it's target planet (Mars). Each mode has a length of 2 minutes with a time step of 5 seconds.

Table 6.1: Initial Conditions Dispersons for Orbit Insertion

Parameter	Dispersion
Semi-major axis of true trajectory	$\mathcal{N}$(-2000, 100)
Eccentricity of true trajectory	$\mathcal{N}$(2, 0.01)
True anomaly of true trajectory	$\mathcal{N}$(-1.5, 0.01)
Semi-major axis of estimated trajectory	$\mathcal{N}$(-2000, 100)
Eccentricity of estimated trajectory	$\mathcal{N}$(2, 0.01)
True anomaly of estimated trajectory	$\mathcal{N}$(-1.5, 0.01)

Table 6.1 shows the dispersions on the initial conditions that are used for training. This allows to train on a wide set of errors orbit insertion introduces new challenges to overcome in order to robustly train agents for autonomous decision-making. These challenges revolve around the binary success criteria of the thrust maneuver. The ability to thrust is an action of equal weight to estimation or control modes. This encourages the agent in training to use it promptly, and very frequently fall into a local minimum that consists in thrusting immediately and attempting to minimize further errors with the control mode.

Table 6.2: Hyperparameters used in Training the Final MOI Iteration.

Parameter	Value/Type
Number of Hidden Layers	1
Hidden Layer Depth	64
Hidden Layer Activation	ReLU
Output Layer Activation	Linear
Learning Rate	0.01
Number of Training Episodes	10,000
Annealing Segment Length	3,000

Table 6.3: Weights in the Orbit Insertion Reward Function

Weight	Value
w_e	-0.1
w_c	-0.05
w_t	$\frac{100}{1+\lvert t_{\text{thrust}}-t_{\text{ref}}\rvert}$

These challenges are overcome by implementing an eligibility trace and by forcing the thrust maneuver to occur in a 5-step window, centered around the expected thrust time (8th step given the initial conditions). This gives the RL agent the ability to optimize the thrust time, but doesn't require it to learn known astrodynamics facts: that the thrust-time is optimal at periapse of the hyperbolic orbit. More details on the complete implementation can be found in Reference 92. The reward function in this scenario consists of a term for the state error (estimated state with respect to the reference), another term is added for control cost, and a positive reward is added for thrust timing

$$r(\boldsymbol{e}, c, \Delta\text{V}) = \underbrace{w_e \boldsymbol{e}_c^T \boldsymbol{e}_c}_{\text{state error}} + \overbrace{w_c c}^{\text{control cost}} + \underbrace{w_t \Delta\text{V}}_{\text{thrust reward}} \tag{6.14}$$

where $\boldsymbol{e}$ is the state error, c is the control authority used and ΔV is the norm of the thrust impulse. Scalars w_e, w_c, and w_t are the weights corresponding to the state error, the control cost, and the thrust respectively. The values of these weights are a key component to the success of the agents learning. If the weight on the state error is large relative to the other costs, a poorly timed thrust could make that penalty abruptly skyrocket. However, if it is too small (or not accounted for) the spacecraft will not continue controlling orbit position after the thrust. Furthermore, it is difficult for the agent to learn to be on the correct trajectory in order to thrust at the right time and position.

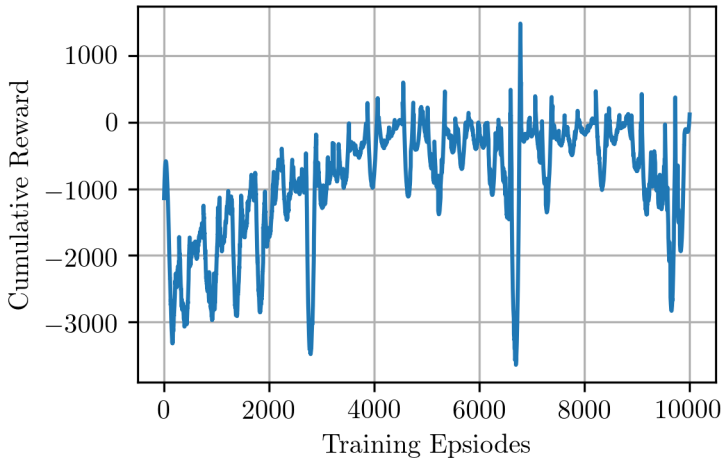
Weights that allow for successful training are listed in Table 6.3, where t_{thrust} is the time at which the thrust action is taken by the agent, while t_{ref} is the time at which the reference changes orbits. This form for w_t allows for the aforementioned smoother, more continuous decay in the reward for thrusting off target. The scale of 100 is for the reward to be commensurate with the state errors penalties.

6.2.2.2 Mars Orbit Insertion Results

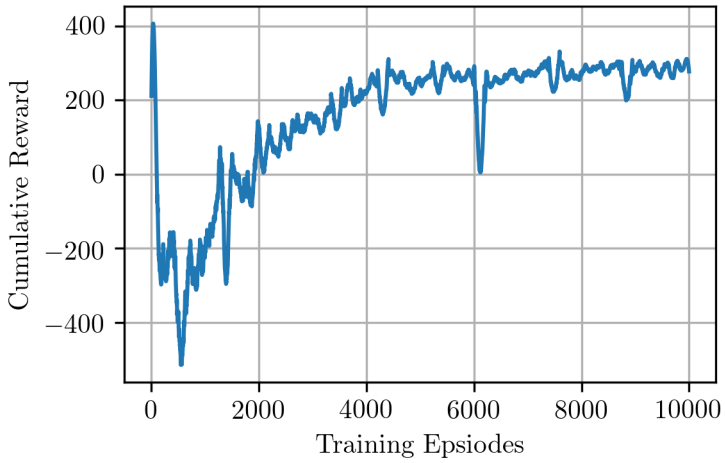
In order to further help with discovered challenges, a rational agent is also run for 100 episodes of the training. This rational agent alternates between estimation and control, and thrusts at the 8th step which is exactly when the reference changes. As in the stationkeeping environment, this helps the agent in the initial exploration phase by guaranteeing the discovery of the global minimum. Hyperperameters used in the Orbit Insertion scenario are listed in Table 6.2. The network inputs are the spacecraft state error with respect to the reference trajectory, the norm of the estimator covariance of these states, and the time to expected thrust. The first two components of the network inputs are identical to the functioning station-keeping environments. The time to expected thrust transmits the knowledge of the simulation time relative to the expected thrust maneuver to the network.

The agent is trained in the MOI environment with deterministic initial conditions and random initial conditions. Figure 6.2a shows the results of the RL algorithms reward as a function of episode of training in the deterministic case (smoothed using a SavitzkyGolay filter). The rational agent appears to be optimal—with cumulative rewards of 331—and the trained agent is able to replicate the correct maneuver despite dispersed initial conditions. Figure 6.2b shows the same reward plot with dispersed orbits for training. In the random case, the rational agent performs far less well with cumulative rewards averaging around -800. The trained agent outperforms the rational agent in the end of it's training with usually positive rewards sometimes reaching 300.

The orbit insertion environment posed challenges usually tied to the discrete nature of the thrust command. Yet with some knowledge of the model and by preventing clearly undesirable actions, a RL agent is able to optimize the necessary actions within 10,000 runs. Figure 6.3 shows the state and estimation errors and thrust maneuver for one of the dispersed runs. As could be expected, the agent first estimates to know its state error, then controls to the reference before thrusting at the correct step (8th step). Then it continues to alternate between estimation and control after the thrust in order to minimize reward losses. The thrust line shows the value of the



(a) Smoothed Reward during Deterministic Training



(b) Smoothed Reward during Dispersed Training

Figure 6.2: MOI Reward as a Function of Episode in Training

thrust vector that is applied during the maneuver. The three trajectories are pictured in Figure 6.1.

6.2.3 Reinforcement Learning for OpNav

6.2.3.1 OpNav Scenario Description

Having now shown the performance of RL for astrodynamics tasks which allow to successfully capture a spacecraft into Mars orbit, the natural next step is to perform on-orbit tasks. This is

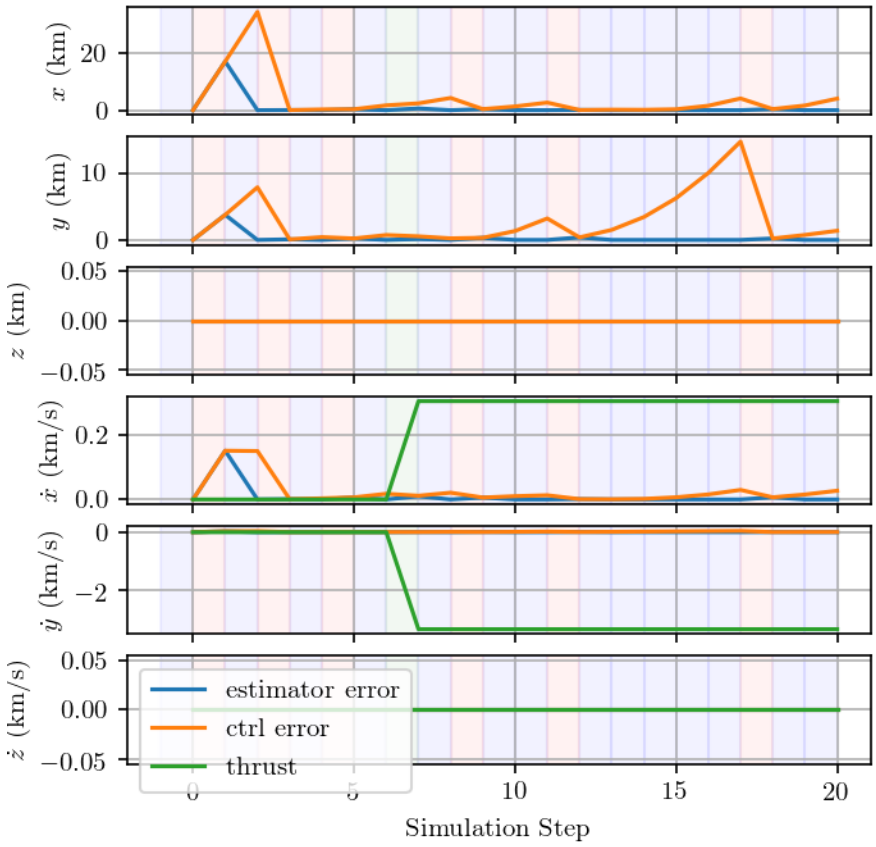


Figure 6.3: Mode Sequence for Dispersed Orbit Insertion

done by interfacing the *OpenAI* framework with the *Basilisk* simulation. More specifically, *Basilisk* simulations are instantiated in the RL environment which gives the agent in training the power to through the simulation taking different actions. In this framework actions become spacecraft modes which trigger the activation of different FSW modules in order to perform a given task. The agent is trained using Proximal Policy Optimization (PPO2)[187] unlike the MOI scenario which was trained using Deep-Q learning. PPO2 merges concepts from advantageous actor critic algorithm (A2C) and Trust Region Policy Optimization (TRPO) to control learning speed and improve training overall.[223]

This scenario assumes a spacecraft is on Mars orbit and needs to periodically point to the Sun

in order to charge it's batteries. In order to do so, it must first have a relatively good understanding of its position on orbit. In order to hone in on it's position, the spacecraft can point to the planet and perform optical navigation using the *Hough Circle* method.

Table 6.4: Actions for OpNav Agent

Actions	Description
Sun-Point	The spacecraft points towards the Sun using CSS data
OpNav	The spacecraft points towards Mars using *SPICE* and performs OpNav

The action of sun-point could have easily been replaced by another mode in *Basilisk*: Earth-point, science-point, thrust etc. The goal of this subsection is to show the ability to train RL agents using the developed OpNav tools in a general astrodynamics context. The expected behavior is to see the spacecraft oscillate between getting rewards and determining orbit. It's expected that the agent will be able to stay in OpNav mode the number of steps necessary to bring the error down reliably, then move to sun-point for a limited time in order to get a consistent reward.

Table 6.5: Observations for OpNav Agent

Observations	Description
Sun-Mars Angle	Cosine of Phase Angle (Sun-Mars-Spacecraft angle)
Normalized Covariance Diagonal Terms	Diagonal terms of covariance normalized by the norm of the state estimate

Table 6.5 describes the states that are observed by the agent. The first state allows the agent to have insight on the lighting conditions of the planet. Assuming the sun-direction is known (using CSS) and the planet position is known (*SPICE* is used in these simulations) the phase angle can be deduced. The only other information observed is the normalized covariance output by the OpNav filter. This will swell as long as there are no incoming measurements and is the only on-board tool to understand the uncertainty around the OD solution.

The nominal orbit is similar to previous simulations in this thesis: $a = 18,000$km, $e = 0.6$, $i = 10°$, $\Omega = 25°$, $\omega = 190°$, $f = 80°$. The variations on the orbit are added as cartesian parameters with dispersions given in Table 6.6.

Table 6.6: Initial Conditions Dispersons for OpNav

Parameter	Dispersion
$^{\mathcal{N}}\boldsymbol{r}_{\text{error}}$	$\mathcal{U}$(100,-100) km
$^{\mathcal{N}}\boldsymbol{v}_{\text{error}}$	$\mathcal{U}$(1,-1) km/s

These errors (of 200km and 2km/s) are relatively low with respect to the orbit. As stated previously the goal is for the agent to learn how to oscillate between the two modes in order to keep OD errors down and get rewards on a nominal orbit. Although more data would allow for the agent to learn over a broader set of situations, the desire is to prove the global feasibility rather than focus on a a specific set of actions and orbits.

Table 6.6 shows the dispersions on the initial conditions that are used for training. This allows to train on a wide set of errors orbit insertion introduces new challenges to overcome in order to robustly train agents for autonomous decision-making. These challenges revolve around the binary success criteria of the thrust maneuver. The ability to thrust is an action of equal weight to estimation or control modes. This encourages the agent in training to use it promptly, and very frequently fall into a local minimum that consists in thrusting immediately and attempting to minimize further errors with the control mode.

Table 6.7: Hyperparameters used in OpNav Training

Parameter	Value/Type
Number of Hidden Layers	1
Hidden Layer Depth	64
Hidden Layer Activation	ReLU
Output Layer Activation	Linear
Time in Each Mode	50 min
Learning Rate	0.04
Number of Training Episodes	100,000

A reward is only given to the agent when it goes into sun-pointing mode. This error is inversely quadratic with the spacecraft position error (difference between the truth and the position estimate):

$$r(\boldsymbol{e}) = \frac{1}{1 + \boldsymbol{e}^T \boldsymbol{e}} \tag{6.15}$$

where $\boldsymbol{e}$ is the state error. This means that the agent only sees the truth through the reward function. Not only does this most closely represent a real scenario (in which solar panels would not charge as well without proper OD), but it also aligns with the paradigm of providing simple cost functions that represent the spacecraft goals directly.

6.2.3.2 OpNav Scenario Results

Combining the *Basilisk-Vizard* simulation with RL provides novel results to both the OpNav and RL communities. The spacecraft successfully learns the expected behavior: it oscillates between a sun-pointing mode and a OpNav mode. One of the components that allows this is that the agent determines that two steps (each 50min long) allows to bring down the state estimation error through the use of *Hough Circles*.

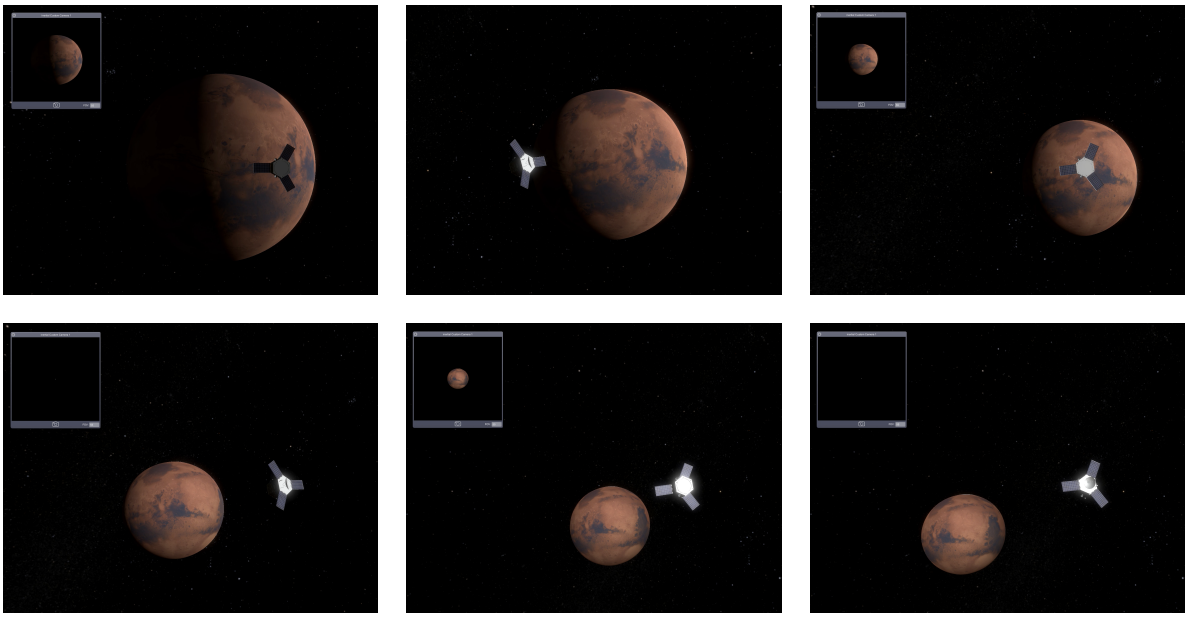
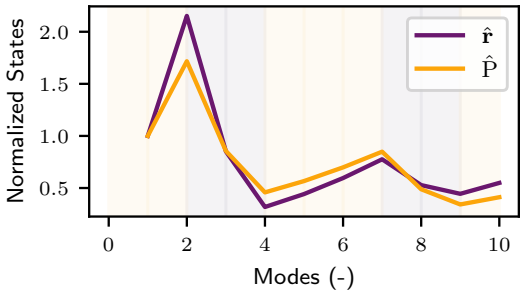


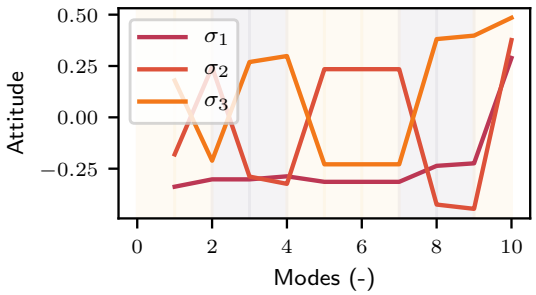
Figure 6.4: Spacecraft Changing Between Modes in order to Maximize Reward

Figure 6.4 shows the spacecraft moving between the different pointing modes while on orbit about Mars. According to the initial goal is formulated, the agent attempts to take in a maximal reward in the simulation time. The fear would be that the agent stays in the sun-pointing mode

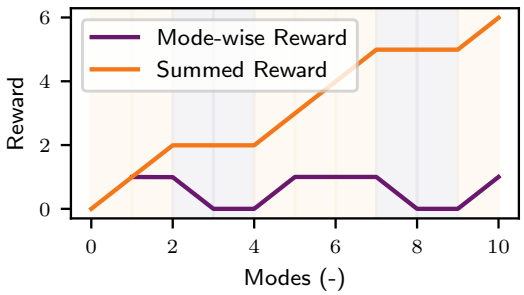
taking in rewards at every step despite a stead decrease it each reward value (this is a trivial local minima). Nonetheless, Figure 6.5 shows the performance and mode switching in more detail: as the state errors and covariances rise, the agent moves into OpNav mode for two steps (a total of 100min), this brings down the covariance and state errors which leads to a high reward in the pointing modes. The reward plot in Figure 6.5c shows the step-wise increases in the reward when the state error is low.



(a) Normalized state estimation



(b) Attitude changes



(c) Reward with different modes

Figure 6.5: States within Mode Sequence (Yellow Shade — Sun-Point, Blue Shade — OpNav)

Figure 6.5a shows how the *Hough Circle* method of OpNav brings the errors down in the state estimation. These values are normalized to be equal to one after the first step in order to show relative changes without scaling the values. Indeed, normalized covariances are given as observables in order to provide the agent with trends instead of physical values which are less numerically stable for optimization.

6.2.4 Conclusions

The work in this subsection illustrate the applicability of RL methods to spacecraft autonomy. It is first applied to a orbit insertion scenario as a proof-of-concept before being integrated with OpNav work developed in previous chapters. High-level autonomy designed through RL can also lead to fault injection during the simulation in order to train the agent to detect and respond to errors.

In the OpNav scenario, interfacing the stable-baselines[1] RL implementations with *Basilisk* simulations, alongside the use of Proximal Policy Optimization (PPO)[187] has shown promising results. Faster learning that better harness the environment characteristics allows for OpNav RL for the first time. These simulations, which combine a realistic spacecraft environment alongside flight-ready FSW stacks, are a key tool to the validation of such autonomous agents. With the new ability of being able to test them alongside all other FSW algorithms for rigorous analysis, the trust in their performance can increase and perhaps enable on-board use in the near future.

6.3 Convolutional Neural Networks for Center and Radius Finding

Apart from high-level decision-making, ML provides many other tools that can be harnessed for spacecraft autonomy. This subsection focuses on a low-level task which uses neural networks for image processing.

Supervised learning is a task in ML which parametrize models using a training dataset. The data contains the input the agent needs to process as well as the truth values associated with each data point. Although supervised learning has been very successful in recent years, it leads to two key limitations. The first limitation is that the quantity of data needed for training is often large and needs to be tagged with truth values (which can be difficult to provide). The second limitation is generalization: if an agent "over-fits" the data, its performance will not generalize well. In other words, the training data set may allow the net to extract undesirable features that do not

[1] https://stable-baselines.readthedocs.io/

generalize well to new and real data. Space imaging suffers from both of those issue for supervised learning in general: there are few real images to train on, and images depend largely on the camera parameters which do not necessarily generalize. The nominal solution to poor generalization — data augmentation — is poorly suited to space imaging because flipping, scaling, and rotational modifications may not be realistic. On the other hand, adding noise and artifacts in the image can be seen as a form of data augmentation specific to the OpNav application. This subsection intends to harness previous work in order to provide new results in the field of machine learning for OpNav.

6.3.1 Introduction to Neural Networks for Image Processing

Neural networks are a structured set of connected, simple processing elements (colloquially called neurons) operating on a given input. Through repeated non-linear transforms, a trained network can work as a linear classifier or learn a regression task. Although originally inspired by biology, the modern iterations have evolved into many distinct forms ranging from simple Feed-Forward Neural Networks (FFNN),[199] to Convolutional Neural Networks (CNN),[164] as well as Boltzmann Machines[198] and Bayesian Networks.[34] In general, these functions are trained by assuming that the spaces from which the input and outputs are pulled are stochastic. By defining a differentiable metric between outputs, an independent identical distribution of the input-output pairs is pulled from the data. In turn, the average derivative of the distribution is calculated to approximate the true gradient (output with respect to the input) to improve the networks function.[189] Theoretically, a neural network with sufficiently large hidden layers — which provide intermediate transformations between the inputs and the output — are universal function approximators.[138]

In the field of image processing and computer vision, CNNs have seen a great success from medical imagery[150] to digit recognition.[188] In navigation, CNNs are recently used for depth mapping[149] and Terrain Relative Navigation (TRN).[179] The field is constantly growing and evolving: developments with autoencoders[118] have shown promising denoising capabilities and super resolution,[211] while recent results display strong feature detection and generalization notably with the use of deep variational autoencoders.[96] Neural networks are also being developed in order to navigate

around other spacecraft for detumbling or docking by pairing physical test-beds with simulated environments to enhance performance.[119]

A common task for CNN in image processing is to locate and classify all objects in an image and provide bounding boxes, which is known as the object detection task. In the realm of object detection, modern variants of YOLO[177] and Faster R-CNN[82] are the state of the art: YOLO operates in realtime while Faster R-CNN produces higher quality object detection at a slower speed. These approaches operate on a image feature space produced by a large CNN known as a "backbone", rather then directly on a image. Residual Neural Networks[94] (ResNets) were originally tested and trained on the immense, general classification task ImageNet[60] as classifiers. ResNets allow for much deeper networks and better propagation of signal from the loss resulting in faster learning. Due to richness of their resulting features, ResNets have become nearly ubiquitous in the object detection and localization task as the aforementioned backbone, only losing out in the application to constrained compute environments.[101]

A strong natural image prior is placed on the backbone through training with a larger dataset. By utilizing the feature space of the trained backbone CNN, smaller networks have been shown to learning a different task with a much smaller dataset then would normally be required. This idea of reusing the priors of larger networks to make a new task easier is known as Transfer Learning.[201]

Using only real data to train a CNN for learning for space applications is practically intractable due to the relative sparsity of data and absence of labeled datasets on which to train. Transfer learning can not directly be used either because of the absence of a rich enough prior distribution which would require a large labeled dataset similar to the one to be trained on.

If armed with an accurately simulated environment, a well simulated camera, and faster-than-realtime speeds, CNNs for OpNav become a powerful option. CNNs allow for the processing of raw measurements and can learn the camera parameters as well as environmental parameters such as lightning and planetary features. Although visualizations continue to improve, the space environment is not the most difficult to model: lighting is purely unidirectional with parallel light rays, near celestial bodies are well mapped, and the background is very dark. With these concepts

in mind and the tools developed throughout this thesis, a first cut at CNNs for radius and centroid extraction are designed.

6.3.2 MarsNet Architecture

Within any network paradigm, sets of hidden layers of varying sizes (number of neurons per layer) can be activated through different activation functions such as the sigmoid function, hyperbolic tangent, or Rectified Linear Unit (ReLU). The activation function is responsible for transforming the summed weighted input of each neuron into an output for downstream layers. The ReLU activation function is a piecewise linear function that either outputs a scaled value of the input if it is positive, or outputs zero. It is the default activation function for many types of neural networks due to ease of training and better performances in a wide variety of applications. Given it's success throughout the literature — notably because it helps solve vanishing gradients[14] — ReLU is the activation function chosen for CNNs in this thesis.

However, space applications offer a challenging problem which is not seen in traditional applications of CNNs: the fact that often a non-zero part of the image is nearly black other then the noise of the sensor. After normalization, this provides an unusual distribution of pixel intensities as almost all are very weak or negative. For a normal ReLU, this is problematic as it will eliminate almost all the signal in the first layer. For this application, it was found that Leaky ReLU[136] performed well:

$$\text{ReLU}(x) = max\{\lambda x, x\}, \quad 0 < \lambda \ll 1 \tag{6.16}$$

Leaky ReLU — Equation (6.16) — doesn't completely eliminate the signal, yet has the same desired traits of the normal ReLU. Furthermore, if the weights become predominantly negative, the slight slope defined by λ transfers what would otherwise be strictly zero signal back to a positive domain and prevent "dead neurons".

Net optimization happens during the back-propagation step and where methods can either be adaptive (as Adam, Adagrad or RMSprop) or stochastic like commonly used Stochastic gradient descent (SGD). Although methods like Adam tend to perform well in the initial portion of training

it is outperformed by SGD at later stages of training. Despite some downsides, Adam is chosen for the training of the ResNets implemented given fast training speeds and high performance on sparse datasets thanks to per-parameter step-size.

The model was implemented and trained using *Pytorch*[1] which provides an array of test scripts and implemented methods to facilitate development. The higher level architecture is includes a convolutional layer, set of ResNet-sequences, and linear output layer:

- The first layer applied is a 2D-Convolution layer which takes in 3 channels, outputs 16, with a 3×3 kernel, 1×1 stride and padding
- A set of five ResNet sequence blocks are then defined each followed by a 2D, down sampling convolution (twice the depth and half the height and width).
- The output is then flattened, linearized, and activated with a "Leaky-ReLu" function defined as $\max\{\lambda x, x\}$ where $0 < \lambda \ll 0$.

Each model trained for 40 epochs with a batch size of 14, using a learning rate scheduler which reduces the learning rate after 5 epochs without improvement. The loss function used is Huber loss[102] because it is less sensitive to outlier predictions from the network, particularly in earlier epochs when the prediction error is arbitrarily large in the regression task. This allows to train with a larger learning rate without running into exploding gradients for this problem, similar in the approach to object detection.[83]

For each dimension size, three ResNet blocks are applied which each perform the follow sequence twice before summing the input and the output (identity operation central to the ResNet design) :

- Applies a 2D-Convolution to the input
- Activates the convolved input with a "Leaky-ReLU"
- Batch-normalizes the activated layer

[1] https://pytorch.org/

- Performs a $\frac{1}{2}$ dropout. In training this means zeroing each neuron with probability one-half, in evaluation weight each neuron output by half to get a cumulative contribution.

The details of the implementation as designed using the *Pytorch* library can be found in Appendix A. The size and number of ResNet blocks are taken from the literature — notably Reference 94 — in order to ensure a efficient and performant implementation. The specific additions are the use of Batch-normalization and Dropout. Batch-normalization makes weight normalization a part of the model architecture as it incorporates it within the net layers. Batch-normalization allows the use of higher learning rates engenders robustness to initialization and acts as a regularizer.[105] Dropout arbitrarily deactivates neurons in a layer during training. Deactivation of random neurons leads protects against overfitting by preventing intertwined feature detection.[98]

6.3.3 Training Data

As stated in the introduction to this subsection, the difficulty with using CNNs for OpNav lies primarily in the data. There needs to be an easy way to generate large quantities of realistic training data, and provide a test for the fully integrated algorithm. The simulation developed throughout this thesis provides the ability to do so with a large control over the noise of the data and the camera parameters.

Table 6.8: Orbital Monte-Carlo Dispersions

Parameter	Dispersion
Semi-Major Axis (km)	$\mathcal{N}[20,000, 3,000]$
Eccentricity	$\mathcal{U}[0.1, 0.6]$
Inclination (°)	$\mathcal{U}[-80, 80]$
True Anomaly (°)	$\mathcal{U}[0, 359]$
RAAN (°)	$\mathcal{U}[-40, 40]$
Off-pointing	$\sigma = [\mathcal{U}[-0.05, 0.05], \mathcal{U}[-0.05, 0.05], \mathcal{U}[-0.05, 0.05]]$

The dispersions used on the spacecraft orbit and position of the planet on the camera plane are given in Table 6.8. These cover a wide variety of orbits with inclinations nearly reaching from pole to pole and eccentricities going up to 0.6 to allow for many images to be generated in each

episode. Each scenario therefore generates 100 images according to the random generation of the orbit and position of the planet on the camera.

Depending on the goal, it is possible to vary camera parameters within the training set randomly as well as all simulation parameters. As a proof-of-concept and first iteration, the training data did not contain any noise and only dispersed the orbit of the spacecraft as well as the position of the planet on the camera frame. Figure 6.6 shows the total loss during training of the CNN in

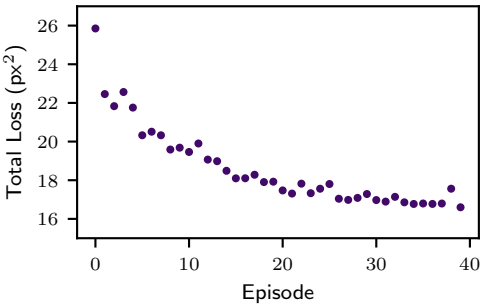


Figure 6.6: CNN Loss During Training

squared pixels over all episodes. The training was performed by splitting the images in 40 episodes each containing 5500 images. The plot shows the steady decline of the loss on the validation data over the episodes, as is desirable. An ideal plot would be monotone decreasing which, outside a few exceptions, is pictured.

Figure 6.7 shows the variety of images that the neural net has to train on. The selected images display some of the 90,000 images that were produced. The truth values for the circles are found by using the inverse transform used for OpNav in previous chapters. The images illustrate the variety of perspectives that can be achieved and the ease with which the data can be generated. It is also important to note that although Mars was chosen as a target, *Vizard* easily allows for other planets or bodies to be added instead.

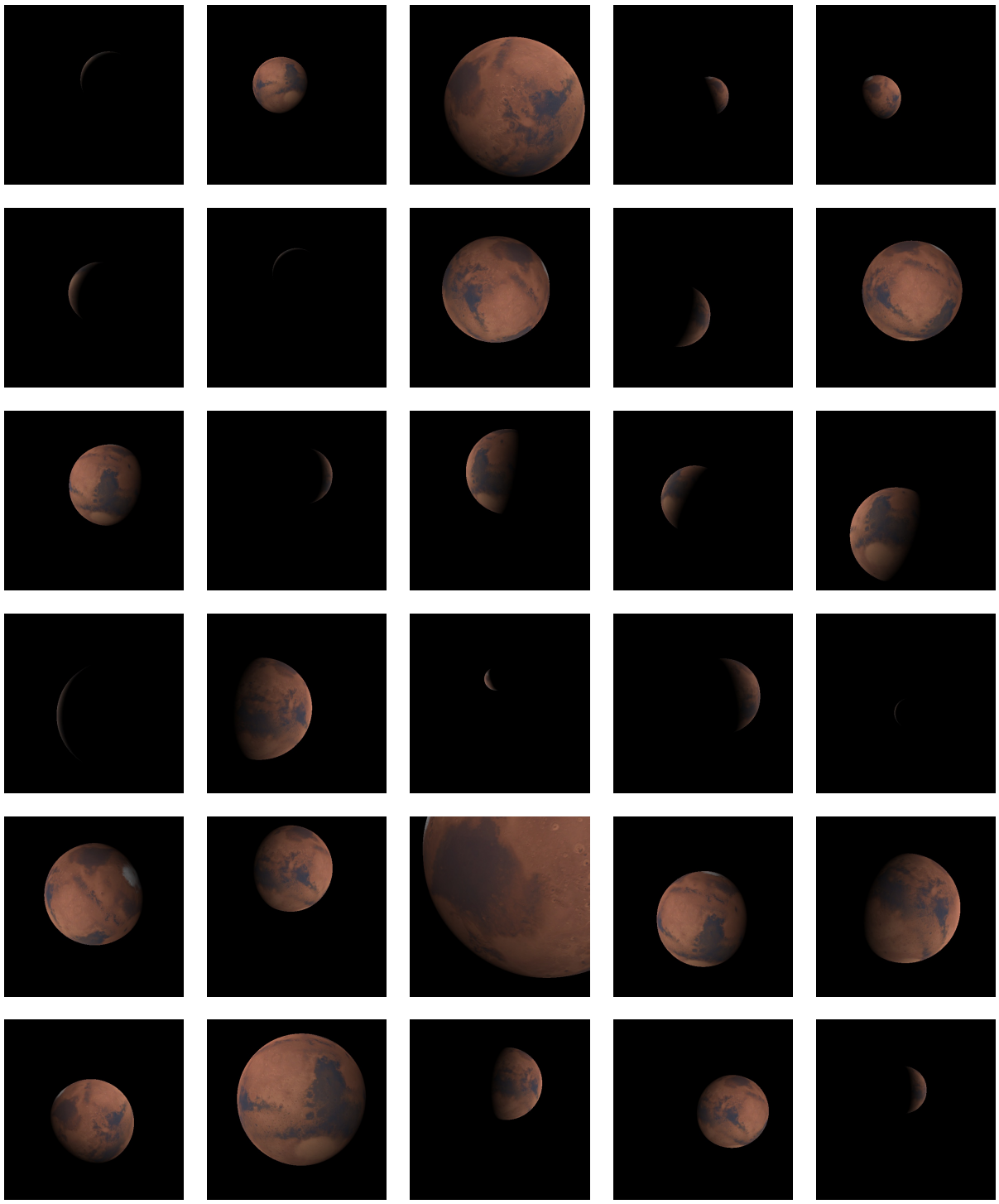


Figure 6.7: Samples from the CNN Training Data

6.3.3.1 Neural Net Integration and Initial Performance

Once trained the neural net can be tested on a subset of the data (10% isolated into the validation set) that is kept aside. Some of those results are seen in Figure 6.8. The trained CNN, although a first cut implementation, displays some positive results. Some of the images display very dim if not undetectable limbs to the eye, yet the CNN often produces very accurate and precise estimates. If no information is in the image, the algorithm guesses the center is in the middle of the sensor and provides a radius roughly equal to half of the image width. This shows that the CNN has learned offsets from the center and mean radius.

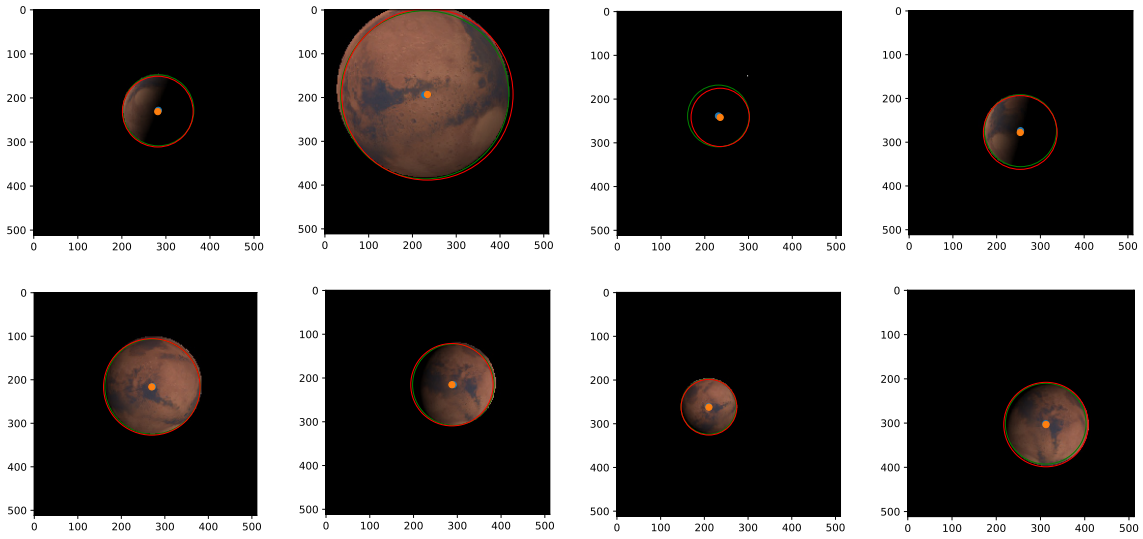


Figure 6.8: Trained CNN Evaluated on New Images

Figure 6.8 shows how fast the CNN was able to learn given a relatively small dataset. Some limitations in the data set do prevent strong generalization at the moment and will be improved upon. Before flying such an algorithm, it would be key to test it on real images or images with the instrument hardware in the loop. At the current stage of development such results are not available but in progress.

Once the CNN is trained in Python using the generated images, it can be incorporated as a *Basilisk* module through the *OpenCV* DNN library. This allows to load a net (exported in an

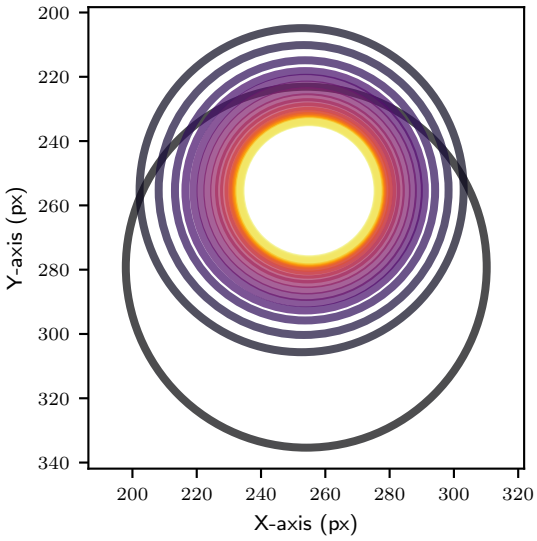
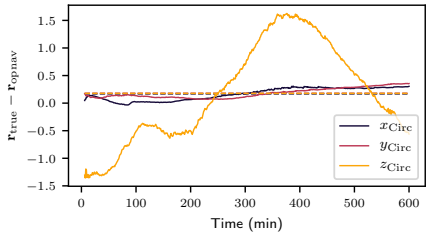


Figure 6.9: Circles Found by the CNN

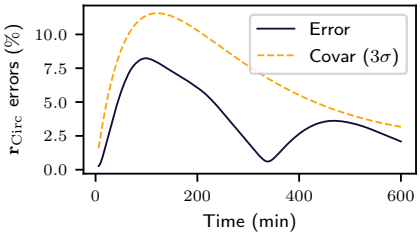
onyx format here), and read in an image to be processed. The process is implemented in the `centerRadiusCNN` module found in *Basilisk* under `fswAlgorithms/imageProcessing`. The circles found by the CNN for the identical reference OpNav scenario of Chapter 4. Results stemming from the simulations described in Tables 4.4, 4.5, 4.6 5.2, and 5.3 are shown in Figure 6.9 and perform similarly to *Hough Cirlces*.

Finally, when integrated into the full OpNav stack, the CNN performs well overall. This is seen in Figure 6.10 as the filter output is of the same order of magnitude as the other methods. The net does show some varying biases that are not akin to those of the other methods. Indeed the measurements seen in subfigure 6.10a show a large oscillation on the range which does not match the lighting conditions affects seen by *Hough* or *Canny*. This indicates that although the center finding is successful, the radius of the planet is not well matched.

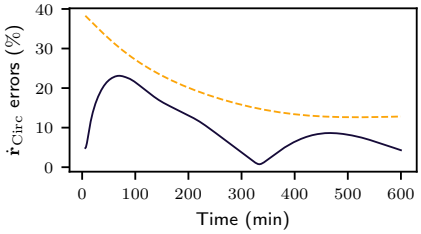
The results leave room for improvement, but are the first of it's kind. Neural nets for OpNav image processing to extract radius and centroid had yet to be developed and the framework opens the door to many more CNN architectures. By developing more nets, prior nets can be used on new OpNav problems hence developing the zoology of such CNNs.



(a) CNN Measurements



(b) CNN Position Errors



(c) CNN Velocity Errors

Figure 6.10: Measurements and State-Error Covariances for CNN

6.3.3.2 Enhancing Training and Results

The results presented encouraged more training in an environment that allows for more generalization. With the prior of orbit variability, the effect that needed to be added echos the work of Chapter 5: image corruptions. By training on a new set of data with image corruptions, the neural net can be expected to better generalize and extract more pertinent features for center and radius finding.

Table 6.9 shows the new set of dispersions used to generate data. Although very similar to the previous set seen in Table 6.8, these values constrain the images to be closer to the reference orbit, and with corruptions added to the images as in Chapter 5. The data is therefore constrained to a smaller torus surrounding the planet, notably by reaching less towards the polar orbits. Furthermore, the camera is pointed more directly at the planet which helps to generate more centered data as the pointing scheme will naturally provide much more of these images. This data set therefore aims to target the weaknesses of the previous iteration as it builds on the already working prior.

Most of all, the new data set adds corruptions and noise to the images. This is done with

Table 6.9: Orbital Monte-Carlo Dispersions

Parameter	Dispersion
Semi-Major Axis (km)	$\mathcal{N}[18,000, 1,000]$
Eccentricity	$\mathcal{U}[0.1, 0.5]$
Inclination (°)	$\mathcal{U}[-40, 40]$
True Anomaly (°)	$\mathcal{U}[-90, 90]$
Off-pointing	$\sigma = [\mathcal{U}[-0.01, 0.01], \mathcal{U}[-0.01, 0.01], \mathcal{U}[-0.01, 0.01]]$
Gaussian Blur	$\mathcal{U}[0, 3]$
Hot-Dead Pixels	$\mathcal{U}[0, 3]$
Cosmic Rays	$\mathcal{U}[0.5, 2]$
Blur	$\mathcal{U}[1, 5]$

the aim of enhancing the data and forcing the net to learn useful features rather than overfitting certain components of the image. Having noise and blur also provides more signal in each image to aid optimization hence partly solving the sparse data problem.

The new set of data improved the loss during training greatly. Staring from the prior of the previously trained net, the loss continued to decrease monotonously and ending at nearly half of the loss of the previous iteration as seen in Figure 6.11

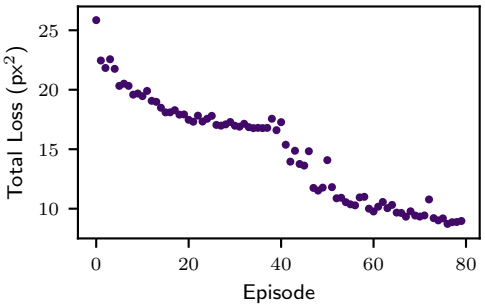
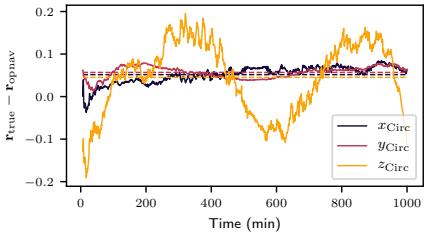
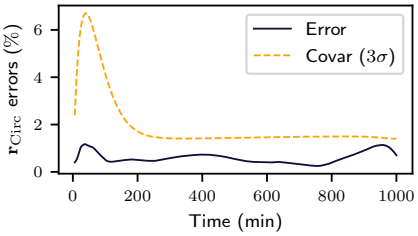


Figure 6.11: CNN Loss with Additional Training

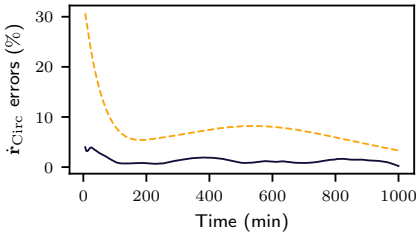
The results seen in Figure 6.12 are better than any of the previous methods. While both *Hough Cirlces* and *NIH-SVD* saw errors that reach 2% for position and 5% for velocity (Figures 4.13 and 4.12), the errors now stay within roughly 1% for both values. The covariances are also lesser than previously notably on the velocity components. This outlines an overall better filter performance than the previous net iteration, and better than both other methods studied.



(a) CNN Measurements



(b) CNN Position Errors



(c) CNN Velocity Errors

Figure 6.12: Measurements and State-Error Covariances for Enhanced CNN

A few important novelties are interesting with these results. With regard to robustness, the newly trained CNN produces the same results with noisy images or clean images. This stresses the fact that training under corrupt conditions has aided the weights to feature detection to converge towards the key components of the image: the limb and lit disk. One of the novelties of using *Hough Circles* was moving away from ellipses while centering the image in order to benefit from the measurement robustness. The downside, nonetheless, is still that camera distortions naturally falsify the circle positions. The net trained in this section does not suffer this default: the net trains over expected circle position given the geometry and hence learns the camera perturbations and distortions as a part of it's training. Finally, when looking at fault detection, it became clear that comparing methods with different strengths and weaknesses would help detect anomalies: if a first method works well with crescents and poorly with full disks while the second performs inversely well, combining them will output reliable faults and results. The CNN trained here shows in Figure 6.12a that the lighting conditions do not affect performance as it did with other methods, hence providing good supporting data in a multi-measurement paradigm.

Two possible directions can be taken to keep training these CNNs, the first being to model the camera accurately in simulation and train a net to capture its defects as well as possible, the second would aim to use a large set of cameras and generalize the circle fitting as much as possible. Both methods could provide interesting future results. The one caveat is that the speed of the network evaluation is one order of magnitude slower on general purpose compute hardware: roughly 100× faster than real-time instead of 1000×. This can be improved by decreasing the size of the network through training a smaller network to match this function, though it is not prohibitive for on-board use as is. However, by utilizing a hardware accelerator such as a graphics accelerator or a Field Programmable Gate Array (FPGA), one could easily perform these computations quickly and power efficiently. Furthermore, the use of half-precision floating point or even 8-bit integer arithmetic would enhance speed, use less power, and theoretically show nearly identical regression performance.[159]

6.4 Conclusions

The developments of both RL and CNNs for spacecraft autonomy and enhancing image processing is presented. Although just an introduction to the wealth of possibilities, ML presents a unique use-case that prior work in this thesis has enabled. Preliminary RL results show the possibility of training agents for decision-making and are applied to OpNav scenarios. Training RL agents in realistic scenarios while interacting with a simulated space environment is displayed for the first time.

In a second part, CNNs are integrated in a full OpNav FSW stack. By training under different camera models and orbits, the CNN has proven to outperform all other methods and has provided a new and robust way of doing image processing. Validation using real images as well as further training are still needed, and different net architectures can still be designed. The growing field of ML is being increasingly applied to aerospace, these results tie in to that body of work.

Chapter 7

Conclusions and Future Work

This final chapter recapitulates some of the goals and contributions achieved in this dissertation, and recommends some future work. These future efforts could not only improve the current state of the results, but also generate new research areas. On one hand, current results could see improvements in simulation quality, speed, and ease of use, on the other hand research areas might require new methods and analysis to prove feasibility of certain concepts. The chapter first covers the research deliverables that were named in the introduction, then covers some recommendations for future work.

7.1 Research Goals

The proposed research aims to enhance spacecraft autonomy. This is is achieved through novel filtering methods, noise quantification in image processing, novel software development, fault detection and mitigation, guidance analysis, and machine learning. Progress in this field will allow to manage spacecrafts on orbits such as the ones depicted in Figure 7.1 with increased autonomy. Simultaneously, new mission designs and objectives can be imagined and attempted. Small and nano satellites provide good test-beds to provide technology demonstration results, all the while opening space up to more people.

Full spacecraft autonomy is incremental and the work provided in this dissertation provides a few tools and key results that have further explored the field. Some of the goals achieved throughout this dissertation are listed in this section.

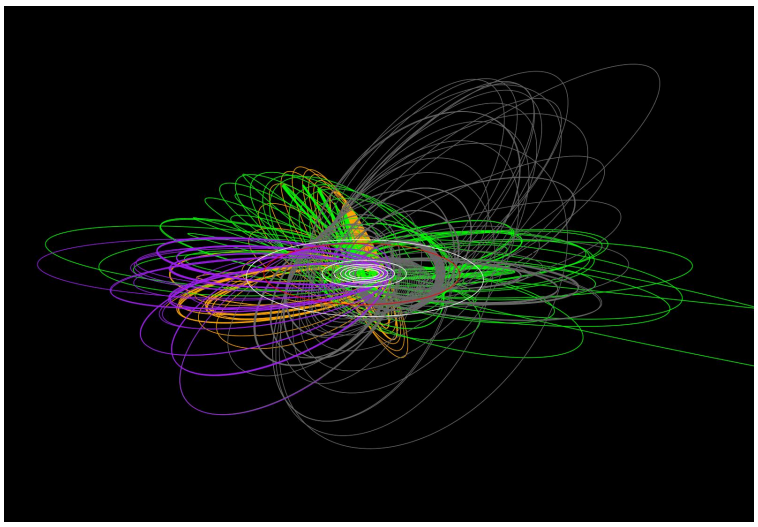


Figure 7.1: Cassini's many orbits "Ball of Yarn" Credit: NASA/JPL-Caltech

The first research deliverable presented in this thesis has been the link between the astrodynamics simulation *Basilisk* and the visualization tool *Vizard*. As a first open-source package of it's kind — generating high-fidelity spacecraft dynamics and generating synthetic images for navigation at Monte-Carlo speeds — the simulation is an over-arching contribution to the state-of-the-art. Without an open-source OpNav simulation which provides closed-loop image generation at faster-than-realtime speeds, none of the results could have been provided. The coupled nature (pointing and orbit determination) of the developed scenarios provide novel results and allows to couple attitude and orbit determination considerations in every simulation. In general, the work carried throughout this dissertation has followed a rigorous development and implementation cycle on par with mission ready FSW. In fact, the square-root unscented version of the Switch-Filter will fly on an interplanetary mission. Providing open-source and nearly flight ready code, this dissertation has responded to one of the goals cited in the introduction: produce robust and available GNC developments. Simultaneously, the set of software packages designed in the AVS lab provides a full suite of code which supports the testing and developments of FSW and simulation algorithms from cradle to grave.

The new design of heading determination filters using either coarse-sun-sensor or OpNav

measurements is also a contribution. By harnessing a method similar to how MRPs map to their shadow set to avoid singular representations of attitude, the filters have shown high performance results in a rigorous framework. The extension to OpNav in close proximity to a body being orbited has challenged some of the assumptions made and extended the use case of such a filter. For angles-only navigation using distant beacons, such a filter would also provide good estimates for headings.

The analysis carried using the *Hough* transform has indeed shown the applicability of a coarse, robotics inspired method to OpNav. The results approach state-of-the-art methods in ideal conditions, but greatly outperform them in a wide variety of faulty scenarios. The contribution of accepting the use of circle-fitting in order to use robust clustering methods and fully quantifying the effects of this limitation is shown. The performance of circle fitting is equal to the state-of-the-art comparative method, while speeds are matched as well. For certain OpNav cases, *Hough Circles* are a valid method of navigation, while a good fault detection tool even for more refined navigation. Monte-Carlo capability allows for sensitivity and performance analysis of the full OpNav stack, which challenges all the acting modules of FSW.

Finally, all the previous developments have enabled direct applicability of machine learning methods to astrodynamics. Both at a high-level with a reinforcement learning framework, and at a low-level with CNNs for center and radius finding, ML tools have displayed their applicability to the field. The modularity of the OpNav environment developed has also enabled seamless introduction of these modules to FSW. Integrating and testing the models with other FSW algorithms, using the same simulation and environment, and demanding the same level of performance is a crucial step towards the use of ML onboard spacecraft.

7.2 Future Work

This section covers some future goals that are enable by the thesis as a whole. Some of these goals are short-term improvements to existing results, while other will require more long-term efforts. Some short-term goals include simulation improvements. On the python level, simulation

architecture could provide speed boosts with less frequent updates on certain tasks. The camera model could also contain better logic in order to not decode and encode images unnecessarily.

The heading determination module could be used on the Sun and OpNav measurement simultaneously in order to produce a full attitude solution. While not a priority given the ubiquitous and highly-functional star-tracker, providing a full attitude solution could be used for instrument alignment calibration and better relative attitude guidance. In orbit determination, testing the speed and performance of a generalized *Hough* transform for ellipse fitting could be valuable. If so, improvements in the algorithm to reduce the search space could be harnessed. Regarding fault detection, many different existing or new methods could augment the existing suite. Clustering methods and machine learning can also be applied, providing a different approach to a well studied problem. SLAM techniques are also ripe for implementation in *Basilisk* and could provide a different approach altogether to some of the inherited navigation solutions. Already commonly used on the ground, and now for formation flying, the use-case of SLAM for asteroid navigation or even planetary navigation could be fruitful.

More involved additions include the natural extension of the work to Terrain Relative Navigation (TRN) and Entry Decent and Landing (EDL). In this field, new filters have been recently implemented to link features maps to detected features robustly.[147] With some additions, the visualization can integrate dynamic textures which will allow to keep running fast simulations with fast changing environments. With more developments the provided simulation will allow for end-to-end EDL simulations.

In parallel, thanks to the general nature of the *Basilisk-Vizard* interface, other visualization can be paired and used on exactly the same scenarios as developed in this thesis. This can allow to determine the robustness of the astrodynamics to visualizations and physical models of the environment. More specific to *Basilisk-Vizard*, developments for small body navigation are imminent. The Colorado Center for Astrodynamics Research (CCAR) at University of Colorado, Boulder is a leading hub in astrodynamics and small body navigation. Combining the research of students and professors into the framework developed in this dissertation would greatly benefit the field and

enhance research capabilities.

Finally, the work left to do in ML is vast. The reinforcement learning component alone, contains many challenges and are being tackled by AVS lab students as well as JPL engineers. This thesis has provided a stepping stone towards these methods, but more work is yet to be done. With regard to neural networks for OpNav, the number of implementations and net functions is unlimited. Different kinds of neural nets can provide various learning speeds and capabilities, while the training and validation data continues to be diversified. From covariance estimation, to direct spacecraft pose-estimation off of raw data, the limits are primarily dictated by time and computation power.

Bibliography

[1] J. Alcorn and H. Schaub. Simulating attitude actuation options using the basilisk astrodynamics software architecture. In 67th International Astronautical Congress, Guadalajara, Mexico, Sept. 26–30 2016.

[2] John Alcorn, Cody Allard, and Hanspeter Schaub. Fully-coupled dynamical jitter modeling of variable-speed control moment gyroscopes. In AAS/AIAA Astrodynamics Specialist Conference, Stevenson, WA, Aug. 20–24 2017. Paper No. AAS-17-730.

[3] John Alcorn, Cody Allard, and Hanspeter Schaub. Fully coupled reaction wheel static and dynamic imbalance for spacecraft jitter modeling. AIAA Journal of Guidance, Control, and Dynamics, 41(6):1380–1388, 2018.

[4] Brandon Alexander. Robot web tools [ros topics]. IEEE Robotics and Automation Magazine, 19(4):20 – 23, December 2012.

[5] Cody Allard, Manuel Diaz-Ramos, Patrick W. Kenneally, Hanspeter Schaub, and Scott Piggott. Modular software architecture for fully-coupled spacecraft simulations. Journal of Aerospace Information Systems, 2018. (in press).

[6] Cody Allard, Manuel Diaz-Ramos, and Hanspeter Schaub. Spacecraft dynamics integrating hinged solar panels and lumped-mass fuel slosh model. In AIAA/AAS Astrodynamics Specialist Conference, Long Beach, CA, Sept. 12–15 2016.

[7] Cody Allard, Hanspeter Schaub, and Scott Piggott. General hinged solar panel dynamics approximating first-order spacecraft flexing. AIAA Journal of Spacecraft and Rockets, 55(5):1290–1298, Sept.–Oct. 2018.

[8] Cody J. Allard. Modular Software Architecture for Complex Multi-Body Fully-Coupled Spacecraft Dynamics. PhD thesis, University of Colorado, Boulder, 2018.

[9] Shawn Allgeier, Matt Mahin, and Norman Fitz-Coy. Design and Analysis of a Coarse Sun Sensor for Pico-Satellites. American Institute of Aeronautics and Astronautics, 2017/12/19 2009.

[10] Christine Anderson. Lewis spacecraft mission failure investigation board. Final report, Air Force Research Laboratory, 1998.

[11] Andrew W. Appel and Zhong Shao. Empirical and analytic study of stack versus heap cost for languages with closures. Journal of Functional Programming, 6(1):47–74, January 1996.

[12] B. Schutz B. Tapley and G. Born. Statistical Orbit Determination. Number ISBN 9780126836301. Elsevier Academic Press, 2004.

[13] Richard H. Battin. An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition. American Institute of Aeronautics and Astronautics, 2019/01/10 1999.

[14] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):157–166, March 1994.

[15] M. Benn and J.L. Jørgensen. Autonomous vision-based detection of non-stellar objects flying in formation with camera point of view. International Journal of Space Science and Engineering, Vol 2, Issue 1(2048-8459):49–62, 2014.

[16] Benjamin Bercovici. Mapping And Navigation Of Small Bodies In The Presence Of Uncertainty. PhD thesis, Aerospace Engineering Sciences Department, University of Colorado, Boulder, CO, May 2019.

[17] Shyam Bhaskaran. Autonomous Navigation for Deep Space Missions.

[18] Shyam Bhaskaran, Sumita Nandi, and Stephen Broschart. Small body landings using autonomous onboard optical navigation. The Journal of the Astronautical Sciences, 58(3):409–427, July-September 2011.

[19] Dylan Boone, Julie Bellerose, and Duane Roth. Resolution of orbit determination prediction instabilities at titan during cassini's solstice mission. In 26th International Symposium on Space Fight Dynamics, Matsuyama, Japan, June 3-9 2017. JPL, NASA.

[20] Stoian Borissov and Daniele Mortari. Centroiding and sizing optimization of ellipsoid image processing using nonlinear-least squares. Number 18-229, 2018.

[21] Nicholas Bradley and Shyam Bhaskaran. Navigation accuracy at jupiter and saturn using optical observations of planetary satellites. In SpaceFlight Mechanics Conference, number AAS 19-231, Maui, Hawaii, 2019.

[22] William G. Breckenridge and Alfred J. Treder. In-flight gyro drift rate calibration on the viking orbiters. Journal of Guidance, Control, and Dynamics, 1(6):433–439, 2019/01/29 1978.

[23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. pages 1–4, 2016.

[24] Jonathan M. Cameron. Next generation simulation framework for robotic and human space missions. Number 5151 in AIAA SPACE 2012 Conference and Exposition, Pasadena, California, September 2012 2012. Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91107.

[25] J Canny. A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell., 8(6):679–698, June 1986.

[26] Paolo Cappuccio, Cody Allard, and Hanspeter Schaub. Fully-coupled spherical modular pendulum model to simulate spacecraft propellant slosh. In AAS/AIAA Astrodynamics Specialist Conference, Snowbird, UT, August 19–23 2018. Paper No. AAS-18-224.

[27] Russell J. Carpenter and Christopher N. D'Souza. Navigation filter best practices. Technical Report 20180003657, NASA Langley Research Center; Hampton, VA, United States, April 2018.

[28] John M. Carson, Michelle M. Munk, Ronald R. Sostaric, Jay N. Estes, Farzin Amzajerdian, James B. Blair, David K. Rutishauser, Carolina I. Restrepo, Alicia M. Dwyer-Cianciolo, George Chen, and Teming Tse. The SPLICE Project: Continuing NASA Development of GN&C Technologies for Safe and Precise Landing.

[29] John M. Carson, Carl Seubert, Farzin Amzajerdian, Chuck Bergh, Ara Kourchians, Carolina Restrepo, Carlos Y. Villalpando, Travis O'Neal, Edward A. Robertson, Diego F. Pierrottet, Glenn D. Hines, and Reuben Garcia. COBALT: Development of a Platform to Flight Test Lander GN&C Technologies on Suborbital Rockets. American Institute of Aeronautics and Astronautics, 2018/01/28 2017.

[30] Anthony R Cassandra. A Survey of POMDP Applications. Uncertainty in Artificial Intelligence, pages 472–480, 1997.

[31] Francesco Castellini and David Antal-Wokes. Far approach optical navigation and comet photometry for the rosetta mission. In In Proceedings of the 25th ISSFD, number 1, pages 1–19, 2015.

[32] G. H. Chapman, R. Thomas, R. Thomas, K. J. C. S. Meneses, T. Q. Yang, I. Koren, and Z. Koren. Single event upsets and hot pixels in digital imagers. In 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pages 41–46, Oct 2015.

[33] Yih-Kanq Chen, Thomas Squire, Bernard Laub, and Michael Wright. Monte Carlo Analysis for Spacecraft Thermal Protection System Design.

[34] Jie Cheng and Russell Greiner. Comparing bayesian network classifiers. In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI'99, pages 101–108, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[35] Y. Cheng, J. Goguen, A. Johnson, C. Leger, L. Matthies, M. S. Martin, and R. Willson. The mars exploration rovers descent image motion estimation system. IEEE Intelligent Systems, 19(3):13–21, May 2004.

[36] Binoy Chheda, Agamemnon Crassidis, and Wayne Walter. Attitude Estimation Using an Accelerometer and Rate Gyro Based Device. American Institute of Aeronautics and Astronautics, 2019/01/29 2006.

[37] Richard Y. Chiang. Star Tracker Rate Estimation with Kalman Filter Enhancement.

[38] Richard Y. Chiang and Tom Tsao. Gyroless 3-Axis Sun Acquisition via Sun Sensors Only Unscented Kalman Filter Estimation. American Institute of Aeronautics and Astronautics, 2017/12/19 2013.

[39] Steve A Chien, Daniel Tran, Gregg Rabideau, Steve R Schaffer, Dan Mandl, and Stuart Frye. Timeline-Based Space Operations Scheduling with External Constraints. Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS), (Icaps):34–41, 2010.

[40] J. Christian. An on-board image processing algorithm for a spacecraft optical navigation sensor system. AIAA Space Conference and Exposition, Anaheim, CA, 2010. AIAA.

[41] J. Christian. Onboard image-processing algorithm for a spacecraft optical navigation sensor system. Journal of spacecraft and rockets, 49(2), 2012.

[42] J. Christian. Optical navigation using planet's centroid and apparent diameter in image. Journal of guidance, control, and dynamics, 38(2), 2015.

[43] J.A. Christian. Accurate planetary limb localization for image-based spacecraft navigation. Journal of Spacecraft and Rockets, Vol. 54(No. 3):pp 708–730, 2017.

[44] J.A. Christian. Starnav: Autonomous optical navigation of a spacecraft by the relativistic perturbation of starlight. Sensors, 19(4064):doi:10.3390/s19194064, 2019.

[45] John Christian and Shane B. Robinson. (preprint) aas 16-151 observations on the geometry of horizon-based optical navigation. 2016.

[46] John A. Christian. Optical Navigation for a Spacecraft in a Planetary System. PhD thesis, The University of Texas at Austin, May 2010.

[47] John A. Christian. Optical navigation using iterative horizon reprojection. Journal of Guidance, Control, and Dynamics, 39(5):1092–1103, 2019/01/17 2016.

[48] John A. Christian. Optical navigation using iterative horizon reprojection. Journal of Guidance, Control, and Dynamics, 39(5):1092–1103, 2016.

[49] John A. Christian and Shane B. Robinson. Noniterative horizon-based optical navigation by cholesky factorization. Journal of Guidance, Control, and Dynamics, 39(12):2757–2765, 2019/01/17 2016.

[50] John A. Christian and Shane B. Robinson. Noniterative horizon-based optical navigation by cholesky factorization. Journal of Guidance, Control, and Dynamics, 39(12):2757–2765, 2016.

[51] Alicia D Cianciolo, Robert W Maddock, Jill L Prince, Angela Bowes, Richard W Powell, Joseph P White, Robert Tolson, O Shaughnessy, and David Carrelli. Autonomous aerobraking development software : Phase 2 summary. 2013 AAS/AIAA Astrodynamics Specialist Conference, pages 1–16, 2018.

[52] D. Claus and A. W. Fitzgibbon. A rational function lens distortion model for general cameras. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, pages 213–219 vol. 1, June 2005.

[53] Gregory Cohen. Event-based sensing for space situational awareness. The Journal of the Astronautical Sciences, pages 1–17, 2017.

[54] Mar Cols Margenet, Patrick W. Kenneally, Hanspeter Schaub, and Scott Piggott. Simulation of heterogeneous spacecraft and mission components through the black lion framework. In John L. Junkins Dynamical Systems Symposium, College Station, TX, May 20–21 2018. No. 7.

[55] John L. Crassidis and F. Landis Markley. Three-axis attitude estimation using rate-integrating gyroscopes. Journal of Guidance, Control, and Dynamics, 39(7):1513–1526, 2016.

[56] John L. Crassidis and F. Landis Markley. Unscented filtering for spacecraft attitude estimation. Journal of Guidance, Control, and Dynamics, 26(4), July-August 2016.

[57] John L. Crassidis, F. Landis Markley, and Yang Cheng. Survey of nonlinear attitude estimation methods. Journal of Guidance, Control, and Dynamics, 30(1):12–28, 2007.

[58] Anton H. J. de Ruiter, Long Tran, Balaji Shankar Kumar, and Andriy Muntyanov. Sun vector-based attitude determination of passively magnetically stabilized spacecraft. Journal of Guidance, Control, and Dynamics, 39(7):1551–1562, 2016.

[59] J. Delaune, G. Le Besnerais, T. Voirin, J.L. Farges, and C. Bourdarias. Visual-inertial navigation for pinpoint planetary landing using scale-based landmark matching. Robotics and Autonomous Systems, 78:63 – 82, 2016.

[60] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009.

[61] Mehregan Dor and Panagiotis Tsiotras. ORB-SLAM Applied to Spacecraft Non-Cooperative Rendezvous. American Institute of Aeronautics and Astronautics, 2018/05/21 2018.

[62] Randal K. Douglas and Jason L. Speyer. Robust fault detection filter design. Journal of Guidance, Control, and Dynamics, 19(1):214–218, 1996.

[63] Richard O. Duda and Peter E. Hart. Use of the hough tranformation to detect lines and curves in pictures. Graphics and Image Processing, 15(1), January 1972.

[64] Scott G. Edgington and Linda J. Spilker. Cassini's grand finale. Nature Geoscience, 9(7):472–473, 2016.

[65] Todd A. Ely and Jill Seubert. One-way radiometric navigation with the deep space atomic clock. In AAS/AIAA Space Flight Mech. Meeting, number AAS 15-384, 2015.

[66] J. Enright. Moon-tracking modes for star trackers. Journal of guidance, control, and dynamics, 22(1), 2010.

[67] Gregory Flandin et al. Maturing Vision Based Navigation Solutions to Space Exploration.

[68] Jamie A. Kennea et al. Controlling the Swift XRT CCD temperature via passive cooling. In Oswald H. W. Siegmund, editor, UV, X-Ray, and Gamma-Ray Space Instrumentation for Astronomy XIV, volume 5898, pages 341 – 351. International Society for Optics and Photonics, SPIE, 2005.

[69] K. Gwinner et al. The high resolution stereo camera (hrsc) of mars express and its approach to science analysis and mapping for mars and its satellites. Planetary and Space Science, 126:93 – 138, 2016.

[70] Masateru Ishiguro et al. The hayabusa spacecraft asteroid multi-band imaging camera (amica). Icarus, 207(2):714 – 731, 2010.

[71] Flandin, Polle, Frapard, Vidal, Philippe, and Voirin. Vision based navigation for planetary exploration. Number 052, Breckenridge, CO, February 4 2009. AIAA/AAS.

[72] Cyrus Foster, Henry Hallam, and James Mason. Orbit determination and differential-drag control of Planet Labs cubesat constellations. Advances in the Astronautical Sciences, 156:645–657, 2016.

[73] Wei-Ting Chou. Tendon Cato Fu-Yuen Hsiao and Carla Rebelo. Coarse Sun Acquisition Only with Sun Sensors for Micro Satellites. AAS 15-319, 2015.

[74] Jesse Fusco, Sean Shan-Min Swei, and Robert Nakamura. Sun Safe Mode Controller Design for LADEE. American Institute of Aeronautics and Astronautics, 2017/12/19 2015.

[75] Thomas K. Gaisser, Ralph Engel, and Elisa Resconi. Cosmic Rays and Particle Physics. Cambridge University Press, second edition, June 2016.

[76] O. Barnouinand R. Gaskell and E. Kahn. Assessing the quality of topography from stereo-photoclinometry. 2014.

[77] R.W. Gaskell. Optical navigation near small bodies. In 21st, AAS/AIAA Spaceflight Mechanics Meeting; 2011, volume 140 of ADVANCES IN ASTRONAUTICAL SCIENCES, pages 1705–1718, San Diego, Calif., 2011. Published for the American Astronautical Society by Univelt.

[78] R. F. Gates and K. J. McAloon. Precision star tracker utilizing advanced techniques and materials. Journal of Spacecraft and Rockets, 13(10):594–599, 1976.

[79] Brian Gaudet and Roberto Furfaro. Robust Spacecraft Hovering Near Small Bodies in Environments with Unknown Dynamics Using Reinforcement Learning. 2012 AIAA/AAS Astrodynamics Specialist Conference, (August):1–20, 2012.

[80] Sugata Ghosal and Rajiv Mehrotra. Orthogonal moment operators for subpixel edge detection. Pattern Recognition, 26(2):295 – 306, 1993.

[81] Christopher J. Gioia and John A. Christian. Gyro bias estimation using interior star angles for manual attitude determination. Journal of Spacecraft and Rockets, 54(2):513–522, 2019/01/29 2016.

[82] Ross Girshick. Fast r-cnn. In The IEEE International Conference on Computer Vision (ICCV), December 2015.

[83] Ross B. Girshick. Fast r-cnn. 2015 IEEE International Conference on Computer Vision (ICCV), pages 1440–1448, 2015.

[84] Anthony Giunta, Steven Wojtkiewicz, and Michael Eldred. Overview of Modern Design of Experiments Methods for Computational Simulations (Invited).

[85] Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Entropy-based dark frame subtraction. In Proceedings of the Image Processing, Image Quality, Image Capture Systems Conference (PICS), pages 293–298, 22–25 April 2001.

[86] Yanping Guo and Robert W. Farquhar. New horizons pluto–kuiper belt mission: design and simulation of the pluto–charon encounter. Acta Astronautica, 56(3):421 – 429, 2005.

[87] Yanping Guo and Robert W. Farquhar. Baseline design of new horizons mission to pluto and the kuiper belt. Acta Astronautica, 58(10):550 – 559, 2006.

[88] Bruce Hapke. Bidirectional reflectance spectroscopy: 6. effects of porosity. Icarus, 195(2):918 – 926, 2008.

[89] Ann et al Harch. Accommodating Navigation Uncertainties in the Pluto Encounter Sequence Design, pages 427–487. Springer International Publishing, Cham, 2017.

[90] Gabe D. Rogers; Alice Bowman; Ann Harch. New horizons 2014mu69 (ultima thule) flyby design and execution. In AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI, January 13–17 2019. Paper No. AAS-19-227.

[91] Andrew Harris. Towards reinforcement learning techniques for spacecraft autonomy. AAS Guidance, Navigation and Control Meeting, pages 1–10, 2018.

[92] Andrew Harris, Thibaud Teil, and Hanspeter Schaub. Spacecraft decision-making autonomy using deep reinforcement learning. In AAS Spaceflight Mechanics Meeting, Maui, Hawaii, Jan. 13–17 2019. Paper No. AAS-19-447.

[93] David H. Hathaway, Thibaud Teil, Aimee A. Norton, and Irina Kitiashvili. The sun's photospheric convenction spectrum. The Astrophysical Journal, 811(2), September 2015.

[94] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.

[95] R. Hermann and A. Krener. Nonlinear controllability and observability. IEEE Transactions on Automatic Control, 22(5):728–740, October 1977.

[96] Irina Higgins, Loic Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning, 2016.

[97] Jacob Hikes, Andrew J. Liounis, and John A. Christian. Parametric covariance model for horizon-based optical navigation. Journal of Guidance, Control, and Dynamics, Vol. 40(No. 1):Engineering Notes, January 2017.

[98] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. CoRR, abs/1207.0580, 2012.

[99] Greg N. Holt, Christopher N. D'Souza, and David Saley. Orion optical navigation progress toward exploration mission 1. In AIAA SciTech Forum, 2018 Space Flight Mechanics Meeting, Kissimmee, Florida, 8–12 January 2018. NASA Johnson Space Center.

[100] D. Hong. The History of Apollo On-board Guidance, Navigation, and Control, volume 43, pages 270–300. Univelt, 1976.

[101] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. ArXiv, abs/1704.04861, 2017.

[102] Peter J Huber. Robust estimation of a location parameter. In Breakthroughs in statistics, pages 492–518. Springer, 1992.

[103] Robert O. Hughes. Monte carlo analysis of satellite beam pointing errors. Journal of Guidance, Control, and Dynamics, 15(1):35–39, 1992.

[104] R. L. HUSTON. Twin-gyro attitude control systems. Journal of Spacecraft and Rockets, 3(7):1136–1138, 2019/01/29 1966.

[105] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR, abs/1502.03167, 2015.

[106] R. Isermann. Supervision, fault-detection and fault-diagnosis methods - an introduction. Control Engineering Practice, 5(5):639 – 652, 1997.

[107] T. Iskenderian. Lessons learned from selecting and testing spaceflight potentiometers. Number 19940028812. JPL, NASA. Lewis Research Center, The 28th Aerospace Mechanisms Symposium; p. p 339-358; NASA-CP-3260, 1994.

[108] S. Bhaskaran J.E. Riedel. Using autonomous navigation for interplanetary missions: The validation of deep space 1. Technical report.

[109] J.Eggert. Imaging sensor emulation and dynamics simulation for pil/hil. Technical report, Astos Solutions GmbH, Meitnerstraße 8 70563 Stuttgart, Germany.

[110] John L. Jørgensen, Troelz Denver, and Maurizio Betto. Microasc a miniature star tracker. Technical report, Technical University of Denmark, Ørsted, jlj@oersted.dtu.dk.

[111] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering, 82(Series D):35–45, 1960.

[112] Mike Kasper, Nima Keivan, Gabe Sibley, and Christoffer Heckman. Light source estimation with analytical path-tracing.

[113] Patrick W. Kenneally, Scott Piggott, and Hanspeter Schaub. Basilisk: A flexible, scalable and modular astrodynamics simulation framework. In 7th International Conference on Astrodynamics Tools adn Techniques (ICATT), DLR Oberpfaffenhofen, Germany, Nov. 6–9 2018.

[114] Patrick W. Kenneally and Hanspeter Schaub. High geometric fidelity modeling of solar radiation pressure using graphics processing unit. In AAS/AIAA Spaceflight Mechanics Meeting, pages 2577–2587, Napa Valley, California, Feb. 14–18 2016. Paper No. AAS-16-500.

[115] Patrick W. Kenneally and Hanspeter Schaub. Modeling solar radiation pressure with self-shadowing using graphics processing unit. In AAS Guidance, Navigation and Control Conference, Breckenridge, CO, Feb. 2–8 2017. Paper AAS 17-127.

[116] Patrick W. Kenneally and Hanspeter Schaub. Parallel spacecraft solar radiation pressure modeling using ray-tracing on graphic processing unit. In International Astronautical Congress, Adelaide, Australia, Sept. 25–29 2017. Paper No. IAC-17,C1,4,3,x40634.

[117] Richard A. Kerr. It's official—voyager has left the solar system. Science, 341(6151):1158–1159, 2013.

[118] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[119] Mate Kisantal. Satellite pose estimation challenge: Dataset, competition design and results, 2019.

[120] J. Z. Kolter, Youngjun Kim, and A. Y. Ng. Stereo vision and terrain modeling for quadruped robots. In 2009 IEEE International Conference on Robotics and Automation, pages 1557–1564, May 2009.

[121] Daniel G. Kubitschek. Impactor spacecraft targeting for the deep impact mission to comet tempel 1. Number 03-615. Astronautical Society, 2003.

[122] Daniel G. Kubitschek. Impactor spacecraft encounter sequence design for the deep impact mission. Jet Propulsion, pages 1–14, 2005.

[123] Daniel G. Kubitschek. Deep impact autonomous navigation : the trials of targeting the unknown. 29th Annual AAS Guidance and Control Conference, Breckenridge, Colorado, February 4-8, 2006., Pasadena, CA : Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2006., 2006.

[124] W. Henry Lambright and Agnes Gereben Schaefer. The political context of technology transfer: Nasa and the international space station. Comparative Technology Transfer and Society, 2(1):1–24, 2004.

[125] D.S. Lauretta and et al. Osiris-rex: Sample return from asteroid (101955) bennu. Space Science Reviews, 212(1-2):925–984, 2017.

[126] V. F. Leavers. Which hough transform? pp. 250-264 Vol. 58 No. 2, Department of Physics, King's College, Strand, London WC2R 2LS, United Kingdom, September 1993.

[127] E. Lefferts, F. Markley, and M. Shuster. Kalman filtering for spacecraft attitude estimation.

[128] S. Li. Image processing algorithms for deep-space autonomous optical navigation. The Journal of Navigation, 66(605-623), 2013.

[129] Shuang Li, Ruikun Lu, Liu Zhang, and Yuming Peng. Image processing algorithms for deep-space autonomous optical navigation. Journal of Navigation, 66(4), 2013.

[130] Yuxi Li. Deep reinforcement learning. CoRR, abs/1810.06339, 2018.

[131] C. C. Liebe. Star trackers for attitude determination. IEEE Aerospace and Electronic Systems Magazine, 10(6):10–16, June 1995.

[132] Christopher S. Lim and Abhinandan Jain. Dshell++: A component based, reusable space system simulation framework. Third IEEE International Conference on Space Mission Challenges for Information Technology, Jet Propulsion Laboratory California Institute of Technology, 2009.

[133] A. Liounis. Autonomous navigation system performance in the earth-moon system. AIAA Space Conference and Exposition, San Diego, CA, September 2013. AIAA.

[134] M. G. Lyons and D. B. Debra. A reduced state estimator for orbital heading reference. Journal of Spacecraft and Rockets, 11(2):97–100, 1974.

[135] Y. Ma, S. Soatto, J. Košecká, and S. Sastry. An Invitation to 3-D Vision: From Images to Geometric Models, pages pp. 49–59. Springer, New York, 2010.

[136] Awni Y. Hannun Maas, Andrew L. and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. volume 30, June 2019.

[137] Declan M. Mages, William M. Owen, Joseph E. Riedel, and Shyam Bhaskaran. The benefits of subsamping optical navigation images as applied to the new horizons flyby of (486958) 2014 mu69. In AAS, editor, 2nd RPI Space imaging Workshop. Saratoga Springs, NY., volume 2. RPI, October 2019.

[138] G. M. Maggiora, D. W. Elrod, and R. G. Trenary. Computational neural networks as model-free mapping devices. Journal of Chemical Information and Computer Sciences, 32(6):732–741, 1992.

[139] Pierre Magnan. Detection of visible photons in ccd and cmos: A comparative view. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 504(1):199 – 212, 2003. Proceedings of the 3rd International Conference on New Developments in Photodetection.

[140] M. Cols Margenet, P. Kenneally, and H. Schaub. Software simulator for heterogeneous spacecraft and mission components. In AAS Guidance and Control Conference, Breckenridge, CO, February 2–7 2018.

[141] F. Landis Markley and Yang Cheng. Wahba's problem with one dominant observation. Journal of Guidance, Control, and Dynamics, 41(10), Oct 2018.

[142] F. Landis Markley and Yang Cheng. Wahba's problem with one dominant observation. Journal of Guidance, Control, and Dynamics, 41(10):2318–2323, 2019/02/11 2018.

[143] A. Miguel San Martin, David S. Bayard, Dylan T. Conway, Milan Mandic, and Erik S. Bailey. A minimal state augmentation algorithm for vision-based navigation without using mapped landmarks. In GNC 2017: 10th International ESA Conference on GNC Systems, volume 10, Salzburg, Austria, May 2017.

[144] Iain M. Martin, Martin N. Dunstan, and Manuel Sanchez Gestido. Planetary surface image generation for testing future space missions with pangu. In 2nd RPI Space Imaging Workshop, Saratoga Springs, NY, 28-30 October 2019. AAS.

[145] Tomas J. Martin-Mur, Douglas S. Abraham, David Berry, Shyam Bhaskaran, Robert J. Cesarone, and Lincoln Wood. The jpl roadmap for deep space navigation. In 2006 AAS/AIAA SpaceFlight Mechanics Meeting, Tampa, Florida. Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2006.

[146] Nikos Mastrodemos, Daniel G. Kubitschek, and Stephen P. Synnott. Autonomous navigation for the deep impact mission encounter with comet tempel 1. Space Science Reviews, 117(1):95–121, Mar 2005.

[147] James S. McCabe and Kyle J. DeMars. Anonymous feature-based terrain relative navigation. Journal of Guidance, Control, and Dynamics, pages 1–12, 2020/01/08 2019.

[148] Travis H. Mercker and Maruthi R. Akella. Rigid-body attitude tracking with vector measurements and unknown gyro bias. Journal of Guidance, Control, and Dynamics, 34(5), September-October 2011 2011.

[149] John M. Mern, Kyle D. Julian, Rachael E. Tompa, and Mykel J. Kochenderfer. Visual Depth Mapping from Monocular Images using Recurrent Convolutional Neural Networks.

[150] F. Milletari, N. Navab, and S. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In 2016 Fourth International Conference on 3D Vision (3DV), pages 565–571, Oct 2016.

[151] T. Miso, T. Hashimoto, and K. Ninomiya. Optical guidance for autonomous landing of spacecraft. IEEE Transactions on Aerospace and Electronic Systems, 35(2):459–473, April 1999.

[152] Daniele Mortari, Christopher N. D'Souza, and Renato Zanetti. Image processing of illuminated ellipsoid. Journal of Spacecraft and Rockets, 53(3):448–456, 2016.

[153] Toshikazu Motoda and Yoshikazu Miyazawa. Identification of influential uncertainties in monte carlo analysis. Journal of Spacecraft and Rockets, 39(4):615–623, 2002.

[154] R. Mumtaz and P. Palmer. Attitude determination by exploiting geometric distortions in stereo images of dmc camera. IEEE Transactions on Aerospace and Electronic Systems, 49(3):1601–1625, July 2013.

[155] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. IEEE Transactions on Robotics, 31(5):1147 – 1163, Oct 2015.

[156] Raul Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. CoRR, abs/1610.06475, 2016.

[157] Tam Nguyen, Kerri Cahoy, and Anne Marinan. Attitude determination for small satellites with infrared earth horizon sensors. Journal of Spacecraft and Rockets, 55(6):1466–1475, 2019/01/02 2018.

[158] Tam Nguyen, Kerri Cahoy, and Anne Marinan. Attitude determination for small satellites with infrared earth horizon sensors. Journal of Spacecraft and Rockets, 55(6):1466–1475, 2018.

[159] Nvidia. 8-bit inference with tensorrt, May 2017.

[160] Stephen O'Keefe. Autonomous Sun-Direction Estimation Using Partially Underdetermined Coarse Sun Sensor Configurations. PhD thesis, University of Colorado, Boulder, May 2015.

[161] Stephen A. O'Keefe and Hanspeter Schaub. Sun heading estimation using underdetermined set of coarse sun sensors. In AAS/AIAA Astrodynamics Specialists Conference, Hilton Head, SC, Aug. 11–15 2013. Paper No. AAS-13-891.

[162] Stephen A. O'Keefe and Hanspeter Schaub. Gyro accuracy and failure sensitivity of underdetermined coarse sun heading estimation. In AAS/AIAA Space Flight Mechanics Meeting, Williamsburg, VA, Jan. 11–15 2015. Paper AAS 15-344.

[163] Stephen A. O'Keefe and Hanspeter Schaub. On-orbit coarse sun sensor calibration sensitivity to sensor and model error. In AAS/AIAA Space Flight Mechanics Meeting, Williamsburg, VA, Jan. 11–15 2015. Paper AAS 15-392.

[164] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. CoRR, abs/1511.08458, 2015.

[165] Simon Van Overeem and Hanspeter Schaub. Small satellite formation flying application using the basilisk astrodynamics software architecture. In International Workshop on Satellite Constellations and Formation Flying, University of Strathclyde, Glasgow, Scotland, July 17–19 2019. IWSCFF 19-88.

[166] W. Owen. Methods of optical navigation. 2011.

[167] Wiliam M. Owen. Optical navigation program mathematical models. Engineering Memorandum 314-513, Jet Propulsion Laboratory, August 1991.

[168] William M. Owen, Philip J. Dumont, and Coralie D. Jackman. Optical navigation preparations for new horizons pluto flyby. Technical report, Jet Propulsion Laboratory, California Institute of Technology, 2015.

[169] Paolo Panicucci, Cody Allard, and Hanspeter Schaub. Spacecraft dynamics employing a general multi-tank and multi-thruster mass depletion formulation. Journal of Astronautical Sciences, 2018. (in press).

[170] Woosang Park and Youeyun Jung. Robust crater triangle matching algorithm for planetary landing navigation. Engineering Note 10.2514/1.G003400, Jounral of Guidance, Control, and Dynamics, Korea Advanced Institute of Science and Technology, 2018.

[171] Tomislav Petkovic and Sven Loncaric. An extension to hough transform based on grandient orientation. In Proceedings of the Croatian Computer Vision Workshop, September 2015.

[172] Christopher M. Pong and David W. Miller. Reduced-attitude boresight guidance and control on spacecraft for pointing, tracking, and searching. Journal of Guidance, Control, and Dynamics, 38(6):1027–1035, 2015.

[173] M. Psiaki. Autonomous lunar orbit determination using star occultation measurements. Guidance, Navigation and Control Conference and Exhibit, Hilton Head, SC, August 2007. AIAA.

[174] Wojtek Pych. A fast algorithm for cosmic-ray removal from single images. Publications of the Astronomical Society of the Pacific, 116(816):148–153, feb 2004.

[175] Morgan Quigley. Ros: an open-source robot operating system. Technical report, Computer Science Department, Stanford University, Stanford, CA, 2007.

[176] P. RAO and M. WOESTE. Monte Carlo analysis of satellite debris footprint dispersion.

[177] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. CoRR, abs/1506.02640, 2015.

[178] Joseph E. Riedel, Andrew T. Vaughan, and Robert A. Werner. Optical navigation plan and strategy for the lunar lander altair; opnav for lunar and other crewed and robotic exploration applications. In AIAA Guidance, Navigation, and Control Conference, number AIAA 2010-7719, Toronto, Ontario, Canada, 2010.

[179] Ilaria Bloise Roberto Furfaro. Deep Learning for Autonomous Lunar Landing. Proceedings of the 2018 AAS/AIAA Astrodynamics Specialist Conference, Snowbird UT, 2018.

[180] A.M. Sabatini. Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation. Sensors, 11(10):9182–9206, October 2011.

[181] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. Analytical Chemistry, 36(8):1627–1639, 07 1964.

[182] Hanspeter Schaub and John L. Junkins. Stereographic orientation parameters for attitude dynamics: A generalization of the rodrigues parameters. Journal of the Astronautical Sciences, 44(1):1–19, 1996.

[183] Hanspeter Schaub and John L. Junkins. Analytical Mechanics of Space Systems. American Institute of Aeronautics and Astronautics, 3rd edition, 2014.

[184] Hanspeter Schaub and John L. Junkins. Analytical Mechanics of Space Systems. AIAA Education Series, Reston, VA, 4th edition, 2018.

[185] D.J. Scheeres, C.M. Hartzell, P. Sánchez, and M. Swift. Scaling forces to asteroid surfaces: The role of cohesion. Icarus, 210(2):968 – 984, 2010.

[186] Yanping Guo; Wayne Schlei. New horizons 2014mu69 flyby design and operation. In AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI, January 13–17 2019. Paper No. AAS-19-334.

[187] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017.

[188] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. CoRR, abs/1204.3968, 2012.

[189] Shai Shalev-Shwartz and Shai Ben-David. Understanding Machine Learning. Cambridge University Press, first edition, May 2014.

[190] Suneel I. Sheikh, Darryll J. Pines, Paul S. Ray, Kent S. Wood, Michael N. Lovellette, and Michael T. Wolff. Spacecraft navigation using x-ray pulsars. Journal of Guidance, Control, and Dynamics, 29(1):49–63, 2006.

[191] Y. Shkuratov, V. Kaydash, V. Korokhin, Y. Velikodsky, N. Opanasenko, and G. Videen. Optical measurements of the moon as a tool to study its surface. Planetary and Space Science, 59(13):1326 – 1371, 2011. Exploring Phobos.

[192] David Shteinman, Thibaud Teil, Scott Dorrington, Jaslene Lin, Hanspeter Schaub, John Carrico, and Lisa Policastri. Statistical approaches to increase efficiency of largescale monte-carlo simulations. In AAS/AIAA Astrodynamics Specialist Conference, At Snowbird, UT, USA, 08 2018.

[193] M. D. Shuster and S. D. Oh. Three-axis attitude determination from vector observations. Journal of Guidance, Control, and Dynamics, 4(1):70–77, 2019/02/11 1981.

[194] R. M. Silva, T. R. Jorris, and E. M. III Vallerie. Experiment d009: Simple navigation. Technical report, NASA; United States, 1971.

[195] Joseph A. Starek, Behçet Açıkmeşe, Issa A. Nesnas, and Marco Pavone. Spacecraft Autonomy Challenges for Next-Generation Space Missions, pages 1–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[196] G. P. Stein. Lens distortion calibration using point correspondences. In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 602–608, June 1997.

[197] Filip Suligoj, Bojan Sekoranja, Marko Svaco, and Bojan Jerbic. Object tracking with a multiagent robot system and a stereo vision camera. Procedia Engineering, 69:968 – 973, 2014. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013.

[198] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, Advances in Neural Information Processing Systems 21, pages 1601–1608. Curran Associates, Inc., 2009.

[199] Daniel Svozil, Vladimír Kvasnicka, and Jirí Pospichal. Introduction to multi-layer feed-forward neural networks. Chemometrics and Intelligent Laboratory Systems, 39(1):43 – 62, 1997.

[200] Sean S. M. Swei, Jesse C. Fusco, and Robert H. Nakamura. Design of sun-safe controllers for lunar atmosphere and dust environment explorer. Journal of Guidance, Control, and Dynamics, 39(9):2022–2033, 2017/12/19 2016.

[201] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A Survey on Deep Transfer Learning: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4, 2018, Proceedings, Part III, pages 270–279. 10 2018.

[202] B Tapley, Bob E. Schutz, and George H. Born. Statistical Orbit Determination, chapter 4 - Fundamentals of Orbit Determination, pages 159–284. Elsevier Academic Press, 12 2004.

[203] M. Tegmark. An icosahedron-based method for pixelizing the celestial sphere. volume 470, pages L81–L84. The Astrophysical Journal Letters, 1996.

[204] Thibaud Teil and Hanspeter Schaub. Software architecture for closed-loop autonomous optical navigation scenarios. In 1st Annual RPI Workshop on Image-Based Modeling and Navigation for Space Applications, Troy, NY, June 4–5 2018.

[205] Thibaud Teil, Hanspeter Schaub, and Scott Piggott. Comparing coarse sun sensor based sequential sun heading filters. In AAS Guidance and Control Conference, Breckenridge, CO, Feb. 1–7 2018. Paper AAS 18-011.

[206] Jurgen Telaar. Gnc architecture for the e.deorbit mission. Number DOI: 10.13009/EUCASS2017-317, Keplerlaan 1, 2201 AZ Noordwijk, Netherlands, 1998.

[207] Divya Thakur and Maruthi R. Akella. Gyro-free rigid-body attitude stabilization using only vector measurements. Journal of Guidance, Control, and Dynamics, 38(4):811–818, 2019/02/11 2014.

[208] Tom Tsao and Richard Chiang. Gyroless Transfer Orbit Sun Acquisition Using Only Wing Current Feedback. American Institute of Aeronautics and Astronautics, 2017/12/19 2009.

[209] Yuichi Tsuda, Makoto Yoshikawa, Takanao Saiki, Satoru Nakazawa, and Sei ichiro Watanabe. Acta Astronautica, 156:387 – 393, 2019.

[210] R. van der Merwe. The square-root unscented kalman filter for state and parameter-estimation. Acoustics, Speech, and Signal Processing, 2001.

[211] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res., 11:3371–3408, December 2010.

[212] G. Wahba. A least squares estimate of satellite attitude. SIAM Review, 7(3):409–409, 1965.

[213] Thomas Weismuller, David Caballero, and Manny Leinz. Technology for Autonomous Optical Planetary Navigation and Precision Landing.

[214] Walton Williamson, Jason Speyer, Vu Dang, and James Sharp. Fault Detection for Deep Space Satellites.

[215] Walton R. Williamson, Jason L. Speyer, Vu T. Dang, and James Sharp. Fault detection and isolation for deep space satellites. Journal of Guidance, Control, and Dynamics, 32(5):1570–1584, 2009.

[216] R. A. Windhorst, B. E. Franklin, and L. W. Neuschaefer. Removing cosmic-ray hits from multiorbit hst wide field camera images. Astronomical Society of the Pacific, 106(701):798–806, 1994.

[217] Luke B. Winternitz, Munther A. Hassouneh, Jason W. Mitchell, Samuel R. Price, Wayne H. Yu, Sean R. Semper, Paul S. Ray, Kent S. Wood, Zaven Arzoumanian, and Keith C. Gendreau. SEXTANT X-ray Pulsar Navigation Demonstration: Additional On-Orbit Results.

[218] David Wokes and Stephen Wokes. Surveying and Pose Estimation of a Lander Using Approximative Crater Modelling.

[219] J. Wood, M. Cols Margenet, P. Kenneally, H. Schaub, and S. Piggott. Flexible basilisk astrodynamics visualization software using the unity rendering engine. In AAS Guidance and Control Conference, Breckenridge, CO, February 2–7 2018.

[220] Jin Wu and Zebo Zhou. Fast linear quaternion attitude estimator using vector observations. In IEEE Transactions on Automation Science and Engineering, number 10.1109/TASE.2017.2699221. Institute of Electrical and Electronics Engineers, 2018.

[221] Sungpil Yang, Maruthi R. Akella, and Frédéric Mazenc. Immersion and invariance observers for gyro-free attitude control systems. Journal of Guidance, Control, and Dynamics, 39(11):2570–2577, 2019/02/11 2016.

[222] Qu Ying-Dong, Cui Cheng-Song, Chen San-Ben, and Li Jin-Quan. A fast subpixel edge detection method using sobel-zernike moments operator. Image and Vision Computing, 23(1):11 – 17, 2005.

[223] Pei Yingjun and Hou Xinwen. Learning representations in reinforcement learning:an information bottleneck approach, 2019.

[224] Hyungjoo Yoon and Panagiotis Tsiotras. Spacecraft Angular Velocity and Line-of-Sight Control Using A Single-Gimbal Variable-Speed Control Moment Gyro. American Institute of Aeronautics and Astronautics, 2019/02/11 2005.

[225] Zhengyou Zhang. Parameter estimation techniques: a tutorial with application to conic fitting. Image and Vision Computing, 15(1):59 – 76, 1997.

Appendix A

Neural Net Details

About appendices:

This enumerated list displays all layers of the implemented Convolutional Neural Net for radius and center finding. The syntax is borrowed from *Pytorch* [1], while the architecture is derived from the ResNet[94] paper.

(1) Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

(2) 3×ResNetBlock:

Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

Dropout(p=0.5, inplace=False)

(3) Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), padding_mode=same)

(4) 3×ResNetBlock:

Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

[1] https://pytorch.org/

BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

Dropout(p=0.5, inplace=False)

(5) Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), padding_mode=same)

(6) 3×ResNetBlock:

Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

Dropout(p=0.5, inplace=False)

(7) Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), padding_mode=same)

(8) 3×ResNetBlock:

Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

Dropout(p=0.5, inplace=False)

(9) Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), padding_mode=same)

(10) 3×ResNetBlock:

Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=same)

BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

Dropout(p=0.5, inplace=False)

(11) Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), padding_mode=same)

(12) Flatten

(13) Linear(in_features=131072, out_features=3, bias=True)