SAT-ANS: System Analysis Tool for Autonomous Navigation in Space

Integrated Pulsar, Angle, and Radial Velocity Measurements

A.J. Jongschaap 4510879

MSc Thesis
Arjen Jongschaap

TU Delft Delft University of Technology

Aerospace Engineering

November 18, 2018

SAT-ANS: System Analysis Tool for Autonomous Navigation in Space

Integrated Pulsar, Radial Velocity and Angle Measurements

by

A.J. Jongschaap

In partial fulfillment for

MSc. Aerospace Engineering

at Delft University of Technology

Supervisor: Prem Sundaramoorthy TU Delft/Space Systems Engineering

TU Delft Delft University of Technology

PREFACE

Grandia imbutus tentabis parvis tutis

Pulsar Navigation is a topic I was introduced to during a lecture at TU Delft. Having always liked to look up at the sky on a clear night, the idea that some of those points of light may have utility for space flight, immediately appealed to me. I would like to thank my supervisor, Prem Sundaramoorthy who agreed to let me design and undertake a thesis which included this topic. Having come to this work with minimal knowledge of navigation in general, I really appreciate the conversations and meetings with Róbert Fónod on the topics of filtering, and the difficulties of non-linearity.

Thank you to my family, particularly my Mother, Father, and sister. I have always had your full support in my endeavours and this would not have been possible without you. Thank you to my friends and especially Jacob Cole, whose patience and humour at times of stress and despair was much appreciated. Also to Mathijs van de Poel, whose additional comments on artistic aspects of this work were very much required - albeit to my dismay.

Finally, thank you to my old physics school master, who inspired me to pursue a path to the physical sciences and then on to space, and whom I hope would not have considered this work - unlike more than a few homework pieces - as *drivel*.

Front image of Crab Nebula and pulsar courtesy of NASA's Chandra: `http://chandra.harvard.edu/photo/2018/crab/`

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

AWGN	Additive white Gaussian Noise.
CAS	Cascaded Integration.
CD	Concurrent Design.
CEN	Centralised Integration.
COTS	Commercial Off the Shelf.
CRLB	Cramer-Rao Lower Bound.
Dec	Declination.
DSN	Deep space network.
EKF	Extended Kalman Filter.
EM	Electromagnetic.
FFI	Federated Filtered Integration.
FIM	Fisher Information Matrix.
GDOP	Global Dilution of Precision.
GEO	Geostationary Earth Orbit.
GNSS	Global navigation satellite system.
GPS	Global positioning system.
HYB	Hybrid Integration.
I/O	Input/Output.
ISM	Interstellar Medium.
ISS	International Space Station.
LEO	Low Earth Orbit.
MC	Monte Carlo.
MSP	Millisecond pulsar.
pdf	Probability density function.
PF	Particle Filter.
PNAV	Pulsar navigation.
RA	Right Ascension.
RK4	Runge-Kutta four.
RMSE	Root Mean Square Error.
RNAV	Radio-pulsar navigation.
s/c	Spacecraft.
SNR	Signal to Noise Ratio.
SSB	Solar system barycentre.
TDB	Temps Dynamique Barycentrique.
TOA	Time of Arrival.
UKF	Unscented Kalman Filter.
XNAV	X-ray-pulsar navigation.

1

INTRODUCTION

This document details the work undertaken over a seven month period towards fulfilling the degree Masters of Science in Aerospace Engineering.

This chapter gives brief overview of the problem motivated by research, the research aims and then a description of the method and outcomes of the development process of the software System Analysis Tool for Autonomous Navigation in Space (SAT-ANS) is divided along the chapters in this document.

1.1. AUTONOMOUS NAVIGATION

LEO and GEO have been conquered. The reasons for this level of access to near-Earth space are plummeting launch costs, near-Earth information need, and possible autonomy with respect to navigation. Navigation will be the focus of this research.

Currently for deep space missions, spacecraft navigation generally occurs by Earth-based Doppler radio interferometry. Although accurate, the navigation solution degrades as a function of distance from Earth and requires the operation of multiple large radio telescopes for the detection of the weak signals emitted by the spacecraft. Given the limited number of such systems (Deep Space Network (DSN) and European equivalent, ESTRACK/DSA [27]) and their resource-intensive operation, the number of spacecraft such systems can interact with are limited. A solution to this, is to allow the spacecraft to calculate its own position – autonomous navigation. Autonomous navigation is becoming increasingly important for space systems. By reducing the required user input, the number of operational spacecraft and missions can increase.

Within the bounds of availability (surface of the Earth to possibly the Moon [28]), spacecraft are able, if required, to navigate autonomously using GNSS methods - most famously, the GPS constellation. GNSS navigation works by the reception of coded signals defining the position of the satellite sender and the time of signal emission, which can determine the position and velocity of the observer. Beyond the scope of GNSS, the options for autonomous navigation become limited and the topic remains an active area for research. Methods have been developed which use the angular position of planetary bodies for positional information, gathering velocity information from the doppler shift due to radial motion from the sun [29]. Although the methods developed can be accurate, they are all dependent on the availability of the observables, which is orbit dependent. For true deep-space or interplanetary applications, there is one stand-out method for navigation: pulsar-based navigation. Although pulsar navigation will likely be at least part of the solution for autonomous space navigation, there are some technological issues to date which prevent its implementation. Due to the weak nature of the detected pulsar signals, long integration times, or very sensitive detectors are required. This can currently limit the performance of these systems – perhaps preventing the implementation of a fully autonomous navigation system today.

A possible solution to this is to add additional sensors to the navigation system and fuse this navigation information with the pulsar sensor output.

1.2. RESEARCH AIMS

Having established that pulsar navigation may be improved with additional sensors, there should be a tool to investigate this. Further, by removing the assumption that pulsar navigation will be the only form of autonomous navigation, then the tool should be capable of investigating different types of navigation configurations. Having found the tools currently available to be unsuitable or unavailable for this task, the following research aims were generated in line with time project time constraints:

Develop a general autonomous navigation simulator testbed for spacecraft, capable of quantifying positional navigation accuracy.

Identify the most likely candidates for autonomous navigation sensors and quantify the positional navigation accuracy of a user-defined subset of models of these.

Extend the tool to be capable of user-defined navigation filters and orbital conditions.

Further, as pulsar navigation is likely to be part of future autonomous navigation systems, an investigation using the tool is performed, and the aim generated:

Develop pulsar navigation sensor models and use the software tool to investigate the impact of fusing additional sensor information on positional navigation performance.

1.3. SYSTEMS ANALYSIS AND ARCHITECTURE

Requirements were generated in line with the research aims, scoped to concurrent design principles. From this, a modular software architecture was developed, consisting of four main components:

- Orbit module
- Sensor module
- Navigation module
- Analysis module

With inputs and outputs following the flow chart in figure 1.1

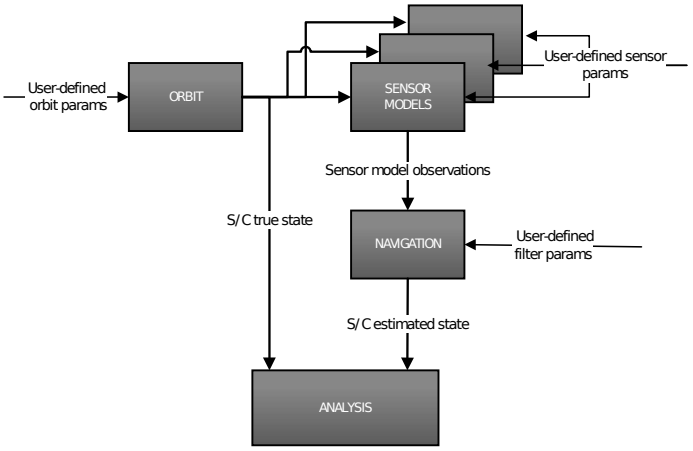


Figure 1.1: Overview flowchart of SAT-ANS

The general overview is as follows: The reference true s/c state is generated by the orbital module, which gives input to the sensor module. The sensor observations are based on this true state and the observables defined in a library. This is then given as input to the navigation module where a state estimate is generated. The analysis module then compares the estimated state and the true state to form an assessment of the navigation system.

1.4. SOFTWARE DEVELOPMENT

The modules were developed in Python. For the orbital model a Keplerian orbit has been implemented with two solvers: one analytical - Mean Motion; and one numerical which is capable of incorporating orbital perturbations. The numerical solver was validated against an analytical Kepler orbit.

Two sensor models were initially developed for the Sensor module: an angle sensor and a radial velocity sensor. It was assumed that the detection mechanism for the radial velocity sensor was based on the doppler shifts in emission lines of celestial bodies. Additionally, a library architecture for the observables was generated and implemented for the two sensors. The sensor models were verified analytically.

An unscented Kalman filter was implemented as a first filter for the Navigation module. This is based on the van der Merwe formulation of the filter. The module and filter were made capable of taking asynchronous updates from the sensors. The filter was verified statistically using a Re-Entry problem.

The Analysis module implemented a Monte Carlo analysis functionality to the system and allowed the quality of the navigation system to be analysed. Further, an initial attempt at the implementation of the posterior Cramer-Rao Lower Bound has been undertaken. The software modules were then integrated and a Low Earth Orbit test was made.

1.5. PULSAR NAVIGATION

Having tested the integrated SAT-ANS, pulsar navigation was then assessed for its best initial implementation. A low fidelity model considering only the arrival times of the pulsar pulses was chosen with the delta-correction based navigation algorithm. Two pulsar sensor models were added to the sensor module: a radio and an x-ray pulsar sensor with respective libraries of pulsars containing observational parameters. The detection equations were verified

against literature. Further, a third order clock model was added to the navigation module, to assess the impact of timing errors on pulsar navigation.

1.6. RESULTS

Two test cases were established to investigate the use of additional sensors on navigation performance: a planetary orbit, in this case LEO; and a deep space, interplanetary orbit - with the same characteristics as the Earth's orbit around the Sun. These cases were tested for x-ray and radio pulsar navigation sensors alone, and then integrated with radial velocity and angle sensors respectively. Furthermore, the addition of clock noise was tested in the cases mentioned. It was found that for the filter and sizing parameters chosen that the x-ray pulsar navigation sensor performed better in general, however the velocity estimation was very sensitive to the addition of clock noise. The general conclusion is that there is a lower limit to the quality of the sensor in different situations for the additional sensors to provide useful information.

2

BACKGROUND AND LITERATURE REVIEW

This work was motivated out of the requirement for autonomous navigation systems, and the current technological limitations of pulsar navigation. This section provides some of the background for the motivation of this work. Background based on [17].

2.1. NAVIGATION IN SPACE

To assess the need for autonomous (positional) navigation, the current different methods of navigation in space are investigated. These methods may broadly be split into GNSS, Earth-based radiometric tracking and sensor-based navigation.

2.1.1. GNSS

Autonomous navigation is generally possible on the surface of the Earth and up to an altitude of twenty thousand kilometers [30] though GNSS. Constellations of satellites orbiting in medium Earth orbit emit time-stamped, coded signals which enable the determination of position and velocity to meter- and millimeter-level accuracy respectively [31]. For s/c operating in LEO, this is an ideal option for autonomous navigation

2.1.2. RADIOMETRIC POSITIONING

Having established that autonomous navigation is in principle possible on Earth in in the vicinity of the GNSS satellites, the attention now turns to deep space. Currently there is one main method for navigation in deep space - radiometric tracking. This uses three radio antenna arrays to position the s/c. An example of this is the Deep Space Network (DSN), with antennas in Australia, Spain and the USA. This method can also be used to communicate with the s/c if required. Radiometric tracking of s/c measures both the range and the range-rate, through the round-trip time of a signal sent from the Earth and the Doppler shift in frequency of that signal (based on a reference frequency from the up-link carrier signal)[32]. This method is known as two-way radiometric tracking, as an up-link is required to generate the reference frequency, so that the range rate may be calculated.

The DSN is currently able to provide meter-level range accuracy and a Doppler accuracy of sub-millimeter-level in the Ka radio band. It should however be noted that the ranging accuracy decreases and the required observation time increases as a function of distance from Earth. The differential one-way ranging accuracy is around 2.5 nm which is equivalent to around 400 m at a distance of 1 AU from Earth [33]. In addition to the position solution degradation as a function of distance, a further issue exists with radiometric tracking - the use of the telescopes. Although capable of accurately tracking s/c, there are a limited number of telescopes able to track spacecraft deep in the solar system. This poses a resource constraint on the use of this system for navigation. As the number of interplanetary and deep space s/c increases, so will the demand on these systems. A solution is to allow the s/c to navigate autonomously, with minimal input from the ground.

2.1.3. SENSOR-BASED NAVIGATION AND DEEP-SPACE-1

Sensor-based navigation will be defined as navigation through the use of on-board sensors, such as radial velocity and angle sensors. Due to them being onboard the s/c, it allows for autonomous navigation. The Deep-Space-1 mission was the first spacecraft to demonstrate autonomous deep space navigation. Using a system called AutoNav, the spacecraft used an optical payload observing sun-lit asteroids as beacons to triangulate itself [9]. The mission validated its requirement of 250 km positional accuracy and may be a solution to autonomous navigation, however local bodies should be visible and their positions known - limiting the scope to the solar system and the known positions of the observables - whose characteristics should be distinct such that positioning is possible in a 'cold-start', depending on instrument accuracy.

will be defined as the s/c is lost in space, and has no positional estimation

2.2. PULSAR NAVIGATION (PNAV)

In this part, pulsars will be introduced and the characteristics motivating their use as navigation beacons will be described, based on [17].

2.2.1. PULSARS

As a fast rotating stellar remnant, a pulsar is the result of a supernova. Due to their generally small radius, they contain magnetic fields which are some of the strongest seen in the universe [19]. As the pulsar rotates, and if the magnetic dipole axis is not aligned to the rotation axis, a beam of radiation will sweep a path, similar to a lighthouse 2.1.

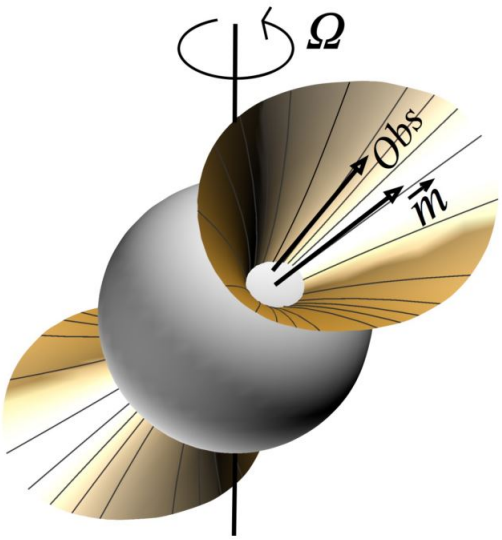


Figure 2.1: Pulsar (radio) EM emission. Ω represents the axis of rotation, $\vec{m}$ is the magnetic dipole axis, and *Obs* is the observer's direction [1]

PULSAR SIGNALS

Pulsars are known for their stable signal and are generally grouped based on the stability of this signal (linked to the emission mechanism) Figure 2.2 shows the rate of change of period of known pulsars as a function of their period. There are two main groups: those broadly above the millisecond range, and those below. Those above are usually more energetic and therefore brighter, but less stable and more prone to errors such as glitching. On the other hand, those generally below the millisecond, known as millisecond pulsars (MSPs) have a much more stable signal but tend to be much dimmer [1].

2.2.2. PROBLEMS WITH PNAV

Navigation with pulsars uses the detection of their pulses, known as Time of Arrival (TOA) based navigation. The pulse is detected and compared against for example a database of known arrival times. These known arrival times may be either an extrapolation of observations of the pulsar or generated by a model of the pulsar. However, the signal emitted from the pulsar is not perfect, and there are issues [31] which should be addressed to use the pulsars as navigation beacons.

PULSAR IRREGULARITIES

As the rotation of the pulsars is due to natural phenomena, the detected signal may not be completely regular. An example of an irregularity is glitching - where the detected signal from the pulsar suddenly changes [34]. There is however an anti-correlation between the age of the pulsar and its tendency to show irregularities (younger, more energetic pulsars are likely to be more unstable). These sort of effects are difficult to model, however. As such, it may be required to observe navigation pulsars regularly to update any possible changes. It should be reiterated that these sorts of effects are rare in MSPs - which are the prime candidates for pulsar navigation.

CLOCK ERRORS

Another source of error is due to timing. The detected pulsar TOAs are compared to a reference TOA, and if the detected TOA is marred by some timing error, this will cause an error in the navigation solution [35]. A solution to this

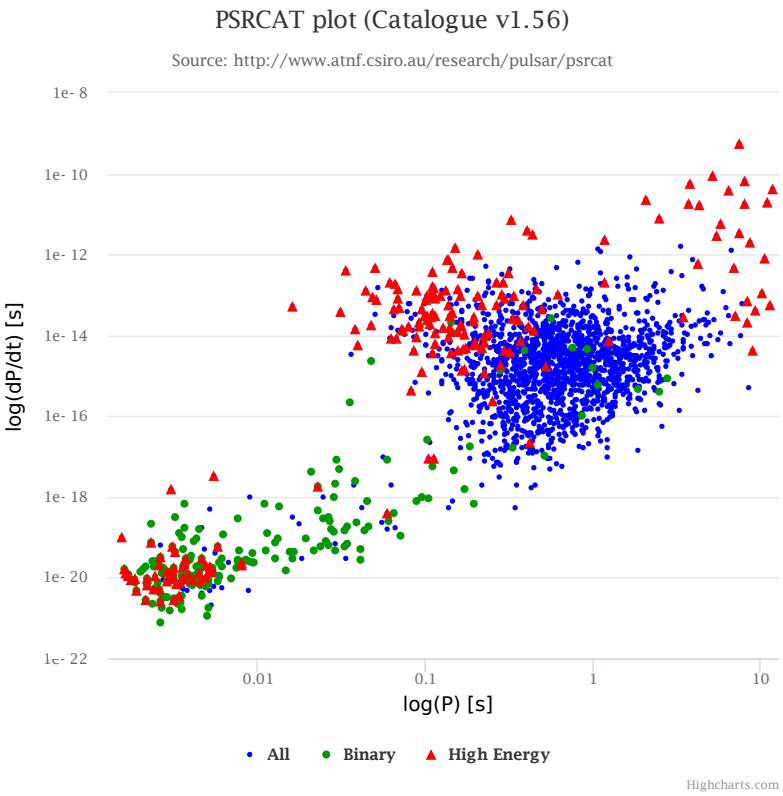


Figure 2.2: Pulsar P-Pdot diagram of all pulsars found in [2]. Blue represents all standard radio emission pulsars, green a binary system containing a pulsar, and red represents those pulsars which emit at higher energies, including x-rays.

is to either observe additional pulsars, depending on the navigation method being used, or to perform differencing (single or double), which can remove clock errors.

AMBIGUITY PROBLEM

There are two main methods used for pulsar navigation which are explained in more detail later in this document: Delta correction and absolute navigation [19]. Delta correction allows for the sequential observation of pulsars, but absolute navigation, as the name suggests, is able to define the observer's position in one observation. This means that 3-4 pulsars must be observed simultaneously. This observation of the pulsar signals defines a solution which is the user's position. However, due to the periodic nature of the detected signals, there is an integer ambiguity in the number of pulses in the detection. It was found that solving the ambiguity problem (for positions known to pulsar-defined accuracies), is computationally intensive, with an iterative search space generally used. The greater the knowledge of the observer's position, the smaller the required search space [36].

The search space for the unique solution is dependent on the knowledge of the current position. The question arises on if it is possible to use pulsars in a cold start situation, with no knowledge of the current position. This was investigated [37] and it was found that although technically possible to navigate with no a-priori position information, there is a likelyhood of an incorrect solution being chosen. However, additional methods were suggested such as Bayesian methods which work to constrain the ambiguity number, improving the performance in a cold start configuration.

PULSAR PROFILE AT DIFFERENT FREQUENCIES

It has been found that the pulsar detection profile can change depending on the observation frequency. The two main detection bands for pulsars are radio and x-ray. As much more research and observation has been done in the radio band (as the Earth's atmosphere is transparent in the radio band), there is less observational data on this in the x-ray band, although this is starting to change [31]. The main mitigation to this, is to increase the observation in different bands of the chosen navigation pulsars.

a single difference in observation equations is the difference between predicted phase at a known location and the the observed phase at the current location [17]

initial position error must be less than $cP_{fast}/2$, where P_{fast} is the lowest period of the observed pulsar[17]

SIMULTANEOUS OBSERVATION OF MULTIPLE PULSARS

If the simultaneous observation of sufficient pulsars to define the state is not possible and long integration times are assumed, then the s/c may move a large distance over the pulsar observation period. Further, the errors in the dimensions perhaps not covered, or covered to a lesser extent by the observations of the pulsars.

Additionally, as mentioned before, there are many pulsars which may be observed for navigation purposes. If the observer is not in a deep space environment, where all pulsars are assumed to be observable, then specific pulsars may not be visible due to occultations. This creates the opportunity for the optimisation of the specific pulsars based on the available observation time, the specific pulsars available and the (estimated) quality of the navigation solution that these specific pulsars will provide. The main pulsars parameters affecting the navigation solution are the timing and the brightness. From this a possible ranked list pulsars may be generated for specific dimensions defining the optimum observation or series of observations [39].

2.3. AUGMENTED NAVIGATION

A solution to the problems shown by PNAV is to augment the navigation system with additional sensors, not or less affected by the issues mentioned above. Research has been done on combining PNAV sensors with additional sensors, such as x-ray pulsar sensors and an angular radius sensor [40] and the combination of pulsar measurement and radial velocity from the sun using doppler means [41]. Both found performance improvements with additional sensors. However, they were constrained to specific orbital cases and so a general comment on the improvement of PNAV with additional sensors could not be made. Further the impact of aspects like clock noise had not been considered in these cases.

CURRENT STATE

There are possible performance gains to be had by integrating different sensors for specific navigation applications. However to date, the assessment of integrated navigation for space applications has been specific to certain applications - either trajectories, combination of sensors or a specific filter [[42], [43], [44]]. Different sensors have different observables, such as stellar object spectra for spectrometers, and celestial bodies for optical sensors. Their suitability is therefore dependent on the availability of those observables. Future space missions, from those which are complex - consisting of many different navigation requirements at different mission phases such as the Orion spacecraft [45], to those operating in unusual trajectory conditions, will require autonomous navigation which is optimised for their situation. Having a system in place during the design phase of a mission where different combinations of sensors can be simulated under different trajectory conditions for navigation purposes could fulfill this need. Further, as pulsar navigation progresses, a system where the technological advances may be simulated, could additionally benefit the design of spacecraft.

2.3.1. S/C NAVIGATION SYSTEM DESIGN

The typical project life cycle (as defined in ECSS-M-ST-40C [46]) with requirements definition and validation can be seen in figure 2.3

Although the international communication standard defines deep space be be any distance greater than $2\times10^6 km$ [38], in addition to that, it will be assumed that deep space uses the Sun as the main orbital body (i.e is not within the sphere of influence of a local massive body) and so synonymous with interplanetary space

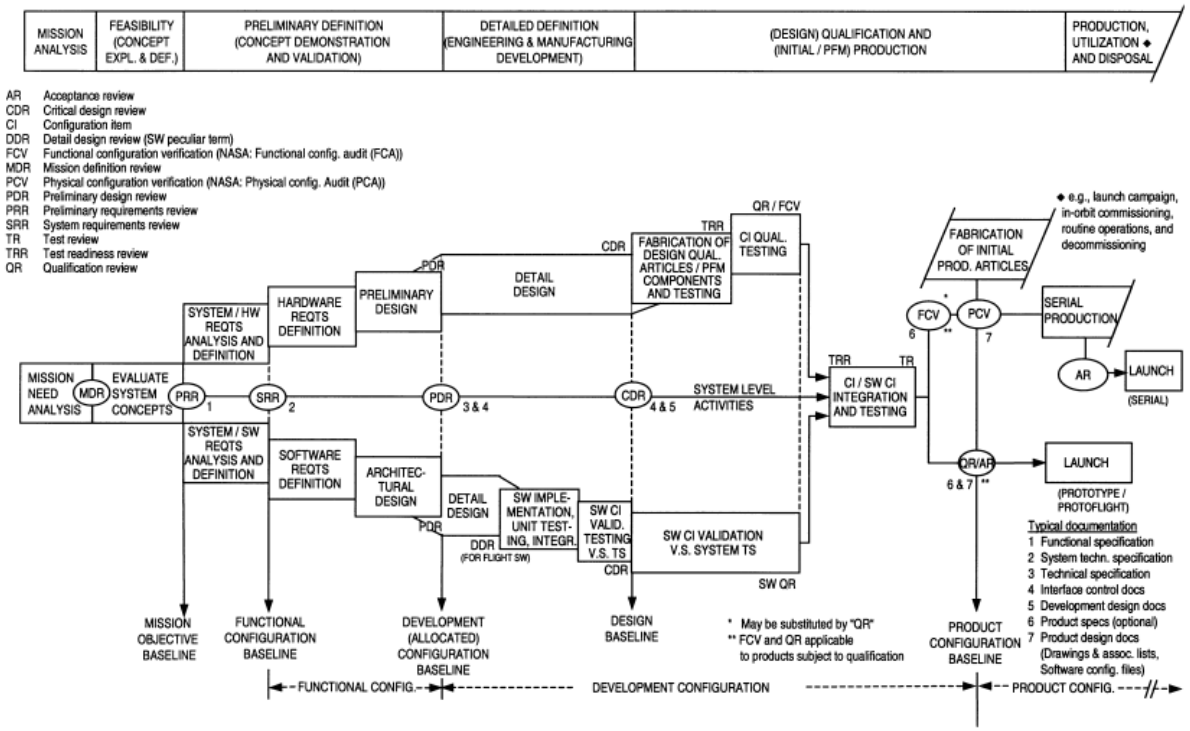


Figure 2.3: Project phases as defined in ECSS-M-ST-40C

As this work considers all autonomous navigation systems, it will be used in the evaluation of system concepts and perhaps the preliminary design phase. The detailed design phase will require in-depth simulation, with dedicated tools.

Within the preliminary design, there are different processes which may be applied. These may broadly be split into two categories: concurrent and serial. Much research has been done in recent times on the use of concurrent design in preliminary phases such that centres for its implementation have been installed in ESA's ESTEC [4] and are often used for preliminary mission design [47]. As concurrent design has shown promise for s/c preliminary design, over serial design in many cases, it will be the only concept considered from here on.

The principle of concurrent design is a consensus-based, parallelised approach to system design, where the customer is kept in the loop, such that their requirements and expectations best reflected in the final system. Practically, this means generating a system where the interaction between the different spacecraft subsystem and sizing aspects (such as cost and risk etc) may be assessed in parallel. This has been done by placing respective subsystem experts in close proximity such that they are able to communicate effectively and generate a mission concept [4]. To facilitate this, software tools have been generated to aid the generation of the different subsystems, which are able to take user requirements and constraints as input.

Having motivated the need for autonomous navigation and that it is likely to become a central part of future post-Earth spacecraft navigation systems, if concurrent engineering is to be used in the future, a tool capable of analysing and sizing these navigation systems for use in concurrent design will be required.

CONCURRENT DESIGN MODELLING

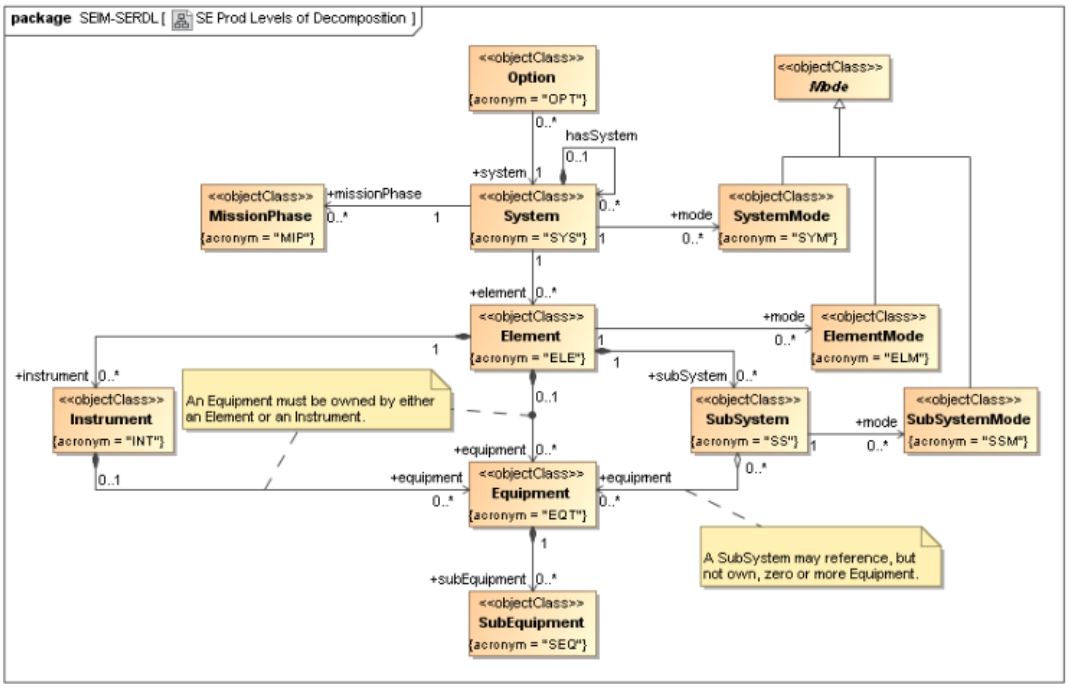


Figure 2.4: Decomposition of the concurrent space engineering information model [3]

Figure 2.4 shows a decomposition of the of the different modules within the CD process. It is defined quite generally, but the basic flow may be considered to be the s/c as a system, with some defining parameter - an element whose value is controlled by some sensor input and/or a subsystem. The modes, equipment and subequipment, although important to the definition and functionality of the s/c will be omitted from the work in this thesis.

The desired input/output of concurrent design may be defined based on figure 2.5.

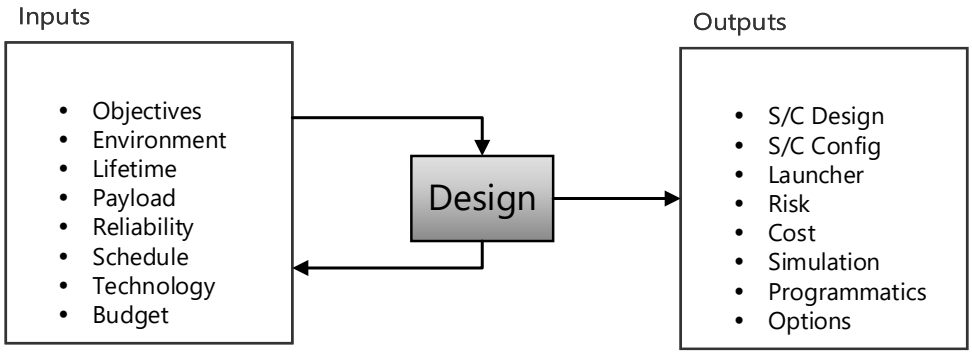


Figure 2.5: Concurrent design inputs and outputs [4]

From a navigation perspective, the scope of the tool will be defined in the next chapter.

CURRENT SOFTWARE TOOLS

Having come to the conclusion that a tool to design and assess the quality of integrated navigation system for deep space s/c which could be used in a concurrent design approach would be useful, the current state of the art in this respect is investigated. Three tools could be found which have the capability for the functionality defined above:

gncde An industry specific navigation system analysis tool by GMV. Not for general use [48].

Athena: Astrodynamics toolbox for high-fidelity error and navigation analysis This is MATLAB a toolkit for the generation and analysis of navigation systems and orbits. It is specialised for navigation in close proximity operations and docking [49]. Although this software may be useful for this work's applications - recent reference to its use could not be found.

Mission Analysis, Operations, and Navigation Toolkit Environment This is a JPL-created library in python which seems capable of the fulfilling the need mentioned above, however a licence could not be obtained and a specific application would still require building [50].

2.3.2. RESEARCH AIMS

With the possible exception of gncde, there is no dedicated application for the definition of an autonomous navigation system and to be able to determine its effectiveness in different orbital scenarios. Generating such a general tool would be a mammoth task, and far too much for a single masters thesis. Although relevant, a more constrained project must be generated. This leads to the following research aims:

Develop a general autonomous navigation simulator testbed for spacecraft, capable of quantifying positional navigation accuracy.

The main constraint here is the area of navigation which will be covered - positional navigation. This means that aspects such as attitude will be omitted from the project.

Identify the most likely candidates for autonomous navigation sensors and quantify the positional navigation accuracy of a user-defined subset of models of these.

As there are many sensors which could in principle be used for navigation and possibly different methods of navigation using these sensors, the implemented sensors should be the ones most likely to be integrated.

Extend the tool to be capable of user-defined navigation filters and orbital conditions.

This adds some generality to the tool - once the sensors has been defined, combinations may be tested in various orbital scenarios and using different navigation filters.

Develop pulsar navigation sensor models and use the software tool to investigate the impact of fusing additional sensor information on positional navigation performance.

Finally, as pulsar navigation is likely to be part of future autonomous navigation systems, an investigation using the tool is performed.

3

SYSTEM ANALYSIS AND DEFINITION

In this section the architecture of the test environment will be defined. This is a software-based project which is modelling space systems, and will therefore try to adhere to ECSS-E-ST-40C (hereby referred to as ECSS) space engineering software standards [51].It will also include best practises where possible and deemed reasonable with respect to complexity and available time.

First, the requirements of the system are generated, then based on these, the software architecture is generated. From here, the different modules within the software are motivated and designed with theory where relevant. Finally, a verification and validation approach is generated.

3.1. REQUIREMENTS ANALYSIS

To define the architecture of the test-bed, the requirements of the system must first be defined. According to ECSS section 5.2.2.1 the following process outlines the generation of requirements:

The customer shall derive system requirements allocated to software from an analysis of the specific intended use of the system, and from the results of the safety and dependability analysis.

As a safety and dependability analysis are not relevant to this work, the intended use of the system will be defined.

3.1.1. INTENDED USE OF THE SYSTEM

Based on the outcome of the previous chapter and the research aim, the intended use of the system may be detailed.

For a (concurrent preliminary) design, the system will assess the performance of a chosen navigation system within the framework of a mission scenario. For this the general input and output of the tool must be defined in accordance with figure 2.5.

TOP LEVEL INPUT/OUTPUT

The tool being generated is to aid concurrent design, not to develop it. The scope of the Input/Output (I/O) may therefore be limited somewhat for the purposes of this work. To that end, only **Environment** and **Technology** will be considered as input for the first iteration of the tool, and **Simulation** will be considered as the only output. The rest of the parameters may be added in later iterations when considering the tool for actual use within a concurrent design framework or else ported from some other tool. The generation of the tool such that it may be expanded into a useful addition for concurrent design is considered acceptable for this version.

USE CASE

To generate the system requirements for the tool, an example use case is generated:

The user will define a trajectory or an orbit around a massive body within the solar system. From this, the user will generate their navigation system by selecting sensors and a filter, specifying relevant sizing and noise parameters. The software will then determine the accuracy of that navigation system over the trajectory. From this, based on user navigation requirements, the software will then optimise the chosen navigation system via sizing parameters to meet these requirements.

3.1.2. FUNCTIONAL FLOW

From the intended use of the system in addition to the need case established in the previous section, a prospective functional flow diagram is developed and shown in figure 3.1. As the figure shows, there are two main functions of the software - modelling the navigation system (F.01), and sizing the system (F.02). The sizing aspect is considered secondary to the generation and modelling of the navigation system. F.02 is therefore left at the high level. F.01 is

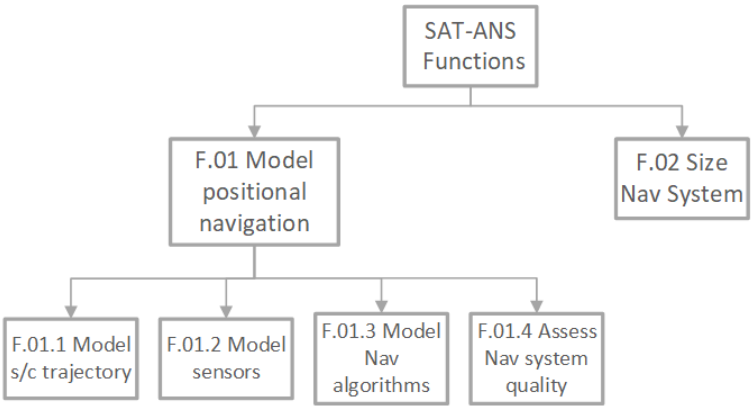


Figure 3.1: Functional flow down of the software model

decomposed and consists of modelling the s/c trajectory, the sensors, navigation algorithms and finally assessing the navigation system quality.

For the I/O as defined above, **Environment** will be defined as the all aspects external to the s/c (such as the orbital surroundings and input to the sensors) and **Technology** will define all aspects internal to the s/c (such as the filter and the sensors). **Simulation** will define the results of modelling the system. In this way, the system will be designed in accordance with concurrent design tools.

3.1.3. REQUIREMENTS

Based on previous section and motivations, the requirements for the software tool are derived.

TOP LEVEL

TL-01: The system shall simulate multi-sensor spacecraft navigation configurations in flight

TL-02: The system shall determine the accuracy of modelled navigation configurations over chosen flight scenarios

TL-03: The system shall optimise the navigation system according to the user requirements

FUNCTIONAL

To design a system capable of modelling and determining the accuracy of multi-sensor navigation configurations in addition to sizing, the following aspects must be considered in combination with the:

- True spacecraft orbit or trajectory
- Models of the sensors to be used
- Modelling the sensor observables of the sensors
- Fusion of the multi-sensor information
- Filtering the observational data and dynamical data to produce a navigation estimate

Which in turn lets the functional requirements be defined:

SAT-F-01 : The system shall model positional navigation

SAT-F-01.01 : The system shall model orbits and trajectories

SAT-F-01.02 : The system shall model spacecraft navigational sensors

SAT-F-01.02.01 : The system shall model the required navigational observables

SAT-F-01.03 : The system shall model navigational algorithms

SAT-F-01.04 : The system shall assess navigation system quality

SAT-F-02 : The system shall size navigation systems

SAT-F-03 : The system shall be capable of modelling more than one sensor in the system

SYSTEM

The system should be capable of modelling multiple sensors and different navigation filters. It should also be expandable to innovations in these fields. With expandability and generality then being two driving factors for this system, the data transfer between the different components must be standardised. Furthermore, user customisation of modules is also important. With these points in mind, the following system requirements are generated:

SAT-SYS-01 : The system shall be comprised of independent components

SAT-SYS-02 : The system shall use a standardised parameter set

SAT-SYS-03 : The system shall be user-configurable

3.1.4. EXPANDABILITY

Based on the SAT-SYS requirements and for the tool to remain relevant and be used in a design setting, the software should be expandable. This is difficult to validate as a requirement, and so is defined as a software feature. Expandability places importance on the logical definition of the software, such that if additions are wished to be made - such as additional sensors, filters or orbital aspects, this may be done without requiring the software to be completely disassembled or the architecture changed.

3.2. ARCHITECTURE DEFINITION

Based on the previous defined requirements, there are four clear top-level components in addition to expandability, which allow the system to be considered in a modular way:

- Orbits and Trajectories - which provides the true state of the s/c
- Sensors - which provides the observational input to the navigation system
- Navigation - which provides a state estimate of the s/c
- Analysis - which compares the true state and the state estimate to assess the quality of the navigation system

From this, in addition to the system requirements defined above, the following system architecture is generated and shown in figure 3.2

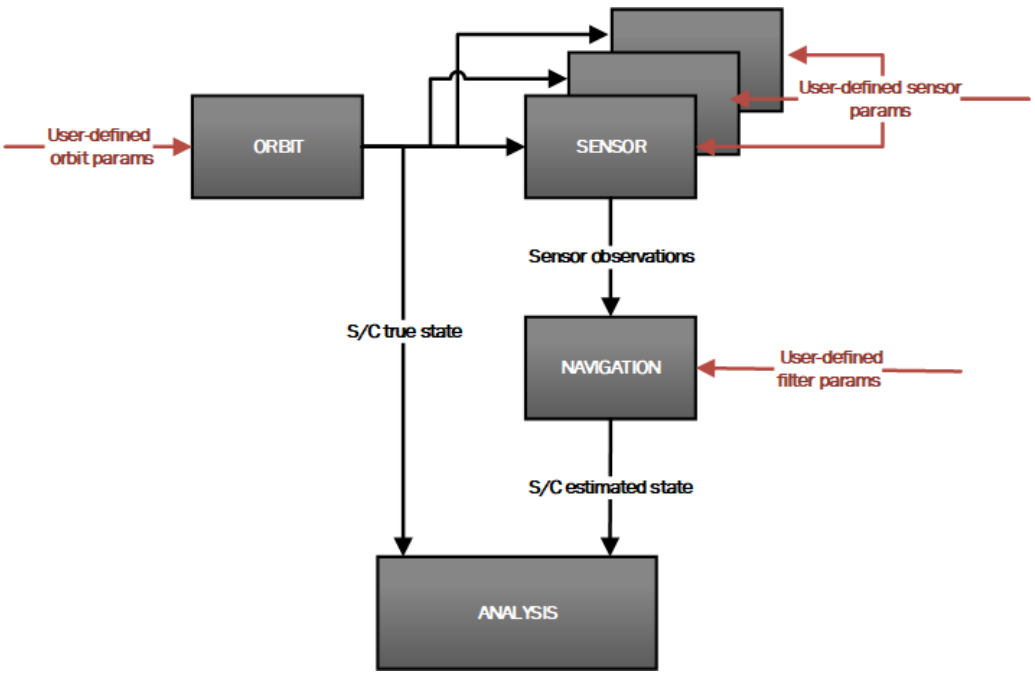


Figure 3.2: Architectural overview of the simulation test bed

3.2.1. INPUT/OUTPUT

For the different modules, a prospective I/O is generated. For the orbital module which is generating and therefore outputting the true state of the s/c, this input requirements are the definition of the trajectory/orbit, and the

time. For the sensor, the sensor observation will be formulated from the specific type of sensor (with its observation model), the relevant sizing parameters which include noise, what the sensors are observing - the observables, and finally if the sensor is to provide an observation, based on the time. For the navigation module, the filter is required, with specific dynamic model, in addition to the sensor observations and when an update is required - based on the time. The navigation module will provide an estimate of the s/c state. Finally the analysis module will take both the true state and the estimated state, from which the navigation system will be assessed.

Table 3.1: Prospective input/output of the modules

Module	Inputs	Outputs
Orbit	Trajectory definition, Time	True state of s/c
Sensor	Sensor types, Sizing parameters, Observables, Time	Sensor observations
Navigation	Filter, Sensor observations, Time	Estimated state of s/c
Analysis	True state of s/c, Estimated state of s/c,	Analysis of navigation system

The following sections will generate a high-level design for the modules defined in the architecture.

3.3. ORBIT MODULE

Generating the true state of the s/c may have two definitions:

1. The relative true state of the s/c against which the navigation output will be compared.
2. The absolute true state which is the real-life trajectory the s/c would follow if it were to be launched.

The two points aren't necessarily mutually exclusive, but point two is very difficult to achieve. There exist tools for precise orbit determination [52] which tend to reality, at the cost of computational resources. And indeed, an ideal model will simulate an exact orbit from which a more realistic assessment of the navigation performance will be achieved. However, a driving requirement of the this work is the expandability of the system and developing a high fidelity orbital model is labour-intensive. If only point one from above is confirmed, then the relative performance of two navigation systems can be compared, provided they both use the same dynamics. With this in mind, a basic Keplerian propagator is chosen as the initial base for the Orbit Module. To this, additional perturbations may be added in the future.

3.3.1. DEFINING A TRAJECTORY: USER INPUTS

There are two categories of parameter which are needed to define a trajectory in addition to a dynamical model: Those which may be considered pertinent to the orbit (spatial), these are considered for a Keplerian orbit, and those which are considered pertinent to time (temporal).

Table 3.2: Orbital user inputs

Parameter	Description
Spatial	
Reference frame	The spatial frame to which the navigation system is referenced. Assume a planet-centered inertial frame.
Central orbital body	Sun, Earth and Mars
Orbital Parameters	Definition elements of the orbit - Assume Keplerian: Semi-major axis, eccentricity, inclination, right ascension of the ascening node and argument of perigee
Temporal	
Reference time frame	The temporal frame to which the navigation system is referenced. Assume Temps Dynamique Barycentrique (TDB)
Start and end time	The duration of the simulation
Update time step	The length of a simulation epoch

3.3.2. UNITS

With a mind to expandability of the system, the prospect of units arises when considering orbits, times and orbital bodies. To have the system be general such that a new component, orbit type be added there needs to be a standard treatment of units This leads to a new requirement:

SAT-SYS-02.01: The definition and conversion of units shall be standardised and common to all software components

3.4. SENSOR MODULE

Although the sensor module has a single output function: to produce the sensor measurements, each sensor has a different working mechanism. Defining an architecture for this module will therefore require it to be high level.

3.4.1. THE GENERIC SENSOR

The generic sensor is one which is provided with observables, from which a measurement is given. The measurement itself is considered to be determined by the governing equations of the observation, in addition to specific sizing parameters as well as noise. As each sensor has a different working method, to have a single sensor module, there must be a standard definition for the working components of a sensor. The following requirements are generated:

SAT-SENS-01 : The sensor shall have observational equations

SAT-SENS-01.01 : The observation equations shall use the true state of the s/c in addition to relevant observables

SAT-SENS-02 : The sensor shall have a library of relevant observables when required

SAT-SENS-03 : The sensor shall model relevant noise sources and sizing parameters

3.4.2. SENSOR TYPE ANALYSIS

Making a general integrated navigation system involving pulsars requires the investigation into the different navigation types. The state vector of an observer with respect to navigation may be broken down into position, velocity, and other parameters such as attitude [53]. A sensor should observe something which will provide information to define at least one of these parameters in at least one dimension. To that end, there are therefore two aspects to navigation involving sensors: the first is taking the raw observational data - such as: timing information as in pulsars; changes in spectra in radiometers; or angles in star-trackers; and the second is converting this into usable navigation information - the position, velocity or other. It is useful to consider which sensors exist and the information they are able to provide. Sensors for navigation can be split into the following categories [54]:

- Vision
- Attitude and Relative Navigation
- Active Ranging
- Beacons

- Radial velocity Sensors

In principle, the test-bed could be expanded to include all these sensor types. However to limit the scope and complexity of this work, only those navigation types which may be considered long-range will be taken further. Active ranging will therefore not be considered, as this relies on the emission, reflection, and detection of an observer-borne signal. At the distances being considered for deep space travel, the time to reception and the diminishing signal strength makes this sensor type unfeasible. Vision, attitude and relative navigation sensors may broadly be grouped in the same category. However, the s/c attitude state will be ignored as defined per the research aims, as the navigation system is positional and therefore may be considered independent of attitude. However, as angles-only navigation is possible this sensor type will be considered.

Further, vision-based navigation, where specific features of objects, such as craters will not be considered in detail. Although navigation through the measurement of planetary/celestial body radii or star trackers may be grouped in the category of vision-based navigation, the more complex types - requiring feature recognition and image processing will be omitted from this work.

This leaves beacons, radial velocity sensors, some vision based and relative sensors. Other than radial velocity sensors, the different types of sensor that exist in the other categories, prevent the generalisation of theory for the categories. From this then, the angle sensor and radial velocity sensor will be chosen as the two sensors to implement into SAT-ANS.

RADIAL VELOCITY - DOPPLER

From here on, it will be assumed that the method through which the radial velocity is detected, is the doppler shift. Doppler sensors detect changes in the observed spectra of EM emission sources due to the relative radial velocity of the observer. This relative velocity causes a Doppler shift in the detected spectrum along the lines of (for non-relativistic cases) [55]:

$$\Delta v = \left(\frac{f}{f_0} - 1\right) c \tag{3.1}$$

where Δv is the radial speed relative to the emission source, f_0 is the reference base frequency in inertial space, f is the detected frequency, and c is light speed.

This may then be translated to the spacecraft velocity as follows: Let the radial velocity of the emission source (assumed a star) in an inertial frame be

$$\Delta v = \frac{\mathbf{r_{star}} \cdot \mathbf{v_{star}}}{|\mathbf{r_{star}}|} \tag{3.2}$$

For the spacecraft state:

$$\mathbf{r_{star}} = \mathbf{r_{SC}} + \mathbf{b} \tag{3.3}$$

$$\mathbf{v_{star}} = \mathbf{v_{SC}} + \dot{\mathbf{b}} \tag{3.4}$$

where **b** refers to the reference position of the star (which would be defined in a library). Note however that a single star will only give information on the specific radial speed relative to that star.

A specific doppler radial velocity sensor such as a spectrometer will have a spectral resolution (the minimum observable wavelength) and a spectral range.

For a specific base wavelength, the detected wavelength, due to Doppler shift would be:

$$\lambda = \left(1 + \frac{\Delta v}{c}\right) \lambda_0 \tag{3.5}$$

Current State of the Art For use further later on in this work, the current state of the art with respect to the use of Doppler sensors (a spectrometer) is investigated: Although not COTS, the Juno spacecraft contains a spectrometer which has a spectral resolution ranging from 0.4 to 1.1 nm[56]. Based on this value, the current state of the art for s/c will be taken to be 1 nm.

ANGLES-ONLY

For angles only navigation, beacons are used and their angular position relative to the observer will aid navigation. The relevant equations are:

For a beacon located at [x,y,z],

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \tag{3.6}$$

$$\phi = sin^{-1}\left(\frac{z}{\sqrt{x^2+y^2+z^2}}\right) \tag{3.7}$$

where the sensor is detecting the altitude and azimuth angles, θ and ϕ.

Current State of the Art The most common-used angle sensor is the star tracker. The current state of the art with respect to star trackers is the SED26, which is capable of providing 1 angular second accuracy for attitude. This will be taken to be the current state of the art. [57].

3.4.3. USER INPUTS

Based on the above descriptions, the user inputs for the sensor module may be defined.

Table 3.3: Sensor module user inputs

Parameter	Description
Spatial	
Specific sizing	The parameters which define the working aspects of the sensor (if required)
Noise	The (additive) noise to add to the measurements
Observables	A library containing the sensor observables
Temporal	
Update rate	The effective integration/observation time between measurements

3.5. NAVIGATION MODULE

In order to satisfy SAT-F-01.03 and SAT-F-03, the navigation module should be designed to use algorithms capable of modelling multiple sensors which may have a varying number of observations in time (such as PNAV). In addition to this, the level of integration should be investigated.

3.5.1. INTEGRATION ARCHITECTURE

Making the navigation module invariant of the specific navigational algorithm is important for the expandability of the system. However, multiple integration architectures for multi-sensor navigation exist and must be investigated to find the most suitable. This will be done by first giving a brief description of some of the different possibilities, and then implementing a trade-off.

ARCHITECTURES DESCRIPTION

Some of the most relevant different architectures for multi-sensor integration are from [5].

Cascaded Integration (CAS) This architecture uses a separate navigation filter per sensor, each providing a navigation solution, which are then fused to produce a final solution.

Centralised Integration (CEN) Directly uses the navigation sensors and combines them through an estimation algorithm, thereby providing a single navigation solution with no intermediaries.

Federated Filtered Integration (FFI) Similar to Cascaded integration. Uses local filters with input from the navigation estimate to improve specific sensor navigation estimates through feedback, if the observation equations have for example dynamics which depend on the state (e.g a state estimate based on a measured signal which is affected by doppler would be improved by using the velocity state estimate).

Hybrid (HYB) Based on the sensors being used, different (sub)-architectures are chosen within the same navigation system.

TRADE-OFF

To formulate a trade-off of the different architecture concepts, trade-off criteria must first be generated. Three criteria are developed with the requirement traceability indicated in brackets:

Independence (TL-01, SAT-F-03) The independence of the navigation architecture refers to the impact of different aspects of the architecture to the rest of the system (such as different components requiring multiple inputs from the rest of the architecture components). This may be simply considered as the number of separate navigation solutions required to form the final state estimation

Scalability (TL-04) The scalability refers to the functionality of the architecture not changing with the number of sensors

Generality (SAT-F-01.03) This work does not look to optimise the performance of specific navigation systems, however this maybe a possibility in the future, therefore the ability to change/augment the architecture is assessed.

The value of the trade off elements will range from 1-5 and the results will be motivated.

Table 3.4: Navigation architecture concept trade-off

Concept \ Criterium	Independence	Scalability	Generality	Totals
CAS	2	4	5	11
CEN	5	5	3	13
FFI	2	2	1	5
HYB	1	2	1	4

RESULTS

Table 3.4 shows the results of the trade-off. Detailed motivation of results are shown below.

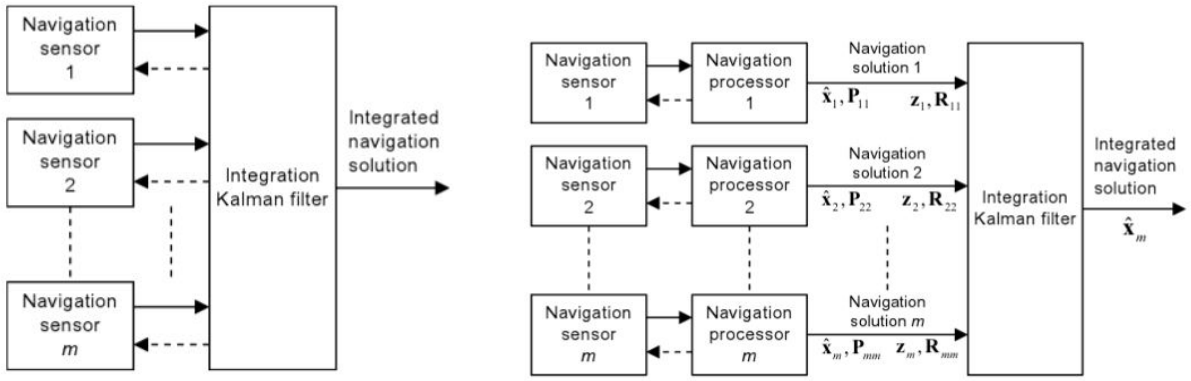
CAS Every sensor requires its own navigation processor for this architecture. This means that the independence of the system is low as each sensor outputs to a processor, which in turn provides a navigation solution to the Kalman filter. Therefore an independence rating of 2 is given. Additionally, the scalability is limited, as each new sensor also requires an additional processor, but the Kalman filter can remain the same (in principle), so this receives a scalability score of 4. Finally, CAS is the most general, as the output to the filter is already in state-space, and just a weighted sum based on the covariance is made. As this information may be used before it is added to the final filter, this leads to a score of 5.

CEN A single input from each sensor is required to the Kalman filter for this architecture, giving an independence score of 5. Further as there is no intermediate filter or processor, this architecture is the most scalable for the sensor number, further receiving a score of 5. Finally, as the input to the filter is in measurement-space, the conversion is required within the filter itself, somewhat limiting the generality. However, the input can be altered to the filter if future improvements or optimisations are desired. This leads to a score of 3.

FFI For each sensor, there is a separate local filter, leading to a dependence of the final filter on many inputs and possible feedback from the main filter, leading to an independence of 2. The addition of more sensors requires many additional components, however this could remain somewhat independent of the other sensors. This leads to a scalability score of 2. The alteration of inputs to the final filter would require a complete redesign of the system, leading to a generality score of 1.

HYB Rather than being single sensor dependent, this architecture is dependent on the combination of sensors being used. This means that there may be interdependencies between all components of the system. This leads to an independence score of 1. The scalability and generality is similar to FFI, and receives a score of 2 and 1 respectively.

CAS and CEN score much higher than FFI and HYB. This is generally due to them being much simpler than their counterparts, and therefore easier to implement, scale and augment. FFI and HYB are quite specific to the sensors being implemented, and so would be difficult to generalise such that any sensor type would be usable. FFI and HYB are then discarded, and although there is a clear concept from the trade off, CAS and CEN are considered more deeply.



(a) Centralised (b) Cascaded

Figure 3.3: Different types of architecture for sensor integration being considered [5]

As figure 3.3 shows, specific navigation solutions, each with their own covariance matrix are produced using the cascaded method. This must then be processed using the integration method (a Kalman filter in the diagram). However, CAS presupposes the format of the filter input, which in this case is the Kalman filter. Although likely to be a Kalman filter, if another filter entirely were to be required for the navigation assessment, a new series of navigation processors would be required. To remain completely independent of the filter, the centralised architecture is therefore chosen.

3.5.2. FILTERING/FUSING METHOD

For the specific implementation of the navigation module, an integration filtering method must be chosen. Within this, some additional requirements are generated. From the two sensors chosen, the angle sensor has non-linear observation equations. In addition to this, the basic Newtonian gravitational differential equation is also non-linear.

SYS-NAV-01 The filtering technique shall operate with non-linear systems

There are many filtering techniques which exist for estimation problems such as navigation. However, to reduce the scale of the trade-off only Kalman-type filters will be considered - which have been widely implemented in navigation systems and sensors [58].

Kalman filtering is a process which enables the combination of imperfect state estimations based on dynamical modeling of a system, and imperfect state estimations based on sensor readings, to form more accurate estimates of the state. This section will detail the use of Kalman filters and will motivate the choice for the system.

To introduce the filter, a discrete linear system is assumed with additive gaussian noise. Let a linear state-space system in discrete time be defined as [59]:

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{3.8}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{3.9}$$

where $\mathbf{x}$ is the state vector, $\mathbf{F}$ is the state transition matrix, $\mathbf{G}$ is the input control matrix, $\mathbf{u}$ is the input vector, $\mathbf{w}$ is the process noise vector, $\mathbf{y}$ is the output vector, $\mathbf{H}$ is the observation matrix, $\mathbf{v}$ is the observation noise, and k is the current epoch. The process and observation covariance matrices $\mathbf{Q}$ and $\mathbf{R}$, are considered to be white Gaussian with zero mean.

The linear Kalman filter may be used to estimate the system state $\mathbf{x}$ in time, according to the following: first a prediction is made about the system and it's corresponding noise characteristics, a measurement is made, and the state and noise are updated according to a weighted combination of the process information and the observation information. Note that in this case the s/c will not make any corrective manoeuvres, the control matrix will be omitted from here. In principle this functionality could be added at a later date.

Prediction The prediction step works using information from the previous time-step, or initial information about the system.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1} \tag{3.10}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \tag{3.11}$$

here, the state and state covariance matrix have been predicted as *a-priori* $(k|k-1)$ estimates based on a previous state estimates $\hat{\mathbf{x}}_{k-1}$

Update Now, using the observation information, the *a-priori* estimates may be updated:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{3.12}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}) \tag{3.13}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{3.14}$$

where **K** is known as the Kalman gain - a weighting factor based on the relative confidence in the process and the observation. The final state, as is outputted by the filter, is known as the *a posteriori* estimate.

This is a very useful algorithm with many different applications. One main drawback however, is that the Kalman filter assumes both a linear process and measurement - as is specified by the implementation of the Kalman gain. However, the navigation test-bed should be capable of integrating the information of multiple sensors and be generally extendable also from a process perspective. This means it should be capable of dealing with strong non-linearity, such as is present in pulsar navigation. Although the linear Kalman filter has been applied to non-linear systems, it is deemed not a good choice for this system as a test case. From this there are 3 main options: Extended Kalman filter (EKF), Unscented Kalman filter (UKF), and particle filter (PF) [60].

The EKF has the benefit of computational efficiency over the other two. This however comes at the expense of analytical complexity. Its implementation is similar to the linear Kalaman filter, although it requires the calculation of the Jacobian of the various observation and dynamic equations, to linearise the system over a time-step. This has an additional drawback, which is the linearisation error which occurs if the relevant process of observation equations are very strongly non-linear. This can lead to incorrect estimates in addition to instabilities. For a system capable of modelling general sensors which may be driven by strongly non-linear functions, this is considered unacceptable. The EKF is therefore discarded. This leaves the UKF and the PF. Both work by a similar method - the nonlinear transformation. The PF has additional mathematical complexity, therefore the UKF algorithm will initially be described and then the PF elaborated based on this.

3.5.3. UNSCENTED KALMAN FILTER

This section is based on the work by Rudolph Van der Merwe on the topic [6]. The principle of the UKF is the assumption that it is easier to describe the statistics of a random variable than an arbitrary non-linear function. The filter therefore tries to approximate the main aspects of a distribution through a series of points, and then pass these points through the non-linear functions of interest.

UNSCENTED TRANSFORMATION

Take an arbitrary non-linear function

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) \tag{3.15}$$

where x is an L-dimensional random variable with mean $\bar{\mathbf{x}}$ and covariance $\mathbf{P}_\mathbf{x}$. The unscented transform can be used to calculate the statistics of **y** up to the covariance:

1. $2L+1$ weighted samples - sigma-points $S_i = \{w_i, \boldsymbol{\mathcal{X}}_i\}$ are chosen to completely define the the mean and covariance of **x** The selection must satisfy the following relations:

$$\boldsymbol{\mathcal{X}}_0 = \bar{\mathbf{x}} \qquad i = 0 \tag{3.16}$$

$$\boldsymbol{\mathcal{X}}_i = \bar{\mathbf{x}} + (\sqrt{(L+\lambda)}\mathbf{P}_\mathbf{x})_i \qquad i = 1,...,L \tag{3.17}$$

$$\boldsymbol{\mathcal{X}}_i = \bar{\mathbf{x}} - (\sqrt{(L+\lambda)}\mathbf{P}_\mathbf{x})_i \qquad i = L+1,...,2L \tag{3.18}$$

   where λ is a factor defined by:

$$\lambda = \alpha^2(L+\kappa) - L \tag{3.19}$$

   which optimises the choice of sigma point. κ is chosen to ensure the covariance matrix is positive definite ($\kappa \geq 0$). α determines the spread of the sigma points and should be chosen such that it avoids the sampling of non-local effects if there are very strong non-linearities ($0 \leq \alpha \leq 1$) The term $(\sqrt{(L+\lambda)}\mathbf{P}_\mathbf{x})_i$ is the i-th column of the matrix square-root of the weighted covariance matrix $(L+\lambda)\mathbf{P}_\mathbf{x}$. The sigma points each have weight w_i

defined as:

$$w_0^{(m)} = \frac{\lambda}{L+\lambda} \qquad i = 0 \tag{3.20}$$

$$w_0^{(c)} = \frac{\lambda}{L+\lambda} + (1-\alpha^2+\beta) \qquad i = 0 \tag{3.21}$$

$$w_i^{(m)} = w_i^{(c)} = \frac{1}{2(L+\lambda)} \qquad i = 1,...,2L \tag{3.22}$$

$$\sum_{i=0}^{2L} w_i = 1 \tag{3.23}$$

where $^{(m)}$ and $^{(c)}$ refer to mean and covariance respectively. Additionally, β is a weighting term which incorporates higher order terms in the distribution (for Gaussian, $\beta = 2$).

2. The sigma points are now propagated through the non-linear function:

$$\boldsymbol{\mathcal{Y}}_i = g(\boldsymbol{\mathcal{X}}_i) \qquad i = 0,...,2L \tag{3.24}$$

3. The mean, covariance and cross-covariance of $\mathbf{y}$ can then be calculated:

$$\bar{\mathbf{y}} \approx \sum_{i=0}^{2L} w_i^{(m)} \boldsymbol{\mathcal{Y}}_i \tag{3.25}$$

$$\mathbf{P}_{\mathbf{y}} \approx \sum_{i=0}^{2L} w_i^{(c)} (\boldsymbol{\mathcal{Y}}_i - \bar{\mathbf{y}})(\boldsymbol{\mathcal{Y}}_i - \bar{\mathbf{y}})^T \tag{3.26}$$

$$\mathbf{P}_{\mathbf{y}} \approx \sum_{i=0}^{2L} w_i^{(c)} (\boldsymbol{\mathcal{X}}_i - \bar{\mathbf{x}})(\boldsymbol{\mathcal{Y}}_i - \bar{\mathbf{y}})^T \tag{3.27}$$

This method for approximating the statistics of non-linear functions is correct up to second order (and exact for linear functions). Now that the defining feature of the UKF has been defined, the algorithm may now be described:

ALGORITHM

Similar to the linear Kalman filter, the UKF consists of two main steps, the predict and update step.

Predict Let the process model be defined by the non-linear equation $f()$ assumed to be time dependent, and let the measurement model be defined as $h()$.

1. Calculate the sigma points

$$\boldsymbol{\mathcal{X}}_{k-1}^a = \left[\hat{\mathbf{x}}_{k-1}^a, \quad \hat{\mathbf{x}}_{k-1}^a + \gamma\sqrt{\mathbf{P}_{k-1}^a}, \quad \hat{\mathbf{x}}_{k-1}^a - \gamma\sqrt{\mathbf{P}_{k-1}^a}\right] \tag{3.28}$$

here, a refers to the sigma points relating to the state, and the noise; and $\gamma = \sqrt{L+\lambda}$

2. Update the state sigma points based on previous epoch

$$\boldsymbol{\mathcal{X}}_{k|k-1}^x = \left(f(\boldsymbol{\mathcal{X}}_{k-1}^x), \mathbf{u}_{k-1}\right) \tag{3.29}$$

3. Form an estimate of the state mean based on a weighted sum of the sigma points

$$\hat{\mathbf{x}}_k^- = \sum_{i=0}^{2L} w_i^{(m)} \boldsymbol{\mathcal{X}}_{i,k|k-1}^x \tag{3.30}$$

4. From the state mean estimate, generate an estimate of the state covariance

$$\mathbf{P}_{\mathbf{x_k}}^- = \sum_{i=0}^{2L} w_i^{(c)} (\boldsymbol{\mathcal{X}}_{i,k|k-1}^x - \hat{\mathbf{x}}_{\mathbf{k}}^-)(\boldsymbol{\mathcal{X}}_{i,k|k-1}^x - \hat{\mathbf{x}}_{\mathbf{k}}^-)^T + \mathbf{Q} \tag{3.31}$$

Update

1. Transform the state-space sigma points to measurement-space

$$\mathcal{Y}_{k|k-1} = h\left(\mathcal{X}^x_{k|k-1}\right) \tag{3.32}$$

2. Estimate the observation mean

$$\hat{\mathbf{y}}^-_k = \sum_{i=0}^{2L} w_i^{(m)} \mathcal{Y}_{i,k|k-1} + \mathbf{R} \tag{3.33}$$

3. Estimate the observation covariance

$$\mathbf{P}_{\tilde{\mathbf{y}}_\mathbf{k}} = \sum_{i=0}^{2L} w_i^{(c)} (\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}^-_k)(\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}^-_k)^T \tag{3.34}$$

4. Estimate the cross-covariance

$$\mathbf{P}_{\mathbf{x_k y_k}} = \sum_{i=0}^{2L} w_i^{(c)} (\mathcal{X}^x_{i,k|k-1} - \hat{\mathbf{x}}^-_k)(\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}^-_k)^T \tag{3.35}$$

5. Generate the Kalman gain

$$\mathbf{K}_k = \mathbf{P}_{\mathbf{x_k}\tilde{\mathbf{y}}_\mathbf{k}} {\mathbf{P}_{\tilde{\mathbf{y}}_\mathbf{k}}}^{-1} \tag{3.36}$$

6. Obtain weighted estimate of the new state mean using the observations

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}^-_k + \mathbf{K}_k(\mathbf{y}_k - \hat{\mathbf{y}}^-_k) \tag{3.37}$$

7. Finally, generate a weighted estimate of the state covariance

$$\mathbf{P}_{\mathbf{x_k}} = \mathbf{P}^-_{\mathbf{x_k}} - \mathbf{K}_k \mathbf{P}_{\tilde{\mathbf{y}}_\mathbf{k}} \mathbf{K}_k^T \tag{3.38}$$

COMPARISON

Figure 3.4 shows an arbitrary distribution being propagated though a non-linear function, from which the mean and covariance is estimated by a sampling method (such as those used in a particle filter), the linear Kalman Filter and the UKF. As can be seen, the mean and covariance estimated by the linear KF is very far from the true value. The comparison therefore lies with the sampling methods and the UKF and this comes down to the computational requirement. A sampling method such requires the propagation of many points generated via a Monte-Carlo method for example, whereas the UKF requires only twice the number of points as the state variable dimension. If the points are well distributed, and non-local non-linearities are avoided with the points, the mean and covariance can be well estimated. This method is therefore chosen, however if it is found that the UKF does not give suitable results in the test-bed, the particle filter may need to be implemented.

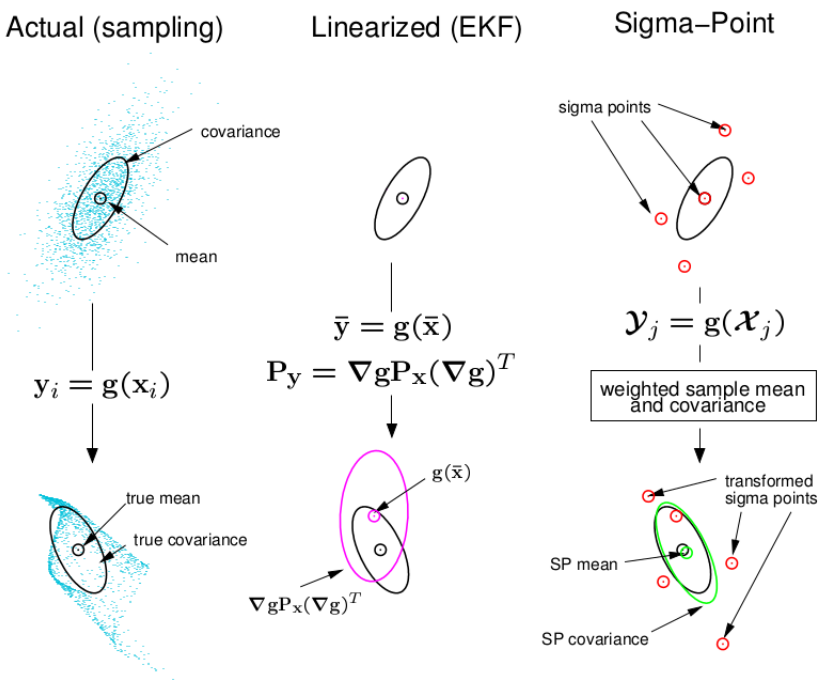


Figure 3.4: Comparison of the estimation of mean and covariance for through non-linear functions, by a particle filter, a linear Kalman filter and the UKF [6]

3.5.4. SENSOR FUSION

With the filter defined, the attention now turns to how best to fuse the observations from different types of sensors. If the sensor update rates are the same and the availability of observations is constant, then the observation matrix can just be extended to include the additional information. However in reality this will not be the case - some sensors may take minutes to produce a reading, compared to the sub-second tracking time of star-trackers [57]. The observation and observation-noise matrices will therefore need to be dynamically defined based on the availability of the measurement data.

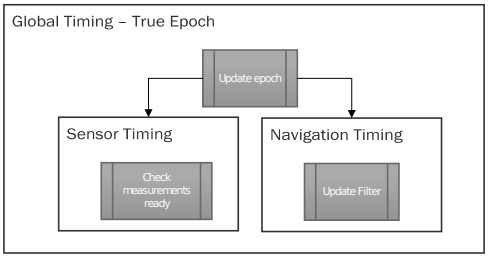


Figure 3.5: Timing in the model

This also leads onto the question on how time will be handled in the model. Figure 3.5 shows the hierarchy of time in the model, with the true base epoch being driven by the global timer which feeds into the sensor and navigation modules. Clock noise from the sensor/navigation side is not yet considered, and is implemented in the pulsar navigation chapter.

3.5.5. USER INPUTS

Based on the above descriptions, the user inputs for the navigation module may be defined.

Table 3.5: User navigation inputs

Parameter	Description
Spatial	
Filter parameters	The parameters which define the working aspects of the filter
Process noise	The estimated error due to aprroximated of the on-board dynamic model
Inital Covariance	Uncertainty in initial conditions
Temporal	
Update rate	The time between filter updates

3.6. ANALYSIS

The output of the system will be the true state of the s/c and the filter-estimated state, in addition to the filter-estimated covariance. To give a standardised comparison the following information will be given based on output:

- Graphs with the true state of the system
- Graphs with the error between the true state and the estimate
- On the above graphs, the covariance calculated through Monte Carlo simulations (where relevant)
- The covariance calculated from a single iteration
- The RMS error of the state dimensions
- The standard deviation

It will also be useful to have a point of reference for knowing the quality of the navigation solution without needing to relatively compare simulation runs with different tuning parameters. There are several ways to do this, however all methods evaluate the contribution of specific measurements to the state vector.

3.6.1. OBSERVABILITY AND LIE METHOD

The term *observability* is often used in control theory and means that the state of a system should be fully recoverable from the measurements made. Explicitly, for a system described by[61]:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \tag{3.39}$$

with initial value $\mathbf{x}(0) = \mathbf{x}_0$ which has the output

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t)) \tag{3.40}$$

If the state vector has dimension n and the observation vector has dimension p, then the following maps hold and are smooth:

$$\mathbf{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^n \tag{3.41}$$

$$\mathbf{h} : \mathbb{R}^n \longrightarrow \mathbb{R}^p \tag{3.42}$$

$$\tag{3.43}$$

In most cases, the state $\mathbf{x}$ cannot directly be observed and is therefore made by observing $\mathbf{y}$ through another system (an observer) which is transformed back to the state.

A system such as the one defined in 3.39, 3.6.1 is said to be observable if two different starting states which have separate behaviour under the same control (i.e. are distinct), are distinguishable via the output. Practically, different sensors have different mapping functions (observation equations) $\mathbf{h}$, and so when combining sensors and if the availability of the observables is changing in time, the availability to extract the state fully from the observations may change. Evaluating the observability can therefore be useful in defining the availability of an optimum navigation solution [62].

To evaluate the observability, the following map may be used[61]:

$$\Theta : \mathbf{x} \mapsto \begin{pmatrix} \mathbf{y} \\ \dot{\mathbf{y}} \\ \vdots \\ \mathbf{y}^{(d)} \end{pmatrix} \tag{3.44}$$

For some fixed d, if Θ is invertable, then the system is observable. For a non-linear system however, it is very difficult to prove global invertability for general non-linear maps, therefore local invertability is used through the Implicit Function Theorem. Θ is locally invertable at some $\mathbf{x}_0$ if its Jacobian (also known as the observability matrix) has full rank:

$$rank\left(\frac{d\Theta(\mathbf{x})}{d\mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}_0}\right) = n \tag{3.45}$$

To calculate Θ and from this the observabilty matrix, Lie derivatives are used:

$$\mathbf{y}^{(k)}(t) \equiv L_{\mathbf{f}}^{k}\mathbf{h}(\mathbf{x}(t)) \qquad (k \in \mathbb{N}) \tag{3.46}$$

where

$$L_{\mathbf{f}}^{k}\mathbf{h}(\mathbf{x}) := \frac{\delta L_{\mathbf{f}}^{k-1}\mathbf{h}(\mathbf{x})}{\delta \mathbf{x}}\mathbf{f}(\mathbf{x}) \quad \text{with} \quad L_{\mathbf{f}}^{k}\mathbf{h}(\mathbf{x}) := \mathbf{h}(\mathbf{x}) \tag{3.47}$$

and this leads to a series representation of the output:

$$\mathbf{y}(t) = \sum_{k=0}^{\infty} L_{\mathbf{f}}^{k}\mathbf{h}(\mathbf{x_0})\frac{t_k}{k!} \tag{3.48}$$

Using the method above can aid the answering of the following questions:

With the current observations from the sensors, can the spacecraft state be fully defined?

What additional observables are required to make the state observable?

However, this technique is very computationally resource heavy, depending on the degree to which the derivatives are required. There are methods to mitigate this, such as automatic differentiation, however there is another separate method entirely though which the navigation solution may be assessed.

3.6.2. FISHER INFORMATION MATRIX AND CRAMER-RAO LOWER BOUND

Rather than looking at the system as a whole and the degree to which the output maps onto the input, it is possible to see the information content of each individual measurement and from this define the limit on navigation accuracy. It is also possible to take into account the random nature of the system and give a more stochastic analysis. For this, the measurement information is given through the Fisher Information Matrix (FIM) and the navigation limit is known as the Cramer-Rao Lower Bound (CRLB).

For a group of random parameters, $\theta = [\theta_1, ..., \theta_n]^T$ estimated from measured data $\mathbf{z}^N = [\mathbf{z}_1^T, ..., \mathbf{z}_N^T]^T$, the FIM matrix is defined as [63]:

$$\mathbf{F}(\theta) = -E\{\nabla_\theta[\nabla_\theta \ln p(\mathbf{z}^N, \theta)]^T\} \tag{3.49}$$

Where ∇_θ is the Jacobian with respect to the random parameters, and $p(\mathbf{z}^N, \theta)$ is the probability density function (pdf) of the state, given the parameters.

The minimum error covariance of the state estimates is related to the FIM by:

$$\mathbf{P} \geq \mathbf{F}^{-1} (= \mathbf{C}) \tag{3.50}$$

where $\mathbf{C}$ is the CRLB for unbiased estimators.

This pdf if often very difficult to define in complex systems with many interacting aspects. However, a recursive relation for the CRLB was developed to get around this issue.

For an additive Gaussian noise system, such as the one used:

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k) + \mathbf{w}_k \qquad k \in \mathbb{N} \tag{3.51}$$

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k) + \mathbf{v}_k \qquad k \in \mathbb{N} \tag{3.52}$$

the recursive CRLB is found to be [63]:

$$\mathbf{C}_{k+1|k}^{-1} = \mathbf{K}_{k+1}^{k+1} - \mathbf{K}_{k+1}^{k+1,k}(\mathbf{K}_{k+1}^{k} + \mathbf{C}_{k|k}^{-1})^{-1})\mathbf{K}_{k+1}^{k,k+1} + \mathbf{L}_k^k \tag{3.53}$$

The derivation for this is omitted here, but may be found in the reference (or is considered trivial and left up to the reader)

where

$$\mathbf{K}_{k+1}^{k} = E\{[\nabla_{\mathbf{x}}\mathbf{f}_k(\mathbf{x}_k)]^T\mathbf{Q}_k^{-1}\nabla_{\mathbf{x}}\mathbf{f}_k(\mathbf{x}_k)\} \tag{3.54}$$

$$\mathbf{K}_{k+1}^{k,k+1} = -E\{[\nabla_{\mathbf{x}}\mathbf{f}_k(\mathbf{x}_k)]^T\mathbf{Q}_k^{-1} \tag{3.55}$$

$$\mathbf{K}_{k+1}^{k+1} = \mathbf{Q}_k^{-1} \tag{3.56}$$

$$\mathbf{L}_{k}^{k} = E\{[\nabla_{\mathbf{x}}\mathbf{h}_k(\mathbf{x}_k)]^T\mathbf{R}_k^{-1}\nabla_{\mathbf{x}}\mathbf{h}_k(\mathbf{x}_k)\} \tag{3.57}$$

$$\tag{3.58}$$

This is clearly a simplification of the CRLB, as there is a first order linearisation of the system with respect to the state at every epoch. Similar to the motivation for the use of the EKF over the UKF, this may present problems for strongly non-linear systems. However, the ability to make an analytical approximation every epoch without the need to define the pdf, makes this method attractive.

It should be noted that there are expectation operators in equation 4.5.1 which must be evaluated. The pdf's for the process and measurement are additive Gaussian as described above, and may be defined as:

$$p(\mathbf{x}_{k+1}|\mathbf{x}) = \mathcal{N}(\mathbf{x}_{k+1} : \mathbf{f}_k(\mathbf{x}_k), \mathbf{Q}_k) \tag{3.59}$$

$$p(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}(\mathbf{z}_k : \mathbf{h}_k(\mathbf{x}_k), \mathbf{R}_k) \tag{3.60}$$

A further assumption is made and evaluated, which may allow the CRLB to be evaluated per single evaluation of the timestep over one simulation, as opposed to requiring an in-depth Monte Carlo analysis.

Assumption For a system with additive (white) Gaussian noise (AWGN) described by 3.60, the expected values of the measurement and process vector elements may be approximated by their true value, assuming independence between the elements.

Justification This assumption seems intuitive for a linear system. However, To test to see if this assumption holds for a non-linear system, the Newtonian gravitational equation is evaluated with AGN. A Monte Carlo analysis is run and compared against the true values, to see if they converge.

The convergence of the observation equation is intuitive as the noisy state is not propagated to the next epoch. For the process equation however, this is not the case. The current noisy state becomes the next true state thereby making the assumption that the system will converge to the true noiseless state over iterations more difficult. However, the derivation of the EKF uses exactly this assumption in its first order linearisation of non-linear equations with additive Gaussian noise [64]. It is therefore considered reasonable to assume.

3.7. VERIFICATION AND VALIDATION

The verification methods of the software will be chosen from the following: Inspection, Review, Analysis, Demonstration and Test. Additionally, as the software may be considered as a complex layered system, the verification will follow the Unit, Integration and Functional model, as shown in figure 3.6 through the waterfall model systems engineering approach. The design process follows the left hand flow, culminating in the coding. Following the coding, the verification and testing will occur along the hierarchy of the model. The term *unit* will be defined in the software development section. However, the validation of the requirements/functionality will happen at these testing stages, thereby ensuring that the system is validated from a bottoms-up principle.

However, other than the software testing, the requirements specified at the beginning of this section require validation approaches. These are outlined in tables 3.6 and 3.7

Table 3.6: Functional requirements validation

Requirement	Validation Method	Description
SAT-F-01.01	Test	Verification of the orbital module will validate this requirement
SAT-F-01.02	Analysis	The implementation of the relevant observation equations with noise
SAT-F-01.02.01	Observation	A library containing the relevant observables will be inspected
SAT-F-01.03	Test	Verification of the implemented filter will validate this requirement
SAT-F-01.04	Test	Comparison with numerical Monte Carlo simulations
SAT-F-02	Test	Comparison with numerical Monte Carlo simulations
SAT-F-03	Observation	Multiple sensors will be modelled and the result analysed

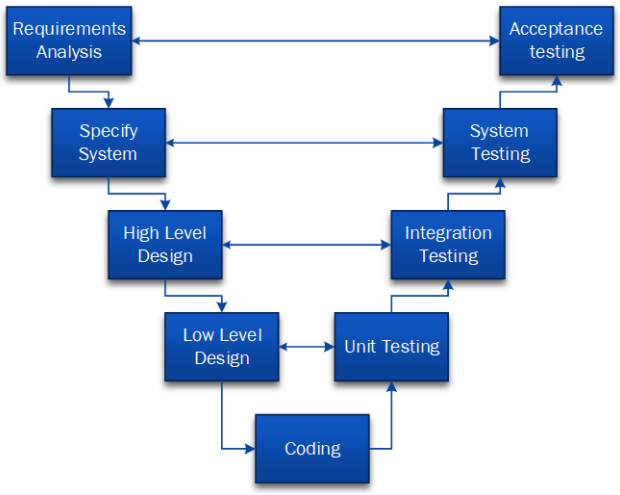


Figure 3.6: Water-fall systems engineering verification and validation model applied to software.

Table 3.7: System requirements validation

Requirement	Validation Method	Description
SAT-SYS-01	Test	The components will be shown to work irrespective of the other components (with relevant inputs)
SAT-SYS-02	Analysis	The input/output of all components will be assessed for irregularities
SAT-SYS-03	Test	Relevant parameters will be changed by the user and the system tested

4

SOFTWARE DEVELOPMENT

Based on the definition of requirements from the previous section, the software is developed in this section. This chapter is divided along the different software modules whose architecture and theory has been developed in the previous chapter. Within each module, the functional software architecture is described using flowcharts, and then relevant components within the module are described. To maintain overview of the modules and their working method, after their development, each module is individually verified. After the verification process, the software modules are integrated to make up the test-bed.

4.1. SENSOR MODULE

The sensor module is the most difficult module to generalise. This is due to the working principles defining each distinct sensor is different. However, each sensor type will contain the following aspects based on the previous chapter:

- Observation equations
- Library of observables, defining their positions/ephemeris and characteristics
- Update rate
- Standard deviations/covariances in observations
- Further sizing parameters

Having a general framework in which to define these aspects, so that the information (namely the observations and observation equations) may be exported to the navigation module is important, as it is a defining feature of the expandability of the system.

4.1.1. FUNCTIONAL ARCHITECTURE AND UNITS

The functional architecture of the sensor module may be split into three units in line with the software functions: The generation of the observables dictionary; the evaluation of the observation using the observable parameters; and the addition of white Gaussian noise. Note that the flow chart for the Make Observation function is not included, as this functionality is defined by the sensor observation equations.

Format Parameters

The observables in the library are iterated over, and the from the different parameters, a dictionary object is created.

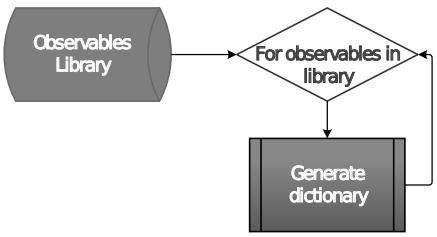


Figure 4.1: Format library parameters

OVERVIEW

In the new epoch (after the simulation time has been updated), the update time of the sensor is compared and checked to see if the measurement is ready. If it is, the observables are chosen from the dictionary, from which the observation equations are used to generate a noiseless observation. Measurement noise is then added to form the output measurement.

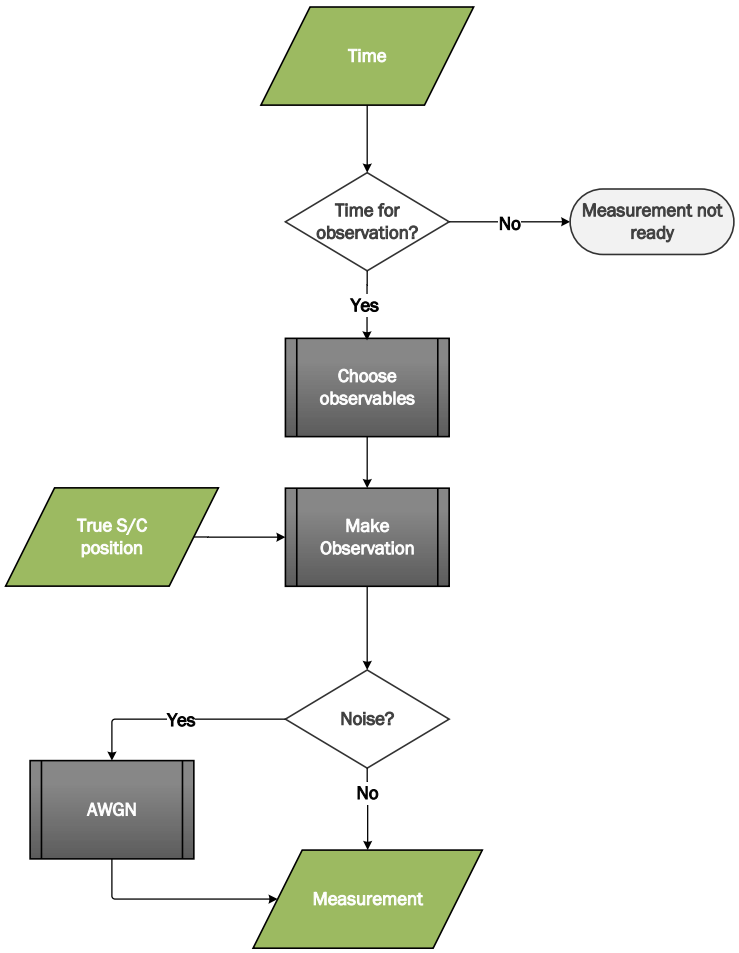


Figure 4.2: architectural overview of the sensor module in the test bed

Additive White Gaussian Noise (AWGN)

The formation of the noisy output measurement follows AWGN. This is zero mean white noise with standard deviation defined based on the sensor noise covariance.

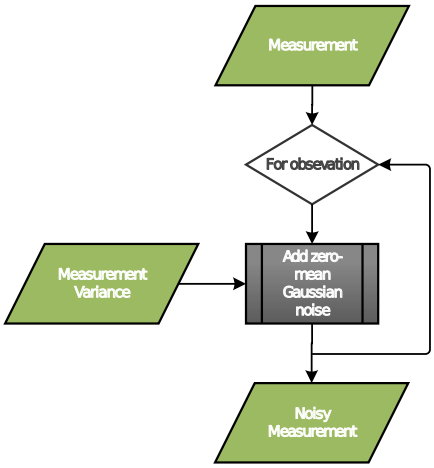


Figure 4.3: Sensor observation AWGN

4.1.2. ANGLE SENSOR

The implementation of the angle sensor follows the observation equations 3.7. For the filter however, as the measured angles are compared to those which are predicted based on the estimated s/c state, a residuals function is required.

VERIFICATION

The verification of equations 3.6 and 3.7, are by analysis. Three test case beacons are made, 1: [0,0,100 km], 2: [500 km, 500 km, 0], and 3: [100 km, 100 km, 100 km]. The first two should show the independence between θ and ϕ implemented in the sensor module, and the last should show a combination.

Table 4.1: Angle sensor observation equations verification

Case	theta [rad]	phi [rad]
1	0	0.5π
2	0.25π	0
3	0.25π	0.615

As expected, the first case shows the beacon on the z-axis, the second in the x-y plane, and the third in the positive x-y-z quadrant.

RESIDUALS AND MEAN

Rather than just being a subtract function, due to the periodicity of angles, an additional function is required to prevent singularities. Although quaternions may in the future be implemented for attitude-work, the following function has been implemented for residuals:

$$resid(\theta_1,\theta_2) = \text{Norm}(\theta_1 - \theta_2) \tag{4.1}$$

$$\text{Norm} = \begin{cases} (x mod 2\pi) - 2\pi, & \text{if } x mod 2\pi > \pi. \\ x, & \text{otherwise.} \end{cases} \tag{4.2}$$

This prevents situations where an estimate of an angle is just over zero, but the measured angle is just under 2π, an the residual is then calculated to be close to 2π rather than the true value close to zero.

The mean of the angles from the sigma points are calculated in the following way:

$$\bar{x} = \text{atan2}\left(\sum \sin(x)(W), \sum \cos(x)(W)\right) \tag{4.3}$$

Where the weights of the respective sigma points (if used) are given by W.

4.1.3. RADIAL VELOCITY SENSOR

The implementation of the radial velocity sensor is relatively simple, as the output measurement is not periodic. Equation 3.5 has been coded into the sensor module. The different observables are iterated over, the dot product of

the the (estimated) velocity of the s/c is made with the (normalised) position vector of the observable. From this the detected wavelength can be calculated. Based on the noise of the sensor, AWGN is added.

VERIFICATION

The verification of the radial velocity sensor follows a similar analytical method as the angle sensor. Here three test cases are mentioned - one where the observer is stationary, and so no doppler shift should be seen, and the second where the the observer is moving perpendicularly to the beacon and again no shift should be seen, and finally where the observer is moving radially toward the beacon, and a shift should be seen. Let a stationary beacon be placed at [100 km,0,0] with wavelength 3000 nm. The observer velocity in the three cases are, 1: [0,0,0], 2: [0, 5 km/s, 0] and 3: [5 km/s, 5 km/s, 5 km/s].

Table 4.2: Radial Velocity Verification

Case	reference wavelength	detected wavelength
1	3000 nm	3000 nm
2	3000 nm	3000 nm
3	3000 nm	3000.05 nm

As the table shows, only radial velocity produces a doppler shift.

4.1.4. LIBRARY

The library for the angle and radial velocity sensor contains the (Cartesian) positions of the observation beacons and the distance they are from the observer. With the use of the XML file format, this in principle could be extended to creating a class containing the propagation of true celestial beacons in a specific reference frame, however for a first implementation, this was deemed acceptable. The XML code below gives an example of a library entry. This format has been used for all other library entries in SAT-ANS sensor observables libraries.

```
<observable name="star1">
    <attribute name="direction.x" value="0.9" type="float"/>
    <attribute name="direction.y" value="0.3082" type="float"/>
    <attribute name="direction.z" value="0.3082" type="float"/>
    <attribute name="range_m" value="3e10" type="float"/>
    <attribute name="base_wavelength_nm" value="500" type="float"/>
    <attribute name="parallax" value="0" type="int"/>
</observable>
```

With regards to the reference frame, angles are usually defined relative to some reference point/object. Again, for the initial implementation of the tool, the reference position will be assumed to be the reference orbital body. Note that for this implementation, the beacons and reference objects are assumed to remain stationary relative to the reference body. Additionally, aspects such as eclipsing of observables have not been considered. This functionality could be added to future versions of this software for higher fidelity simulations.

For the radial velocity sensor, the same principle is used but a single reference wavelength is given for each observable from which the Doppler shift may be calculated.

4.2. ORBITAL MODULE

The implementation of the orbital module allows the user to define a two-body orbit centered on the Sun, Earth or Mars.

4.2.1. WORKING METHOD

To reduce the development time and complexity of the model, external libraries were used. The AstroPy and Poliastro python libraries were used to develop the orbital module. It has the following functionality:

- Specify reference bodies
- Specify reference times
- Specify the type of orbit and orbit length
- Propagate the orbit
- Output the state of the SC
- Specify and maintain units throughout the simulation

The orbit type first implemented in the model is the (reduced) 2-body Keplerian orbit. It is reduced insofar as the mass of the SC is considered negligible compared to the mass of the orbital body.

PROPAGATORS

Four different propagators are implemented into SAT-ANS. Two are included in the library Poliastro - Kepler and Cowell, and two have been added - Mean Motion and RK4. The reasoning behind their implementation is described below.

Mean Motion The Mean Motion solver, propagates the mean motion of the orbit, and then converts them to Cartesian state components. It may be considered a basic propagator specific for Keplerian orbits and also analytical if the state remains as orbital elements (the conversion to cartesian requires numerical methods to solve). The method is described further in the Verification section below.

RK4 For the implemented numerical solver, the Runge-Kutta 4 (RK4) algorithm[65], which was implemented to cope with more complex orbital scenarios than the 2-body problem. It works as follows: For some timestep, h, initial position $\mathbf{r}_0$ and initial velocity $\mathbf{v}_0$, the following partial timesteps are evaluated:

$$
\begin{aligned}
k_1 &= h\frac{d\mathbf{v}}{dt}(\mathbf{r_0}) \\
k_2 &= h\frac{d\mathbf{v}}{dt}(\mathbf{r_0} + \frac{k_1}{2}) \\
k_3 &= h\frac{d\mathbf{v}}{dt}(\mathbf{r_0} + \frac{k_2}{2}) \\
k_4 &= h\frac{d\mathbf{v}}{dt}(\mathbf{r_0} + k_3)
\end{aligned}
$$

From this, the new velocity and position may be calculated:

$$
\begin{aligned}
\mathbf{v} &= \mathbf{v_0} + \frac{k_1+2k_2+2k_3+k_4}{6} \\
\mathbf{r} &= \mathbf{r_0} + h\mathbf{v}
\end{aligned}
$$

RK4 is one of the methods used in the navigation system and so it's application is also tested in the orbital module.

Kepler This method relies on the conversion of the 2-body system to Kepler's equation and its efficient solution [66]. The mechanism for this uses Stumpff functions and has been implemented in the Poliastro library. It is shown with the flowchart A.2 in the Appendix.

Cowell The Cowell method for solving the differential equation is another numerical method and uses a Python library of numerical integrators SciPy to give the s/c's true position.

REFERENCE FRAMES

The implementation of the Orbit Module is general enough such that multiple reference systems may be defined. Particularly a centered-fixed reference frame may be useful in the future for the observation of features on the surface of planetary bodies. However for the first iteration of the model, the orbital body-inertial frame has been used. The orbit module outputs the position of the s/c in body-centered inertial coordinates, with the z-axis aligned with the body's pole, the x-axis aligned with the (Earth's) vernal equinox defined in the J2000 time reference and the y-axis forming a right-hand coordinate system. An inertial reference frame was chosen to limit the complexity and as the sensors chosen have the observables external to the reference body, there is no requirement to model the rotation of the body itself

COORDINATE TRANSFORMS

The method mentioned above, using the reduced Kepler equation, the orbital centre will be the centre of the orbital body. However, for different applications (pulsar navigation, namely), the state of the spacecraft will be required with respect to different coordinate systems. In this way, a coordinate transform method is implemented. For a transform, however the specific epoch and the relevant positions of other orbital bodies are required. This means using an external ephemeris, defining the positions, coordinate systems of relevant bodies.

Poliastro has the functionality of using the JPL de430 ephemeris to give the current position of the orbital body in barycentric coordinates. To transform between the body-centered inertial reference frame and the barycentric inertial reference frame, procession/nutation effects must be taken into account in addition to the position of the SSB. The AstroPy library contains the functionality for these transforms, but a test is made to see if the effects seen are periodic. An overview of the transform method is shown in 4.4. The procession of the North poles of the planets is of the order of tens of thousands of years [67]. The change in pole right ascension and declination must be taken into account for the barycentric conversion.

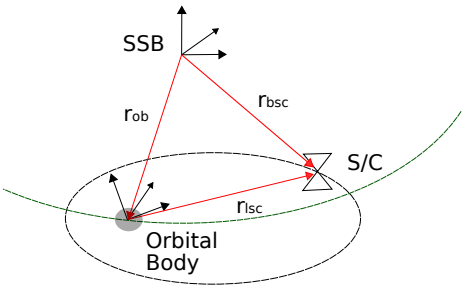


Figure 4.4: Transforming from orbital-inertial to Barycentric coordinates

As the Temporal reference frame used by the model is the J2000 reference, the reference pole direction is Earth. For the implemented planetary bodies of Earth, Mars and the Sun, the following RA and declinations with rates are included from AstroPy [68]:

Table 4.3: Changes in right ascension and declination of the poles of the different implemented planetary bodies, used for transforming to barycentric coordinates

Body	Pole RA	RA [deg/ Julian century]	Pole dec	dec [deg/ Julian century]
Earth	0	-0.641	90	-0.557
Mars	317.681	-0.106	52.887	-0.0609
Sun	286.13	0	63.87	0

Note, as the orbital model has been kept relatively simple, only the positions of the poles relative to the barycentre are considered. Aspects like precession and nutation have been omitted

For the transformation of the vectors the following relation holds:

$$\mathbf{x}_{body|SBB} = R_{body|SBB}(\mathbf{x}_{body|orb}) + \mathbf{x}_{orb|SBB} \tag{4.4}$$

where $R_{body|SBB}$ is the rotation matrix transforming the coordinate system along the angles stated above[69]:

$$R_{body|SBB} = \begin{bmatrix} \cos DEC & \sin DEC \sin RA & \sin DEC \cos RA \\ 0 & \cos RA & -\sin RA \\ -\sin DEC & -\cos DEC \sin RA & \cos DEC \cos RA \end{bmatrix} \tag{4.5}$$

and the system is translated based on the position of the body relative to the SBB (taken from an ephemeris).

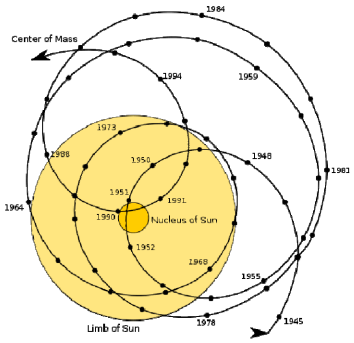


Figure 4.5: Centre of the sun relative to the SSB [7]

Barycentering The barycentre relative to the centre of mass of the sun, does not stay fixed. This is due to the movement of the planets, causing the centre of mass of the solar system to move. This adds time dependence to the barycentering

4.2.2. FUNCTIONAL ARCHITECTURE AND UNITS

The functional architecture of the orbit module is shown in a hierarchical structure with flow charts from the top level of the module: the Make Ephemeris and Update State functions; to their functionality in turn: Propagation of the ephemeris through the chosen solver and the transformation of the state from the chosen reference frame to the SSB.

TOP LEVEL

There are two main functions in the generation and propagation of the true state: The generation of the ephemeris object, and the propagation of the state within that object.

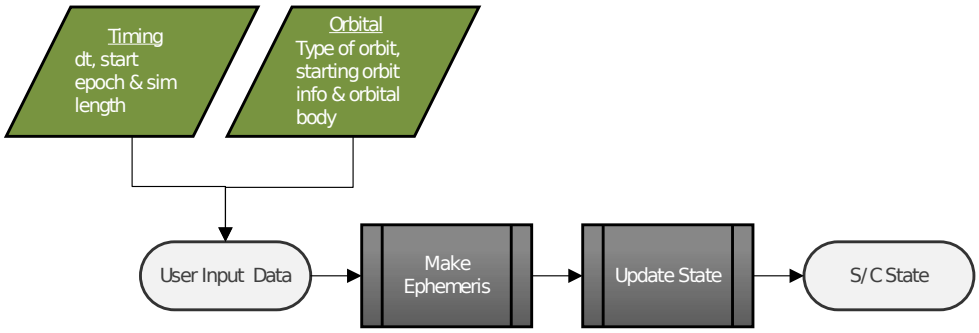


Figure 4.6: architectural overview of the simulation test bed

Make Ephemeris

The generation of the ephemeris is dependent on the type of information provided by the user - given as Keplerian elements and position and velocity (although also in principle an ephemeris or look-up table of true states). From this information, an ephemeris is generated.

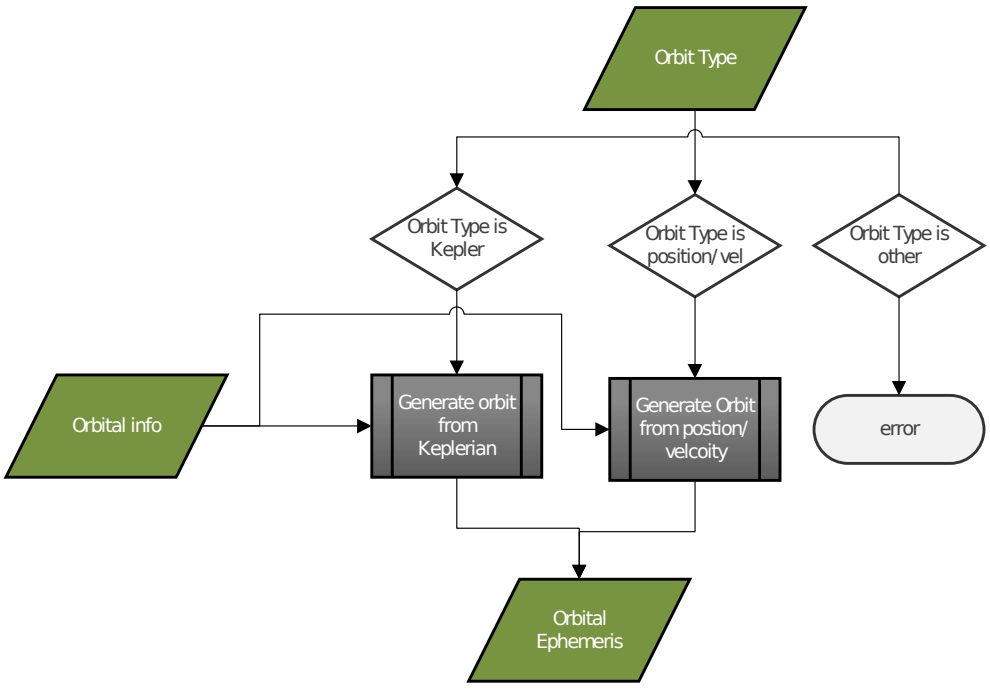


Figure 4.7: Generating an ephemeris

Update State

The update of the ephemeris object involves two main steps, but 2 additional optional steps may be asked for. The main steps are the update of the simulation time - through the Update epoch function, and the propagation the

true state, through Propagate Ephemeris. The two optional functions are the transformation of the true state in the orbital-body-centered reference frame to the SSB, and the addition of AWGN to the state - which was described in the sensor section.

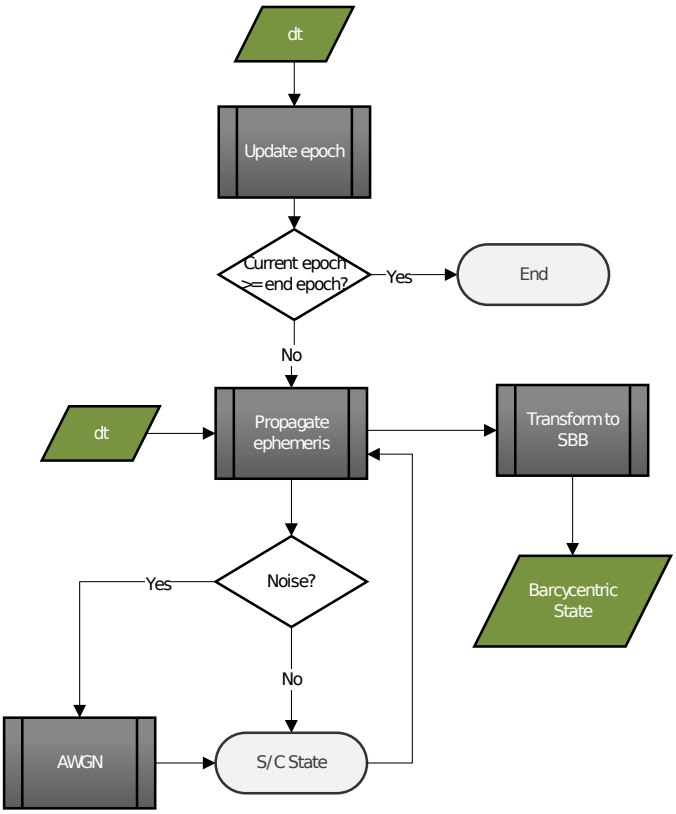


Figure 4.8: architectural overview of the simulation test bed

Propagate Ephemeris

The propagation of the state itself is dependent on the chosen solver. This solver creates the new state, based on the timestep.

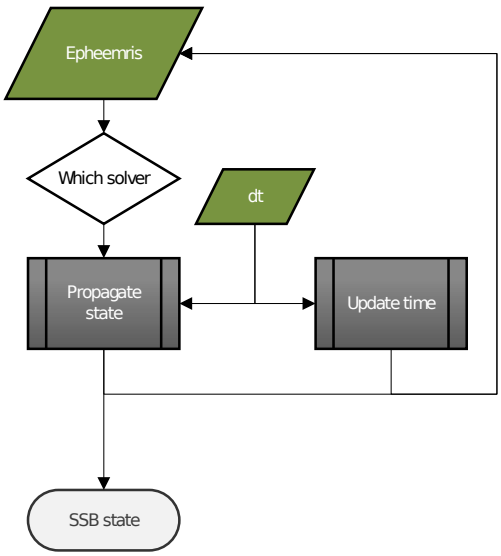


Figure 4.9: Propagation of the ephemeris based on the specific solver

Transform to SSB

The transformation of the state to the Barycentric reference frame involves using a solar system ephemeris, which (although not shown in the flowchart below) is checked to see if it exists, and if not, it is downloaded. This is then used to generate a position of the Barycenter relative to the position of the orbital body. This uses the current simulation time and from this, the planetary positions may be defined and the rotational elements required for the coordinate frame rotation, may be generated. This then allows for the translation and rotation to the Barycentric reference frame to be performed.

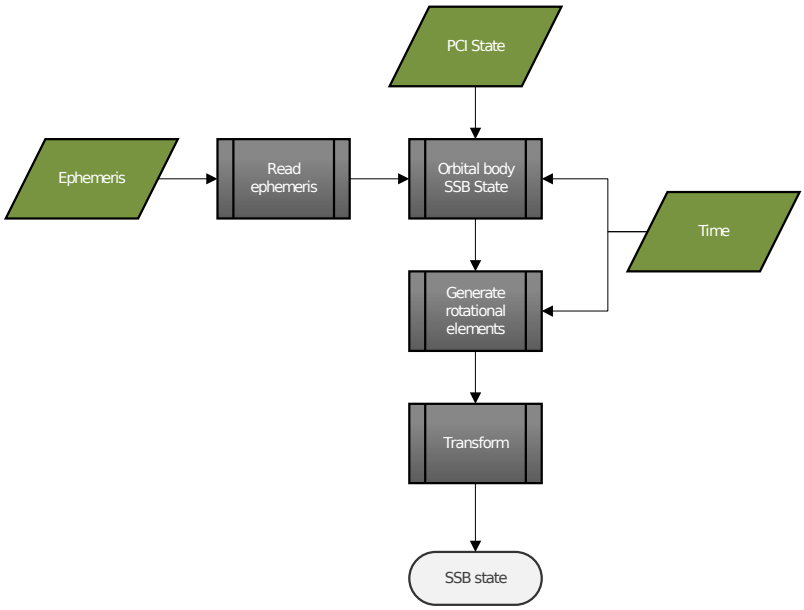


Figure 4.10: Barycentering software flowchart

4.2.3. VERIFICATION

To verify that the model is working as expected, it is first verified. As the model is currently only implementing a perturbation-free keplerian orbit, the propagators may be verified analytically. For a Keplerian orbit, the true anomaly θ may be related to the eccentric anomaly E (the angular position of the orbital body referenced to the center of the orbital ellipse) by[70]:

$$\sin(E) = \frac{r\sin(\theta)}{a\sqrt{1-e^2}} \tag{4.6}$$

$$\cos(E) = \frac{r\cos(\theta)}{a} + e \tag{4.7}$$

where e is the orbital eccentricity. It may be shown that

$$E - e\sin(E) = \sqrt{\frac{\mu}{a^3}}(t-\tau) \tag{4.8}$$

where t is the current time, and τ is the time since the last pericenter passage. The right hand term may be defined as the mean anomaly M:

$$E - e\sin(E) = M \tag{4.9}$$

which is the Kepler equation. The mean anomaly M, has a constant angular rate, defined by the orbital period through Kepler's second law. However the right hand side of equation 4.9 is transcendental and therefore requires minimisation. This must be done numerically, however to verify the propagators, the cartesian output can be converted to Keplerian elements and compared to the analytically propagated mean anomaly.

The cartesian state ($\mathbf{X} = [\mathbf{x},\mathbf{v}]^T$) may be transposed to the semi major axis by:

$$a = \frac{\mu}{2}\left(\frac{\mu}{|\mathbf{x}|} - \frac{|\mathbf{v}|^2}{2}\right)^{-1} \tag{4.10}$$

and for completeness, the eccentricity may be written as:

$$e = \left| \frac{1}{\mu} \left[(|\mathbf{v}|^2 - \frac{\mu}{r})\mathbf{x} - (\mathbf{x} \cdot \mathbf{v})\mathbf{v} \right] \right| \tag{4.11}$$

To do a reasonable comparison, two different orbits are chosen:

Table 4.4: The chosen orbits. The central body chosen is Earth. The first is a (near) circular orbit and the second is highly eccentric and inclined.

a [km]	e [-]	inc [deg]
8000	0.0	0.0
8000	0.75	65

These orbits will show if there is a difference with respect to integration time step between a near circular orbit, and an eccentric, inclined orbit.

Figure 4.14 shows the errors in M and eccentricity for the different solvers in SAT-ANS . For solutions based on the Newtonian gravitational differential equation, the error increases with time step, as expected. However for the analytical solutions, the error decreases with timestep. This is also expected, as if small numerical differences are assumed between the two analytical methods, then the larger the number the evaluations, the larger the error propagation will be. However, for the Cowell (the numerical force-based propagator) the relative errors compared to the analytically derived solution are on average lower than 10^{-10} rad respectively for a simulation time of over two days. This error is considered acceptable for use in the model. The Kepler propagator has quite variable performance and so will be omitted from the model. SAT-F-01.01 is considered validated.

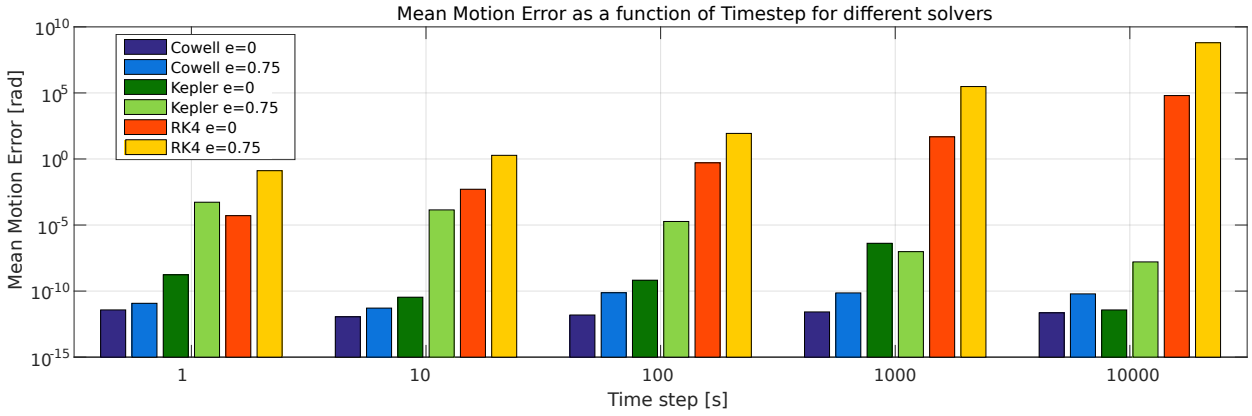


Figure 4.11: Mean motion errors for circular and eccentric, inclined orbit over 2 days of integration

Note that the Euler method, although also an option in the Navigation Module has not been included here, as the trends are similar to those seen by the RK4, with larger absolute errors

4.3. NAVIGATION MODULE

The navigation module is a difficult system to generalise, as the specific navigation method relies on the implemented filter. However, based on the assumptions that any filter will have at least a *predict* component, where a dynamical model is used to propagate the estimated state, and an *update* component, where observations are weighted to form a state estimate, the module may be generalised and the integration of sensors may be done according to the Centralised Integration architecture defined in the previous chapter.

4.3.1. WORKING METHOD

For the implemented navigation filter, there are two main steps as described in the previous section: Predict and Update. The predict step uses a dynamics propagator.

DYNAMICS

The following differential equation has been implemented for the dynamics:

$$\frac{d\mathbf{v}}{dt}(r) = -\frac{\mu \mathbf{r}}{r^3}$$

Which is solved discretely via the Euler method:

$$\begin{aligned}
x_{t+1} &= x_t + v_{x|t}dt \\
y_{t+1} &= y_t + v_{y|t}dt \\
z_{t+1} &= z_t + v_{z|t}dt \\
v_{x|t+1} &= v_{x|t} - \frac{\mu x_{t+1}}{({x_{t+1}}^2+{y_{t+1}}^2+{z_{t+1}}^2)^{3/2}}dt \\
v_{y|t+1} &= v_{y|t} - \frac{\mu y_{t+1}}{({x_{t+1}}^2+{y_{t+1}}^2+{z_{t+1}}^2)^{3/2}}dt \\
v_{z|t+1} &= v_{z|t} - \frac{\mu z_{t+1}}{({x_{t+1}}^2+{y_{t+1}}^2+{z_{t+1}}^2)^{3/2}}dt
\end{aligned}$$

Note, that the above equation has been implemented in the orbital verification and is considered acceptable for use in the dynamical model.

As was shown in the 4.2.3 with the RK4 solver, the Euler method has similar process errors (based on timestep) compared to the analytical solver, which motivate the need for a Kalaman filter

PROCESS NOISE

The model assumes additive white Gaussian noise in both the measurements and the process. As the process noise matrix definition is important for both the filter stability and tuning, its matrix will be derived. The process noise can be defined as:

$$E\{ww^T\} = \mathbf{Q} \tag{4.12}$$

The Kalman filter defined in the previous chapter has assumed a discrete dynamics system. This means that the covariance must be discretised:

$$\mathbf{Q}_k = \int_{t_k}^{t_{k+1}} \frac{d\mathbf{f}}{d\mathbf{x}}(t_{k+1},\tau)\mathbf{Q}(\tau)\frac{d\mathbf{f}}{d\mathbf{x}}^T(t_{k+1},\tau)d\tau \tag{4.13}$$

Where

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Phi & 0 & 0 \\ 0 & 0 & 0 & 0 & \Phi & 0 \\ 0 & 0 & 0 & 0 & 0 & \Phi \end{bmatrix} \tag{4.14}$$

with Φ representing the white noise spectrum.

Based on the integral relationship, the following process noise covariance is made:

$$Q = \begin{bmatrix} \frac{dt^3}{3} & 0 & 0 & \frac{dt^2}{2} & 0 & 0 \\ 0 & \frac{dt^3}{3} & 0 & 0 & \frac{dt^2}{2} & 0 \\ 0 & 0 & \frac{dt^3}{3} & 0 & 0 & \frac{dt^2}{2} \\ \frac{dt^2}{2} & 0 & 0 & dt & 0 & 0 \\ 0 & \frac{dt^2}{2} & 0 & 0 & dt & 0 \\ 0 & 0 & \frac{dt^2}{2} & 0 & 0 & dt \end{bmatrix} \Phi \tag{4.15}$$

Note, that the covariance is assumed to be time-independent. That is, the influence of the noise is additive Gaussian. This is a simplification, but could be made time-dependent in the future if required.

4.3.2. FUNCTIONAL ARCHITECTURE AND UNITS

The functional architecture of the Navigation module is may be represented by the predict and update state, mentioned in the introduction, in addition to the formatting of the sensor measurements such the the filter is able to use them. This involves sorting the measurement data such that the state covariance may be estimated and used for the Kalman gain. Further the collection of additional functions such as the calculation of the residuals is also done. This allows a standard data format to be generated and used in the filter.

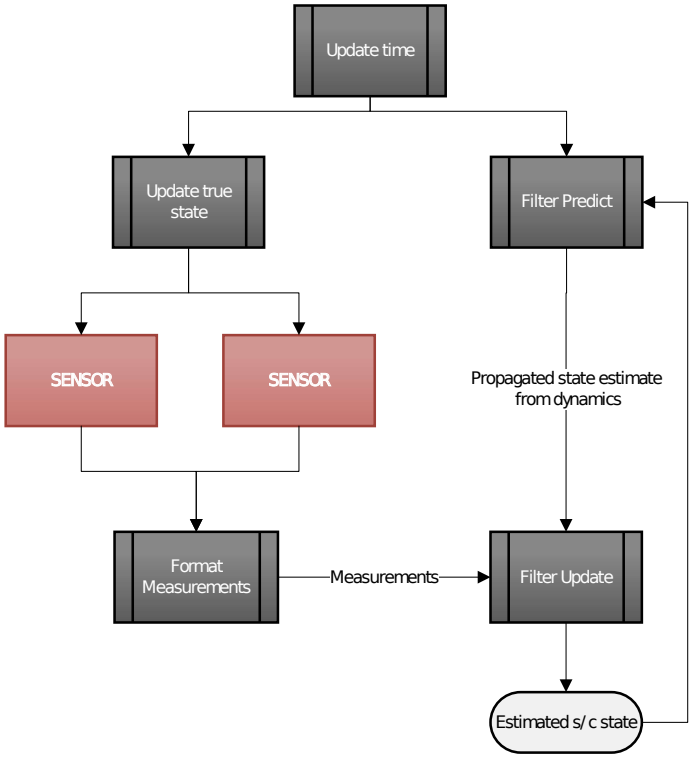


Figure 4.12: Architectural overview of the navigation module with sensors and filter for reference

Format Observations

The formatting of the observations is important for the communication between the sensor and navigation module. As the availability of the observations may be non-regular (subject to failures/non-observations of the sensors) , the formatting is done on the navigation-side.

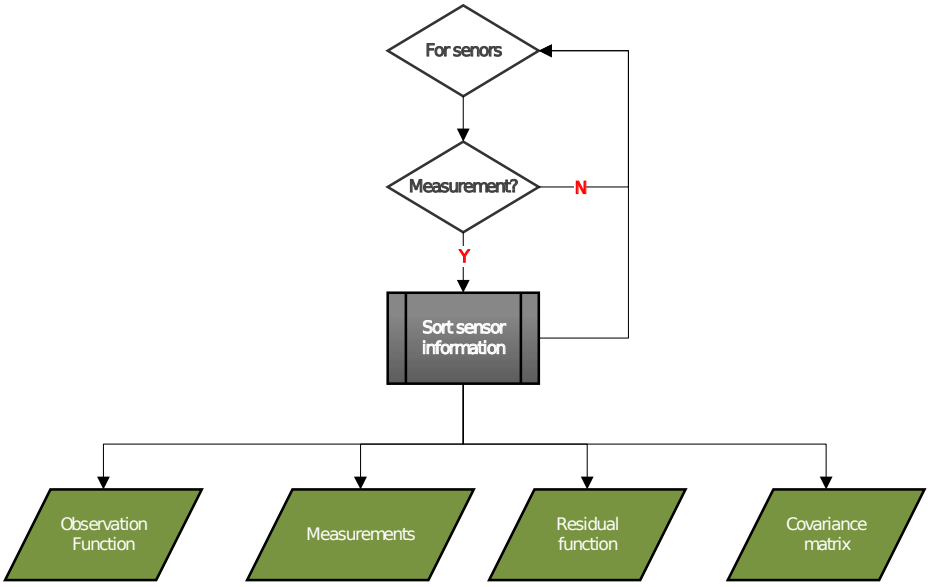
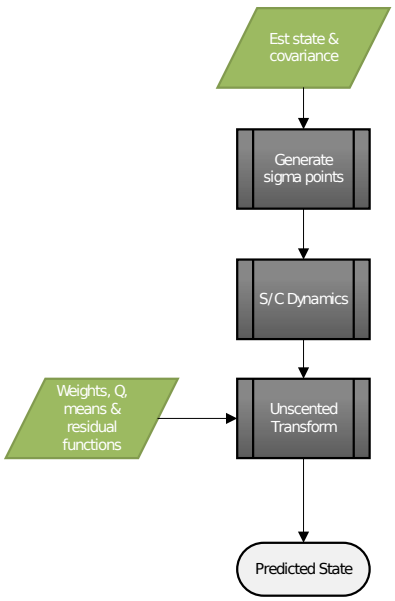


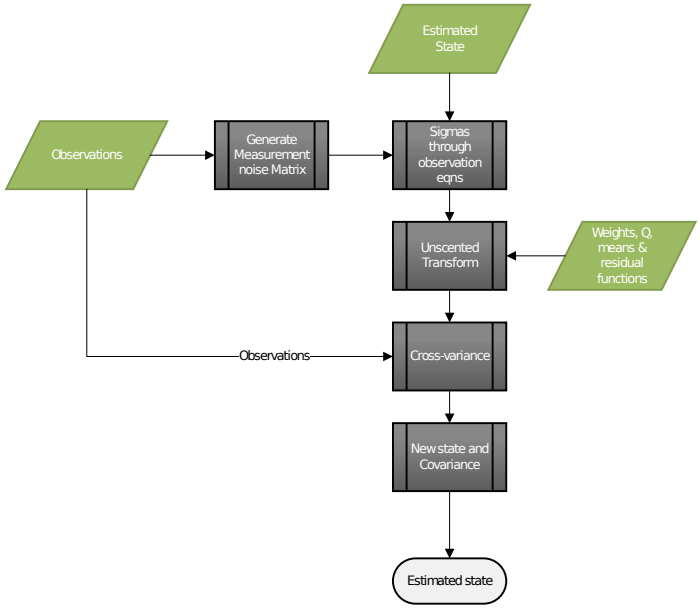
Figure 4.13: Formatting the sensor observations into their constituent components - the observation equations, the measurements, the residual/mean functions and the contribution to the covariance matrix

FILTER FUNCTIONS

The predict and update functions are shown below. Note that the formatted measurement data is required both for the generation of the measurement noise matrix, and the cross-variance of with the prediction.



(a) UKF predict phase



(b) UKF Update phase

Figure 4.14: Different solvers for first orbit (a = 8000 km, e = 0 and i = 0°) The dashed lines refer to the maximum root square error from that solver, and the solid line is the root mean square error

Unscented Transform

For the unscented transform, the type of measurement is iterated over. This is required as different sensor types may have different methods for the calculation of the mean and residual, as mentioned previously. Based on the formatting of the measurement data, these functions can be iterated over. R matrix in this case is the observation noise covariance matrix.

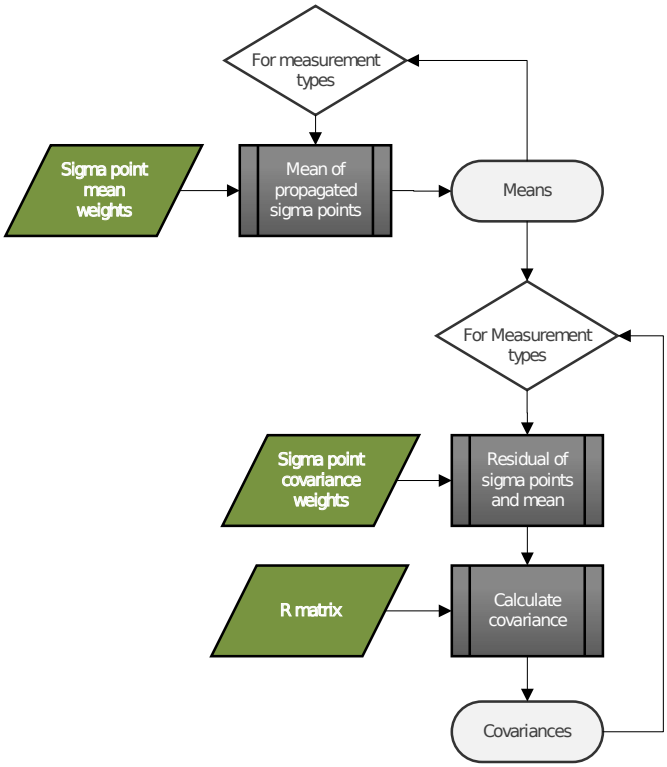


Figure 4.15: Unscented transform: the sigma points which have been propagated through the relevant dynamic/observational equations are then weighted, from which a mean is calculated. This mean is then used to determine the covariance through the weighted residuals of the sigma points and the mean calculated previously.

4.3.3. VERIFICATION

For the navigation module, the UKF was first implemented, along the lines of the algorithm described in the previous section. Once done, the UKF is tested using a non-linear toy problem. It is that of an object falling to Earth under gravity with drag to add the non-linearity to the process. Additionally, there is an observer which is detecting range, altitude (in angular terms) and azimuth. The state is defined as:

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{4.16}$$

and the dynamics driving the movement of the object are discretely defined as:

$$\dot{x}_1(t) = x_3(t) \tag{4.17}$$
$$\dot{x}_2(t) = x_4(t) \tag{4.18}$$
$$\dot{x}_3(t) = D(t)x_3(t) + G(t)x_1(t) + w_1(t) \tag{4.19}$$
$$\dot{x}_4(t) = D(t)x_4(t) + G(t)x_2(t) + w_2(t) \tag{4.20}$$
$$\dot{x}_5(t) = w_3(t) \tag{4.21}$$

Table 4.5: Constants used for the verification of the UKF

Constant	Value
β_0	-0.59783
H_0	13.406
GM_0	3.986e5
R_0	6374

Where x_5 refers to the ballistic coefficient of the falling body. w_n is the process noise, D(t) is the drag, G(t) is the gravitation force, and R and V are the normalsed position and velocity. They are defined as:

$$D(t) = \beta(t) \exp\left(\frac{R_0 - R(t)}{H_0}\right) V_0 \tag{4.22}$$

$$G(t) = -\frac{GM_0}{R^3(t)} \tag{4.23}$$

$$\beta(t) = \beta_0 exp[x_5(t)] \tag{4.24}$$

$$R(t) = \sqrt{x_1^2(t) + x_2^2(t)} \tag{4.25}$$

$$V(t) = \sqrt{x_3^2(t) + x_4^2(t)} \tag{4.26}$$

The following were used as the values of the constants:

with initial conditions:

$$\mathbf{x} = \begin{Bmatrix} 6500.4\text{km} \\ 349.14\text{km} \\ -1.8093\text{km}/s \\ -6.7967\text{km}/s \\ 0.6932 \end{Bmatrix}$$

and process noise:

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2.4064 \times 10^{-5} & 0 & 0 \\ 0 & 0 & 0 & 2.4064 \times 10^{-5} & 0 \\ 0 & 0 & 0 & 0 & 10^{-6} \end{pmatrix}$$

For sensor input, a radar is used which provides a range and angular bearing of the re-entry object with measurement equations:

$$r = \sqrt{(x_1 - s_x)^2 + (x_2 - s_y)^2} + q_1 \tag{4.27}$$

$$\theta = \tan^{-1}\left(\frac{x_2 - s_y}{x_1 - s_x}\right) + q_2 \tag{4.28}$$

Where q refers to the measurement noise which are zero-mean Gaussian distributions with standard deviations of $\sigma_r = 10^{-3}$ km and $\sigma_\theta = 0.17$ mrad.

RESULTS

For the validation of SAT-F-01.02, the measurement outputs were replicated with the verified model, and there was a mean error of XX over 100 MC iterations. Although the validation approach was to be an inspection of the observation equations, it is considered validated.

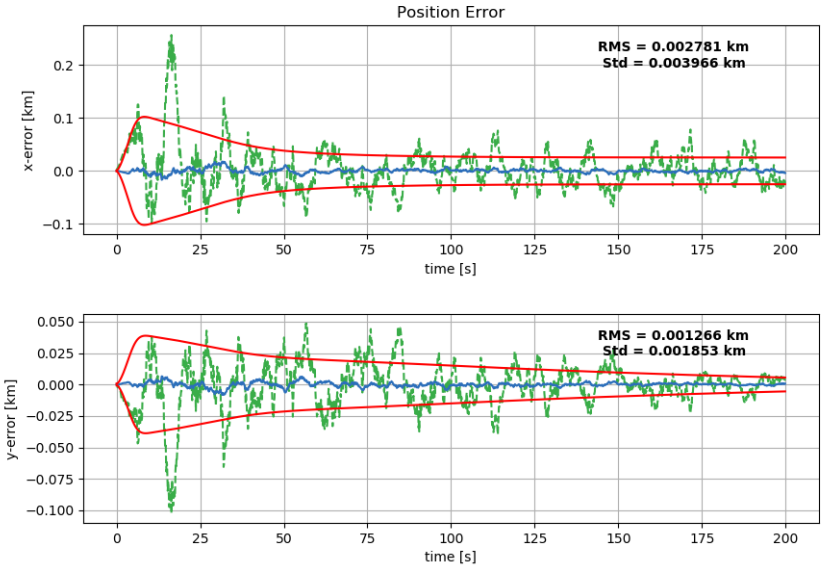


Figure 4.16: Mean position error over the monte carlo simulation as a function of time for the falling body problem. Faded lines represent the position based only on measurements. The red line is the 1-sigma error, and the dashed green line is the error for a single iteration.

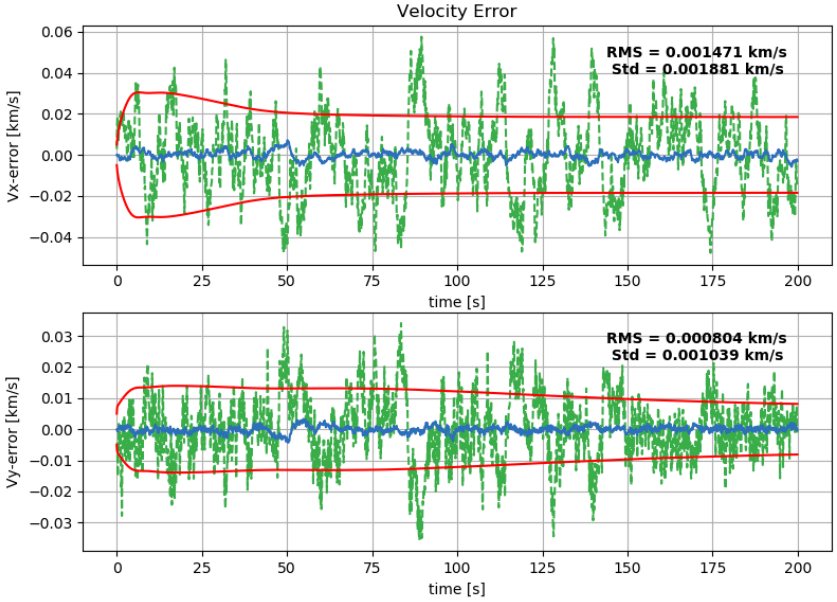


Figure 4.17: Mean velocity error over the monte carlo simulation as a function of time for the falling body problem. The red lines represent the 1-sigma filter-estimated error. The green dashed line is a single iteration of the model.

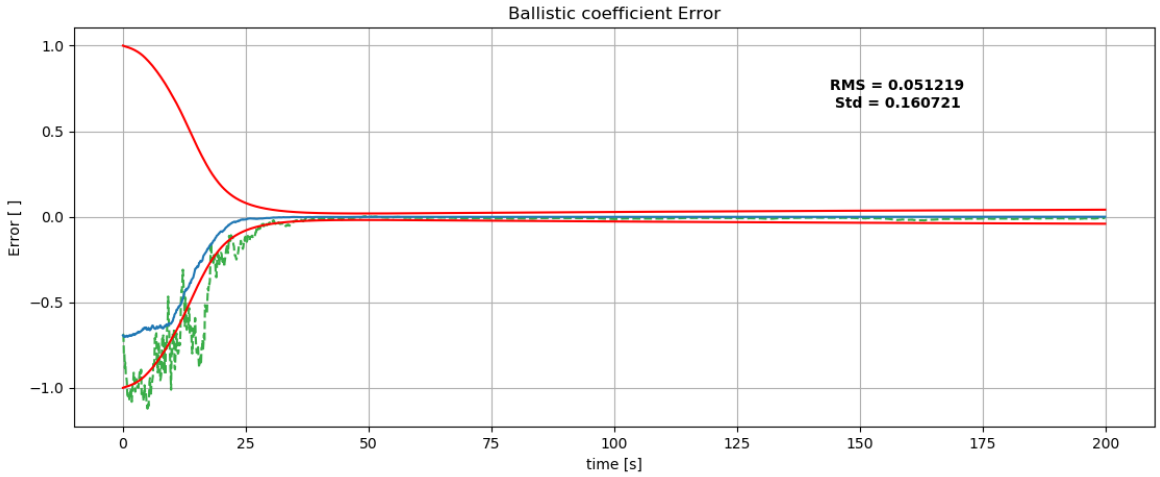


Figure 4.18: Mean ballistic coefficient error over the Monte Carlo simulation, as a function of time for the falling body problem. The red lines represent the 1-sigma filter-estimated error and the green dashed line is a single iteration error.

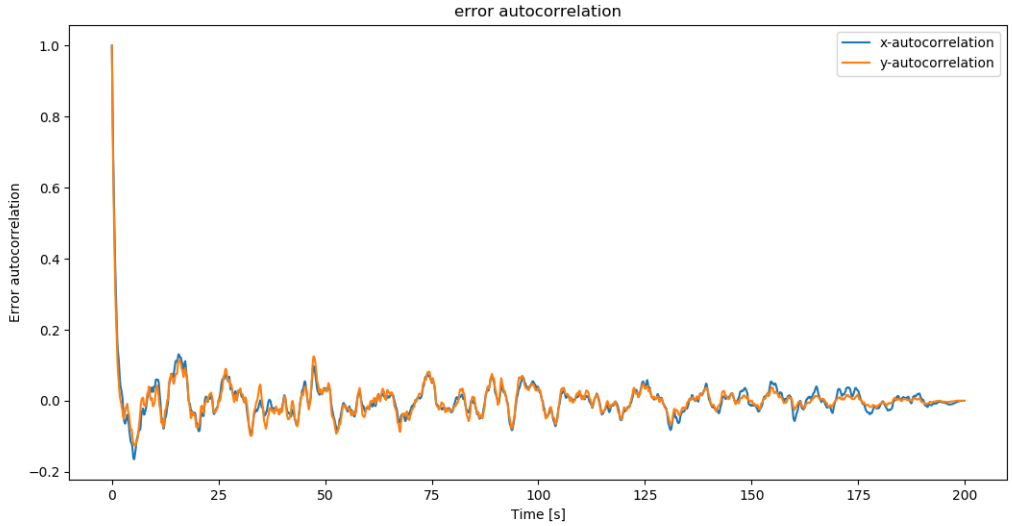


Figure 4.19: Autocorrelation of the x and y position errors

The results show that the model is capable of providing a stable state estimate, which is better than the measurements or state equation alone. The verification of the filter may be seen in the estimation of the state covariance, from which the 1-sigma lines are plotted on the figures. For an accurate system, 99.7% of the points should be bounded by 3 standard deviations. Over the course of a 100-run Monte Carlo analysis, the state estimation error was found to tend to towards the model's estimation of the error from the covariance, as shown in table 4.6.

Table 4.6: Falling body problem verification of the UKF containing the RMS and the percentage of points bounded by the model's 3-sigma error estimation.

Variable	Points bounded by 3-sigma [%]	RMS
Position	99.1	0.00202 km
Velocity	99.8	0.01140 km/s
Ballistic Coeff	99.9	0.0520 kg/m^2

Based on the above, in addition to the mean of the autocorrelation of the errors (0.00360), showing no visual correlation - the UKF is considered verified. This, in turn validates SAT-F-01.03.

4.4. MODEL INTEGRATION

With the main components of the model verified, they can then be combined to form the tool.

4.4.1. SOFTWARE ARCHITECTURE

The general overview of the software architecture is shown in figure 4.20. It can be generally divided into the four sections developed: Orbital - containing the Propagate Ephemeris and Add Noise functions; Sensor Models module, containing the libraries, Sensor Observe and Add Noise functions; and the navigation module with the Filter Predict and Filter Update functions. The Analysis module is shown as a separate function.

4.4.2. TIMING

For the different components with testing only a single 'global' time was considered. However, as it is unreasonable to assume that the different sensors will have the same update rate, additional clocks have been created to allow for asynchronous updating of the filter. This however adds complexity to the filter implementation for general integrated navigation.

4.4.3. MULTI-SENSOR NAVIGATION

As was defined in the previous chapter, a loosely-integrated CEN approach was chosen for this tool. With non-synchronous updating of the sensors for the filter however, the UKF requires dynamically defined sigma points to account for the changing number of available measurements. Furthermore, the relevant

4.4.4. TESTING

No externally verified orbital scenarios could be found which use an angle sensor and a radial velocity sensor for navigation sensors. However, the filter performance can still be analysed in this case to see if an improvement is made when additional sensors are used.

The following orbit was used to test the system. This was chosen as all components of the state change in time, thereby displaying the effectiveness of the navigation combination:

Table 4.7: Orbital parameters used for the testing of the integrated model

Keplerian Element	Value
Semi major axis	7136.6 km
Eccentricity	0.3
Inclination	90°
Right ascension of the ascending node	175°
Argument of perigee	90°
True Anomaly	178°
white noise variance	$1 \times 10^{-5}\, km/s$

With timing parameters

Table 4.8: Test simulation timing parameters

Parameter	Value
Simulation length	2.314 days
Integrator time step	10 s
Filter time step	10 s

In addition to this, the initial error in the state was taken from a normal distribution with standard deviation of 10 km for position and 0.1 km/s for velocity. Finally, the sensor parameters were set to:

Table 4.9: Sensor parameters for the test case. Beacons are defined as stationary in that they remain fixed for the duration of the simulation.

Sensor	Noise (sigma)	Update Rate	Beacons
radial velocity sensor	0.1 nm	10 s	[1,0,0], [0,0,1] stationary
Angle Sensor	4 microrad	10 s	[0.9, 0.31,0.31], [0.31, 0.9, 0.31], [0.31,0.31, 0.9] stationary at $10^9\, km$

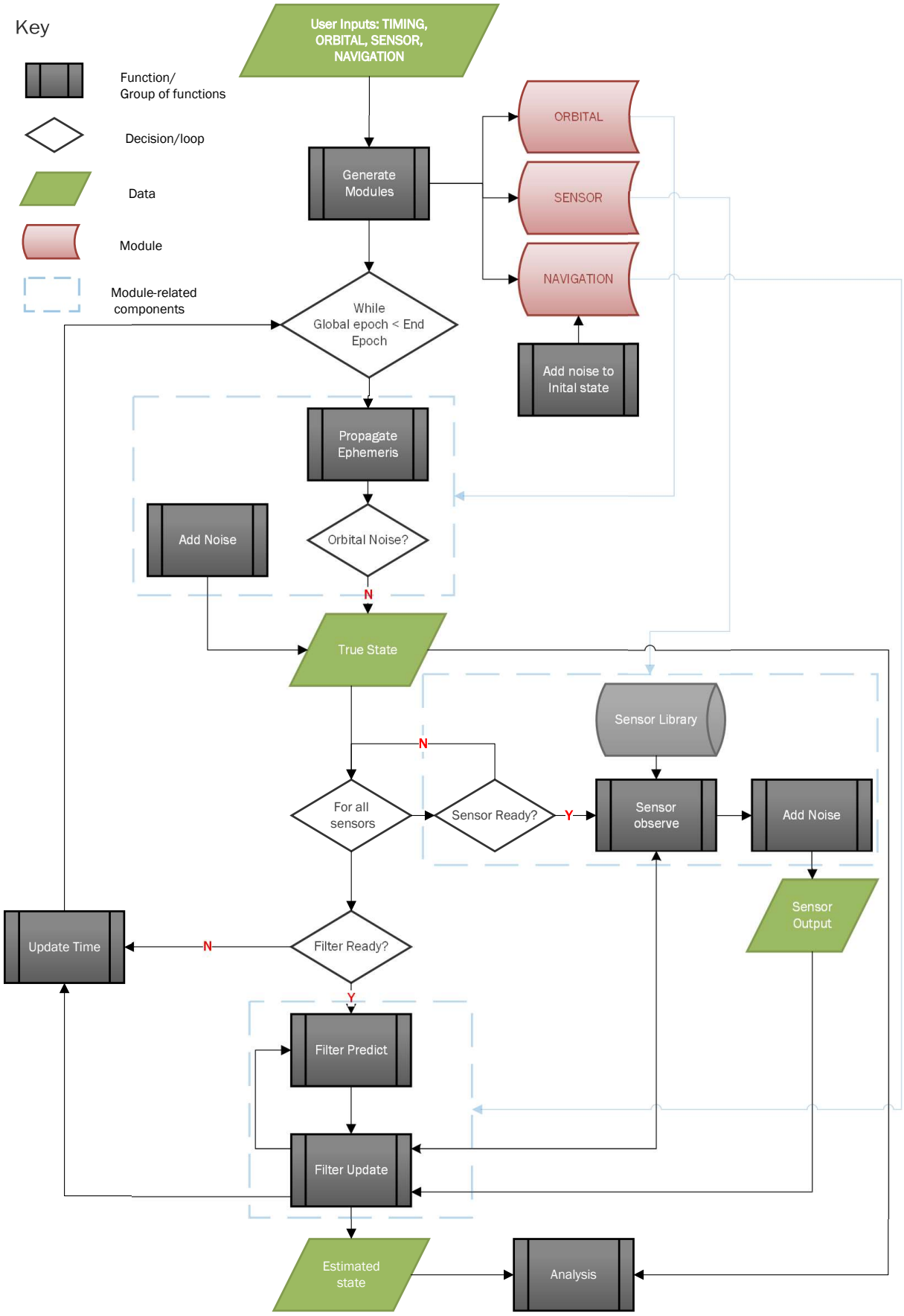


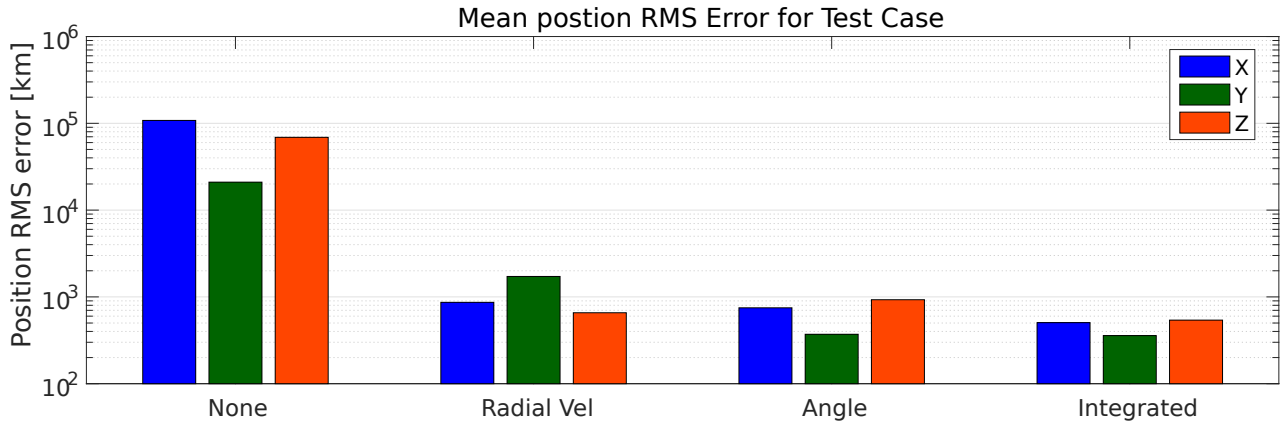
Figure 4.20: SAT-ANS software flowchart

And the results found are displayed in table 4.10

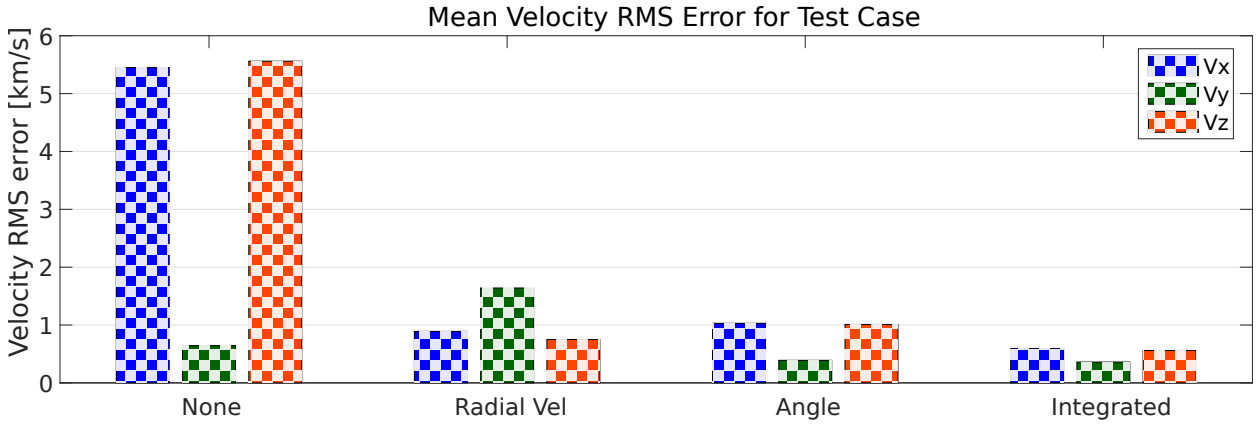
Table 4.10: Test scenarios with results

Number	Scenario	Position Error [km]	Position std [km]	Velocity Error [km/s]	Velocity std [km/s]
1	No sensor observation	x: 1.08×10^5 y: 2.1×10^4 z: 6.9×10^4	x: 5.7×10^4 y: 1.16×10^4 z: 3.44×10^4	x: 5.45 y: 0.64 z: 5.56	x: 1.09 y: 0.11 z:0.97
2	Only radial velocity sensor	x: 866 y: 1720 z: 657	x:179 y: 853 z: 137	x:0.893 y: 1.64 z: 0.753	x: 0.202 y: 0.78 z: 0.165
3	Only angle sensor	x: 749 y: 371 z: 928	x: 266 y:155 z: 360	x: 1.03 y:0.39 z: 1.01	x: 0.55 y: 0.29 z: 0.56
4	Integrated	x: 505 y: 358 z: 539	x: 64 y: 80 z: 65	x: 0.59 y: 0.36 z: 0.56	x: 0.17 y: 0.19 z: 0.14

As the table shows, there is a clear increase in accuracy using a sensor over not. However the increase in accuracy when using an additional sensor is somewhat marginal. It should be noted, that the update rate for both sensors was the same, and this could be investigated to see what performances can be achieved with different effective integration times with different sensors. Figures 4.22 to 4.25 show the RMS errors for position and velocity between using no sensors and integrated navigation. One clear point from the table and graph is the discrepancy between velocity error and position error in the no-sensor and angle-only sensor case. in the x-direction the RMS velocity error is lower than the RMS velocity error in the z-direction, but the position error is higher. At first glance this is strange, but if the standard deviation is considered - this is higher in both cases for the z-direction. The standard deviation may capture the variation and the oscillation of the system to some degree.



(a)



(b)

Figure 4.21: Test case mean RMS errors per dimension

The figures above show a visualisation of the table. Below are the examples over time of Monte Carlo analyses of the test case. The green lines are the true covariance as calculated over the simulation iterations. For the cases without this, please see appendix D.

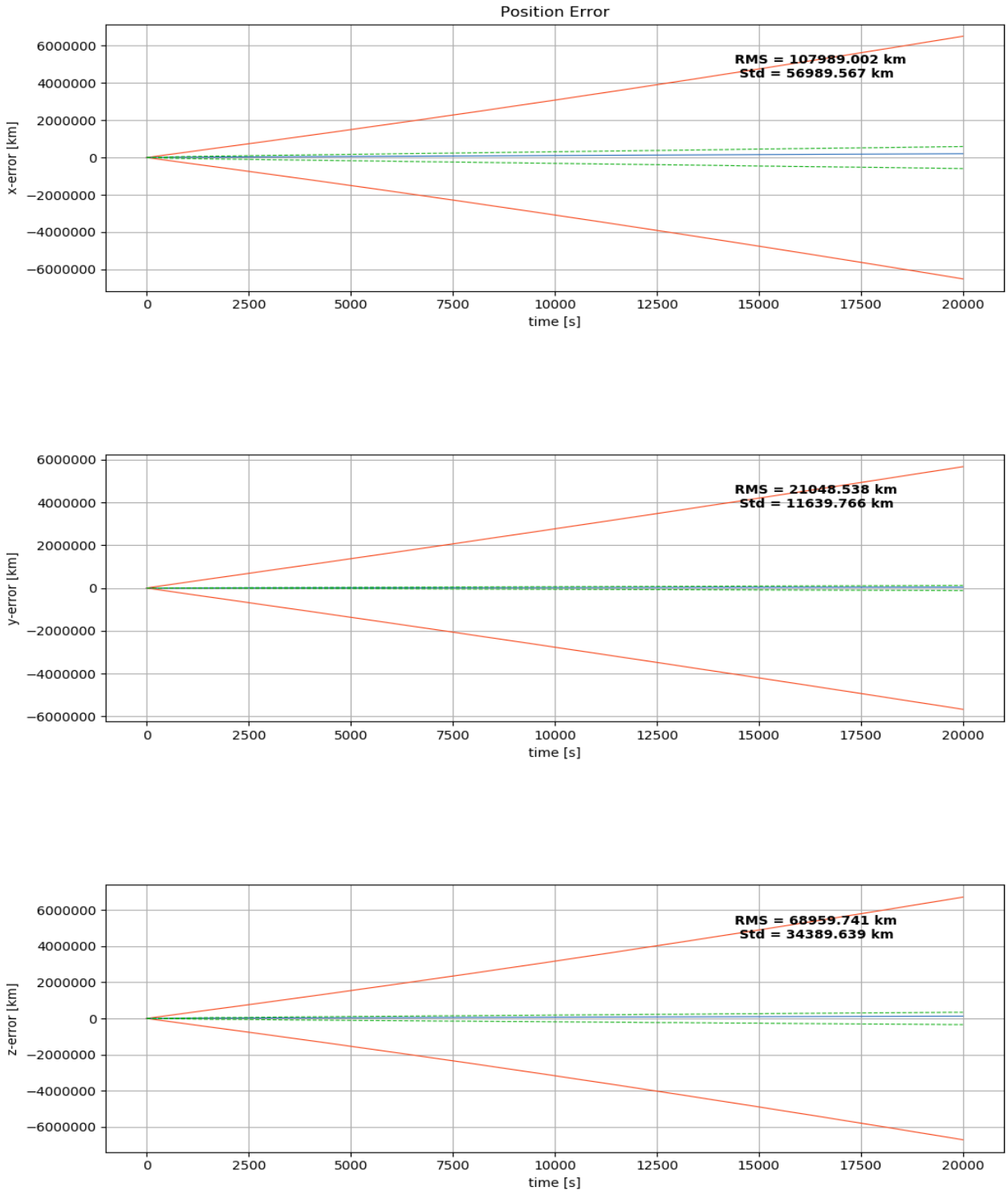


Figure 4.22: Position RMS errors with filter-estimated 3-sigma error for test scenario 1

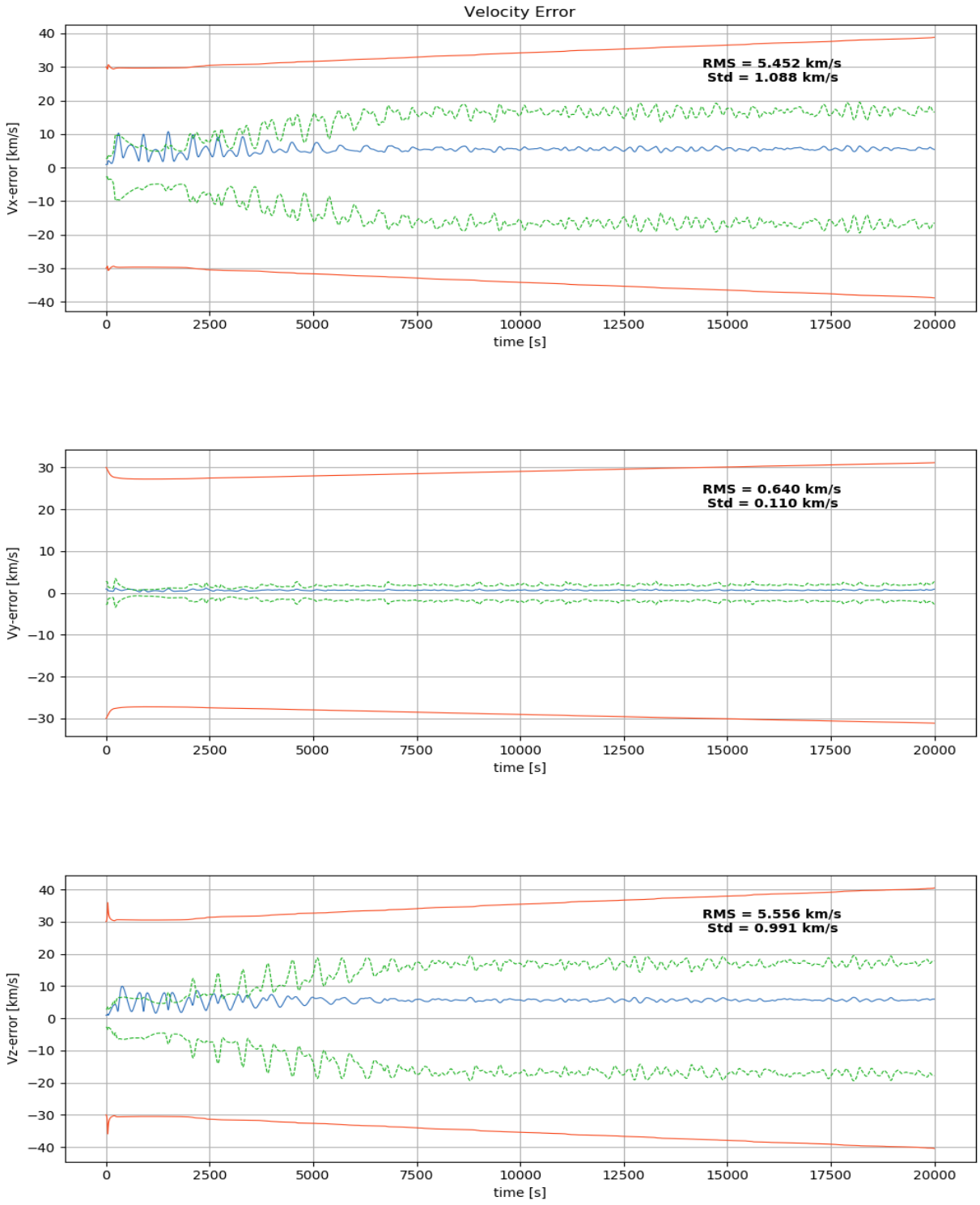


Figure 4.23: Velocity RMS errors with filter-estimated 3-sigma error for test scenario 1

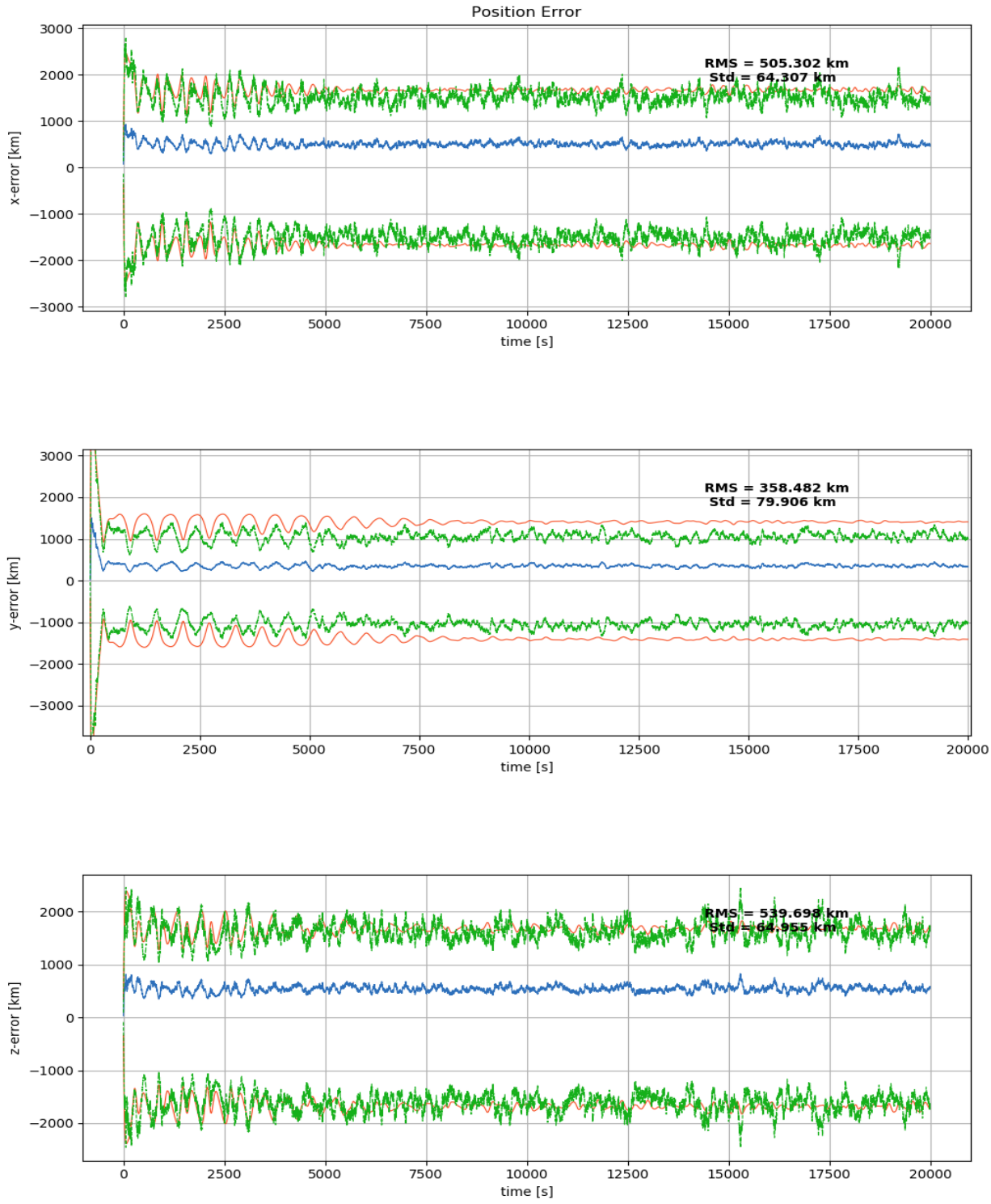


Figure 4.24: Position RMS errors with filter-estimated 3-sigma error for test scenario 4

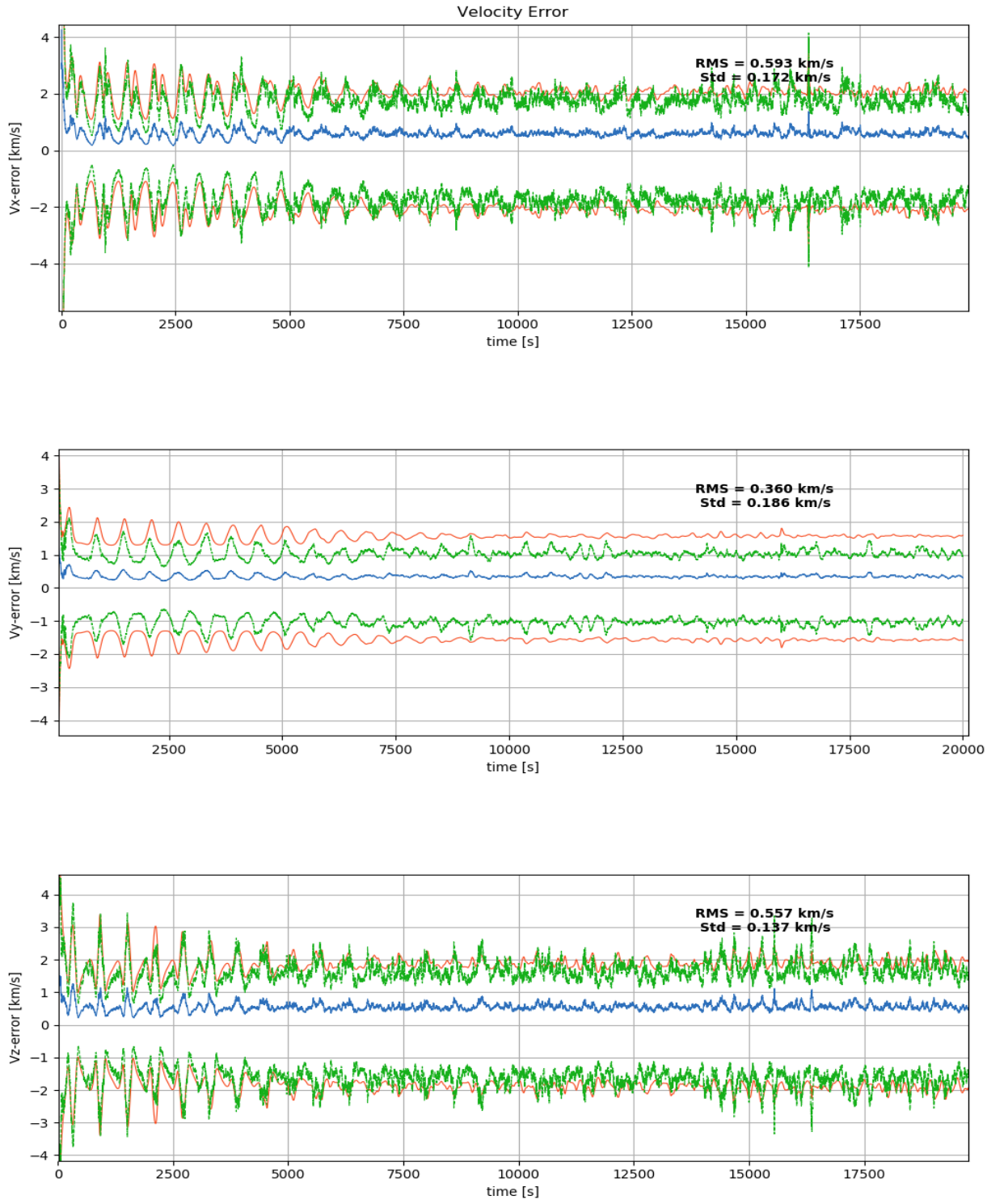


Figure 4.25: Velocity RMS errors with filter-estimated 3-sigma error for test scenario 4

Compared to the plots with no sensor observation, the integrated navigation seems to contain some oscillatory components. Based on this, a Fourier frequency analysis is performed, to see if there are any dominant frequency components in the errors.

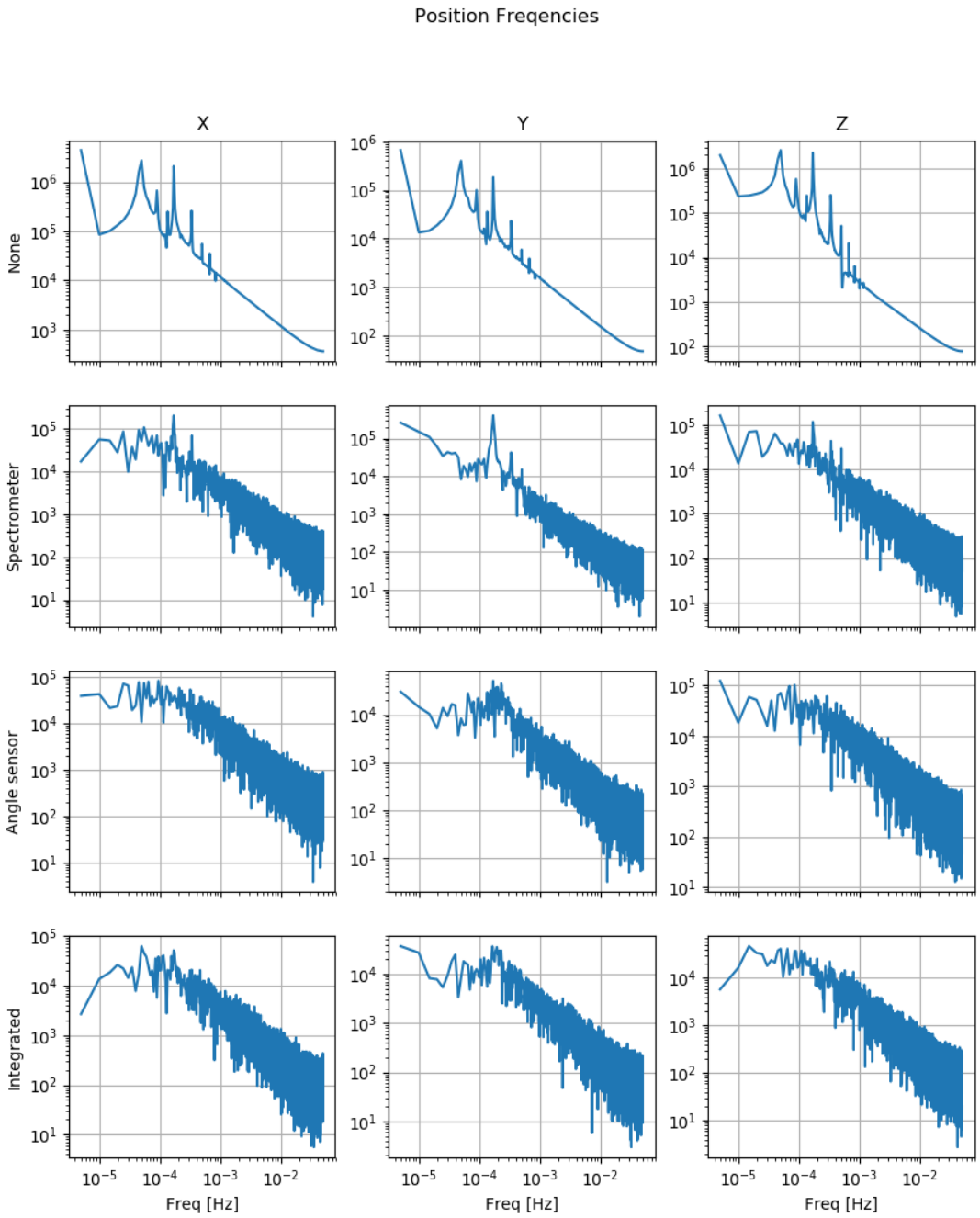


Figure 4.26: Fourier analysis of the position errors for the different test scenarios

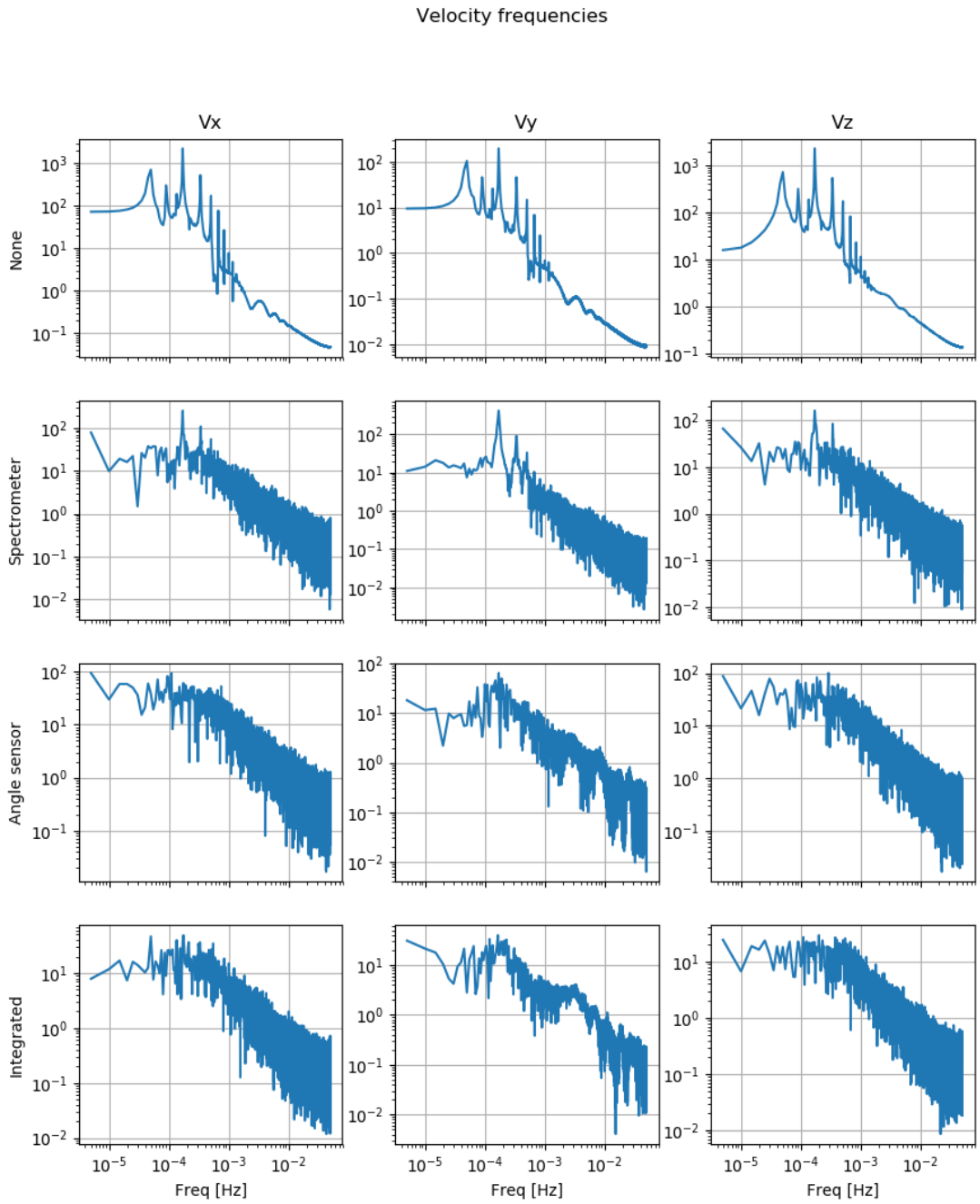


Figure 4.27: Fourier analysis of the velocity errors for the different test scenarios

The main frequencies seen in the Fourier series are at 1.67 $\times 10^{-4}$ Hz, which is the inverse of the orbital period, with harmonics seen. However there are additional frequencies present, not integer numbers of the orbital period. As the s/c is orbiting, the information contribution to the different state dimensions from stationary beacons is changing, which is thought to cause the oscillatory behaviour seen. The addition of the angle sensor removes the strong peak due to the period especially prominent in the Y-component (where there is no information-contributing beacon for the radial velocity sensor), which supports this.

To investigate this, a test is made where the beacons of the angle sensor are placed orthogonally (unit vectors of [1,0,0], [0,1,0] and [0,0,1]) and a circular orbit with zero inclination is made with the same semi major axis as the test done above. The expectation is that the filter estimated covariance which is oscillating in the figures above will, in the X and Y directions, track the true position.

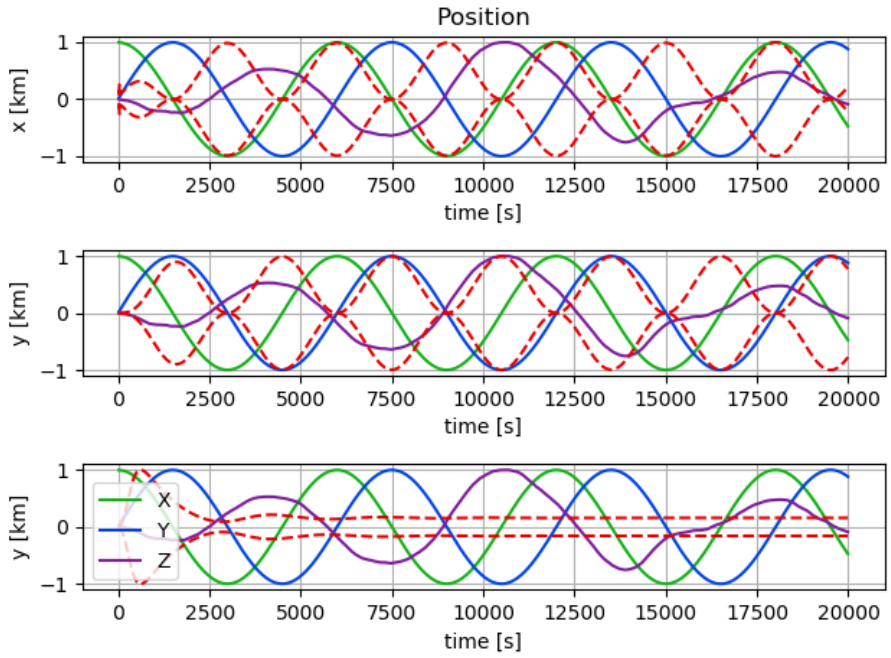


Figure 4.28: Normalised true positions and filter-estimated covariance for the circular orbit case

As figure 4.28 shows, the X and Y component covariances track the true positions, thereby showing the contribution of the beacons in those dimensions, as expected. Also as the orbit is in the X-Y plane, there is little contribution of information in the Z-direction so there are no oscillations seen. Note that it looks like the Z-covariance is much smaller than the X and Y, however the information in the figures is normalised so the absolute values contain no useful information.

This example shows that the oscillations seen in the test figures is likely to be mainly the positions of the navigation beacons and not due to instabilities within the filter.

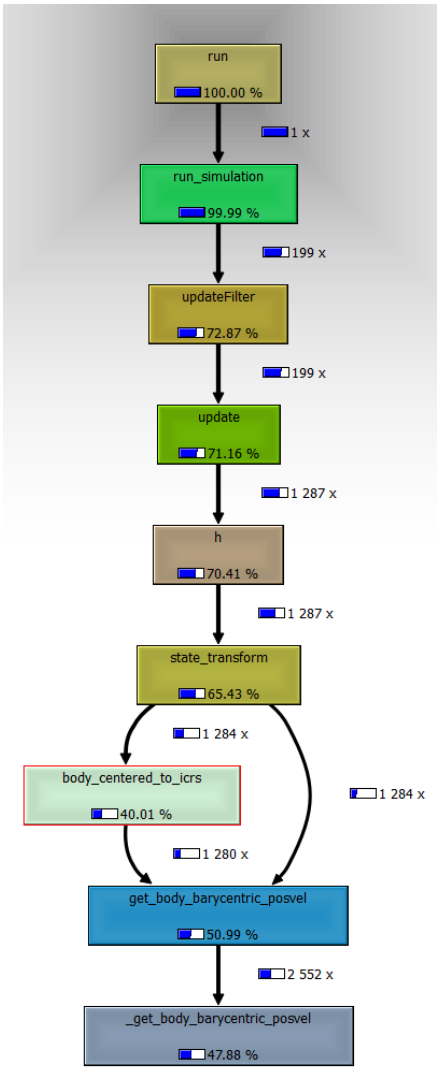
4.4.5. NAVIGATION SYSTEM PERFORMANCE

Requirement SAT-F-01.04 states that the system shall assess the navigation system quality. Table 4.10, with normalised components visualised by figure 4.21 allow the addition of sensors to define the navigation system quality for this specific sensor combination, navigation filter with tuning parameters and orbital environment. The single, normalised values for position and velocity give a reasonable metric for the relative performance of the navigation system, in different configurations. The specific navigation performance can then be assessed over the trajectory of interest though the error and perhaps more importantly the covariance of each state element, as shown in figures 4.22 to 4.25. As these aspects can assess the quality of the navigation system, SAT-F-01.04 is considered validated.

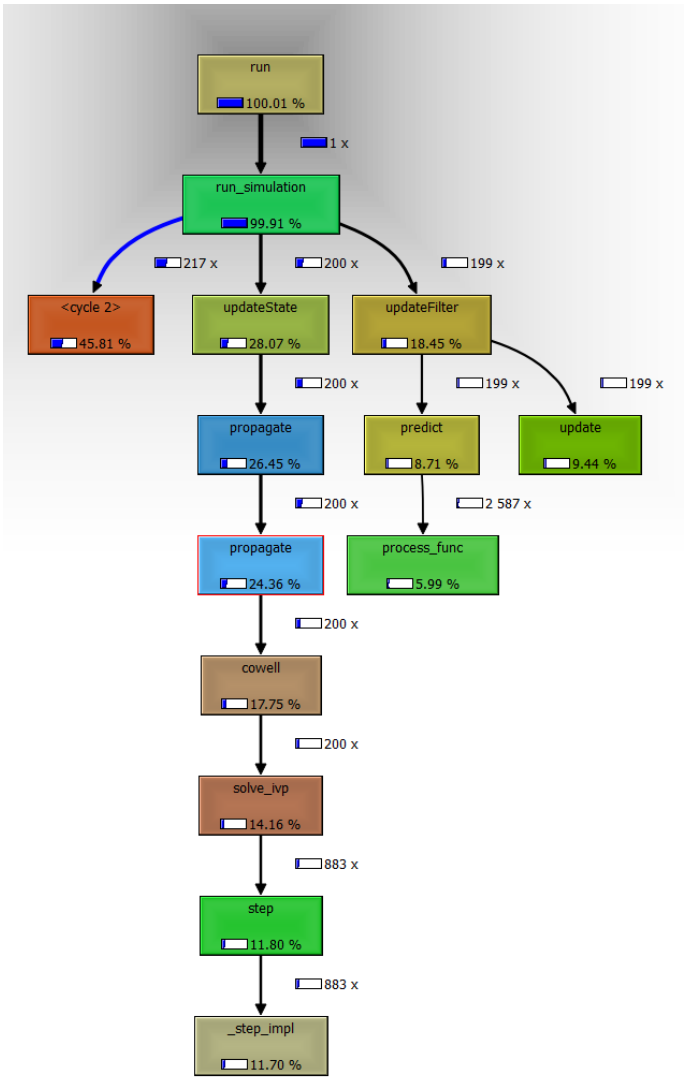
4.4.6. SOFTWARE PERFORMANCE

During various test cases, it was found that the iteration speed from epoch to epoch decreased over time and was independent of the system memory. This was investigated further and it was found that the implementation of the barycentering using an external ephemeris drastically decreased performance. Using cProfile to determine the time spent in individual functions and KCacheGrind to visualise the information. The impact could be seen. Two representative figures 4.29 are included to show this impact. The the number of software iterations per second including barycentering was found to be 15.47, and without it was found to be 141.83. For its use as a software tool, the efficiency has some impact. Although the optimisation of speed is not a focus of this work, a reduction of around 90% in operational performance due to the addition of barycentering is considered unacceptable. This is therefore modified to reduce the impact.

https://docs.python.org/3/library/profile.html
https://kcachegrind.github.io/html/Home.html
An iteration is here defined as the point at which the epoch is updated, to that same point.



(a) Relative impact of barycentering - 15.47 iterations/s



(b) Absence of barycentering - 141.83 iterations/s

Figure 4.29: The software performance impact of barycentering

This problem was somewhat mitigated by further customising the ephemeris usage and barycentering of the module, so that the solar system ephemeris is initialised once and referenced from there. This led to a performance increase of 570% over the previous barycentering to 88.2 iterations/s. Note that the impact of the testing software on iteration speed has not been investigated here - It is assumed to have negligable effect and constant between the different cases, but may need to be defined if software performance is fully investigated.

The barycentering remains a resource-intensive operation. Particularly within the Kalman filter where each sigma point state estimate will require barycentering. Further investigation of the optimisation here may be required.

4.5. ANALYSIS

The CRLB As stated in equation 4.5.1 has been implemented into SAT-ANS, however the results obtained were found to be inaccurate and further work in this area is deemed to be required.

4.5.1. IMPLEMENTATION OF ANALYTICAL CRLB

The equations relevant for the recursive CRLB are restated here for convenience:

$$\mathbf{C}_{k+1|k}^{-1} = \mathbf{K}_{k+1}^{k+1} - \mathbf{K}_{k+1}^{k+1,k}(\mathbf{K}_{k+1}^{k} + \mathbf{C}_{k|k}^{-1})^{-1})\mathbf{K}_{k+1}^{k,k+1} + \mathbf{L}_{k}^{k}$$

where

$$\mathbf{K}_{k+1}^{k} = [\nabla_{\mathbf{x}}\mathbf{f}_k(\mathbf{x}_k)]^T \mathbf{Q}_k^{-1} \nabla_{\mathbf{x}}\mathbf{f}_k(\mathbf{x}_k)$$
$$\mathbf{K}_{k+1}^{k,k+1} = -[\nabla_{\mathbf{x}}\mathbf{f}_k(\mathbf{x}_k)]^T \mathbf{Q}_k^{-1}$$
$$\mathbf{K}_{k+1}^{k+1} = \mathbf{Q}_k^{-1}$$
$$\mathbf{L}_k^k = [\nabla_{\mathbf{x}}\mathbf{h}_k(\mathbf{x}_k)]^T \mathbf{R}_k^{-1} \nabla_{\mathbf{x}}\mathbf{h}_k(\mathbf{x}_k)$$

The expectation operator has been omitted based on the assumptions made in the previous chapter.

Additionally, the covariance matrices both, measurement and process are assumed to be time independent as has been motivated in previous sections. This in principle could be altered depending on requirements, however the simplification is deemed reasonable as the dynamics are not changing in time, nor are sensor characteristics.

4.5.2. TEST CASES

Two test cases were made - in line with the test cases used for the UKF verification and the model integration.

RE-ENTRY PROBLEM

the Jacobian of the Re-Entry equations are given by:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} 1 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & dt & 0 \\ -(\frac{(xDv_x)}{R}H_0 + \frac{(3\mu x^2)}{R^5} - \frac{\mu}{R^3})dt & -(\frac{(yDv_x)}{R}H_0 + \frac{(3\mu xy)}{R^5})dt & (\frac{Dv_x^2}{V^2} + D)dt + 1 & \frac{Dv_x v_y}{V^2})dt & (Dv_x)dt \\ -(\frac{(xDv_y)}{R}H_0 + \frac{(3\mu xy)}{R^5})dt & -(\frac{(yDv_y)}{R}H_0 + \frac{(3\mu y^2)}{R^5} - \frac{\mu}{R^3})dt & \frac{Dv_x v_y}{V^2})dt & (\frac{Dv_y^2}{V^2} + D)dt + 1 & (Dv_y)dt \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

With the observation of the object range and elevation angle, the observation derivatives are:

$$\frac{d\mathbf{h}}{d\mathbf{x}} = \begin{bmatrix} \frac{x}{r} & \frac{y}{r} & 0 & 0 & 0 \\ -\frac{y-y_{obs}}{(y-y_{obs})^2+(x-x_{obs})^2} & \frac{x-x_{obs}}{(y-y_{obs})^2+(x-x_{obs})^2} & 0 & 0 & 0 \end{bmatrix}$$

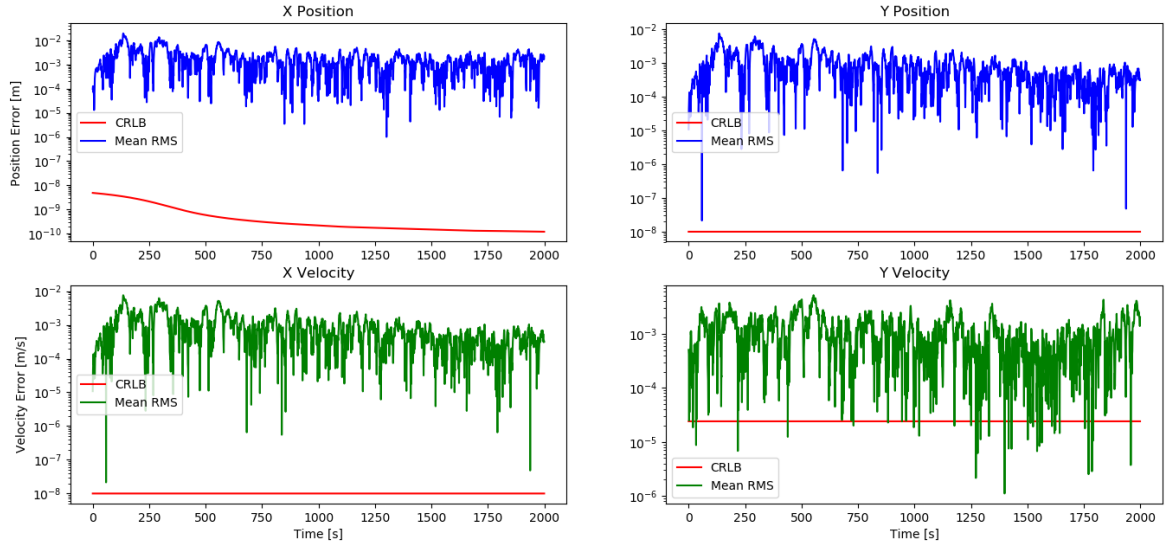


Figure 4.30: Implementation of the CRLB for the Re-Entry case

ORBITAL CASE

The Jacobian of the differential equation mentioned in YY with respect to the state is given as:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ (\frac{3\mu x^2}{r^5} - \frac{\mu}{r^3})dt & \frac{3\mu xy}{r^5}dt & \frac{3\mu xz}{r^5}dt & 1 & 0 & 0 \\ \frac{3\mu xy}{r^5}dt & (\frac{3\mu y^2}{r^5} - \frac{\mu}{r^3})dt & \frac{3\mu yz}{r^5}dt & 0 & 1 & 0 \\ \frac{3\mu xz}{r^5}dt & \frac{3\mu yz}{r^5}dt & (\frac{3\mu z^2}{r^5} - \frac{\mu}{r^3})dt & 0 & 0 & 1 \end{bmatrix}$$

The derivatives of the observation equations are defined as:

Angle sensor

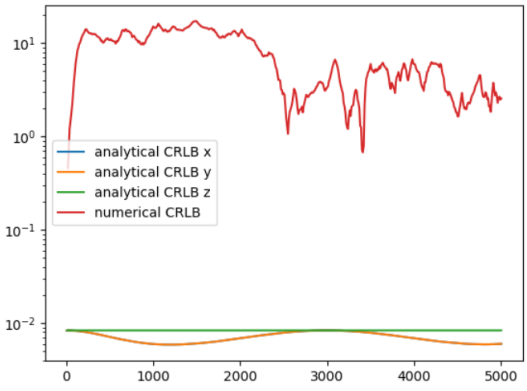
$$\frac{d\mathbf{h}}{d\mathbf{x}} = \begin{bmatrix} -\frac{y-y_{obs}}{(y-y_{obs})^2+(x-x_{obs})^2+(z-z_{obs})^2} & \frac{x-x_{obs}}{(y-y_{obs})^2+(x-x_{obs})^2+(z-z_{obs})^2} & 0 & 0 & 0 & 0 \\ \multicolumn{6}{c}{\mathbf{S}^T} \end{bmatrix}$$

where

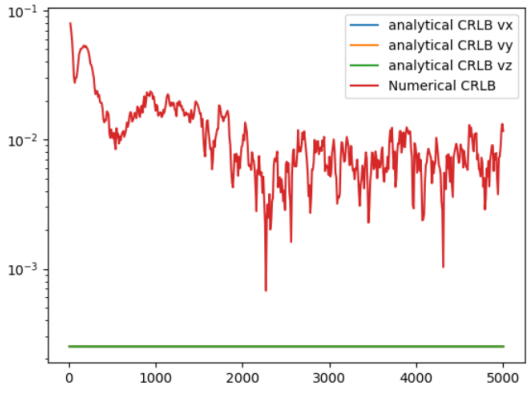
$$\mathbf{S} = \begin{bmatrix} 1/2\,\frac{z(2x-2x_{obs})}{\left((x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2\right)^{3/2}} \frac{1}{\sqrt{-\frac{z^2}{(x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2}+1}} \\ 1/2\,\frac{z(2y-2y_{obs})}{\left((x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2\right)^{3/2}} \frac{1}{\sqrt{-\frac{z^2}{(x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2}+1}} \\ -\left(\frac{1}{\sqrt{(x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2}} - 1/2\,\frac{z(2z-2z_{obs})}{\left((x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2\right)^{3/2}}\right) \frac{1}{\sqrt{-\frac{z^2}{(x-x_{obs})^2+(y-y_{obs})^2+(z-z_{obs})^2}+1}} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Radial velocity sensor

$$\frac{d\mathbf{h}}{d\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & \lambda_0 v_x b_x \left(1+\frac{\Delta v}{c}\right) & \lambda_0 v_y b_y \left(1+\frac{\Delta v}{c}\right)^2 & \lambda_0 v_z b_z \left(1+\frac{\Delta v}{c}\right)^2 \end{bmatrix}$$



(a) Position covariance as a function of time with normalised approximated numerical CRLB



(b) Velocity covariance as a function of time with normalised approximated numerical CRLB

Figure 4.31: Orbtial case CRLB

POSSIBLE CAUSE OF ERROR

In both the Re-Entry and the Orbital case, the analytically calculated CRLB and the numerically approximated - through assessing the statistical lower limit on covariance over the course of an arbitrary simulation, are different by

many orders of magnitude. Although the filter-sensor combination is not the optimum, and so perhaps not likely to meet the CRLB, it is thought that these would not differ by such a wide margin.

Although motivated, the assumption that the expectation value for the process and the measurement Jacobian is the true value, due to the additive Gaussian noise component tending to zero may not be valid for the system. Based on this, it may be that the system cannot be approximated though only one simulation run, and so making this approach invalid.

NEXT STEPS

There are some points for further investigation possible.

Initially, the CRLB based only on the measurement with negligible process noise could be investigated, to see the impact specific measurements have on the lower bound. Additionally, a more 'brute force' approach could be implemented which, at every epoch in the simulation, iterates over multiple versions of that epoch, defined by the noise characteristics of the system, from which the true PDF of the system calculated and from this the CRLB. This approach is most deemed most likely to achieve a result, as the approximations made for the posterior CRLB are no longer applied.

Further research may be required to find another method which is able to give a lower bound on the results.

Finally, no requirement has been set on specifying the method for attaining the performance of the lower bound, the definition of the CRLB for each sensor combination in a specific orbital scenario was considered an addition to the tool, and not part of its core functionality. Due to this and the time constraints of the project, the CRLB topic is left open for future work.

4.5.3. NON-BIASED APPROXIMATION OF STANDARD DEVIATION

To make sure that the calculation of the standard deviation is unbiased, the n-1 approximation is used. In other words [71]:

$$\sigma = \sqrt{\left(\frac{\Sigma(X-\bar{X})^2}{n-1}\right)} \tag{4.29}$$

4.5.4. MONTE CARLO ANALYSIS

With the CRLB having being shown not to work with the assumptions made, this motivates the use of Monte Carlo analyses to assess the performance of the navigation system. Although the CRLB will be ommitted from here on, this technique could be used to assess the true lower limit of the navigation sensor and dynamics performance. With Monte Carlo analysis comes the question of the desired output which can define the quality of the system, in addition to the number of required iterations.

DESIRED OUTPUT

Based on development of the overall architecture in the previous chapter, the outputs of a single trajectory simulation with the navigation system will be:

- State error (difference between true state and estimated state)
- Filter estimated covariance

Which can define the system performance for a single run. Due to the stochastic nature of the system however, to look at the trends in performance, these outputs may be used to look the general trends through use. These defining features are:

- Mean error
- RMS error
- True filter covariance

This will define the navigation system performance.

NUMBER OF MONTE CARLO ITERATIONS

To determine the number of Monte Carlo (MC) iterations, the test case for the integrated system is run but with varying numbers of iterations. The final mean error in state will be compared against the largest number of iterations and the standard deviation assessed. Note that the running time of the software plays some role in this, as requiring the software to operate for many hours at a time is deemed unfeasible for use. For this assessment, the angle sensor

is chosen, due to its non-linearities for the 2-day orbit used in the verification. The largest number of iterations is chosen to be 300 and this will be the benchmark 'truth' value.

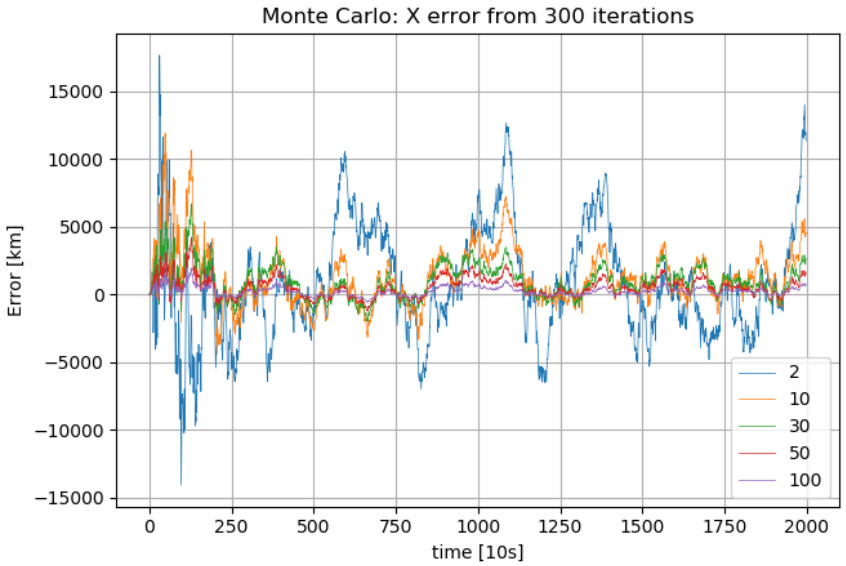
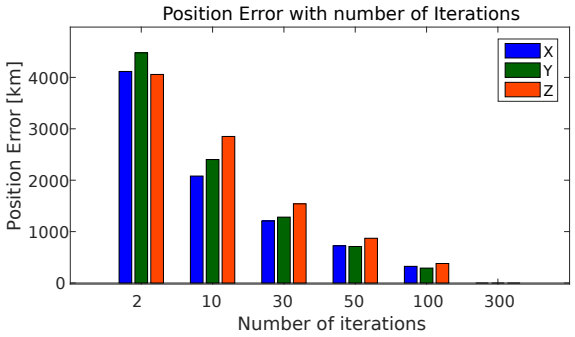
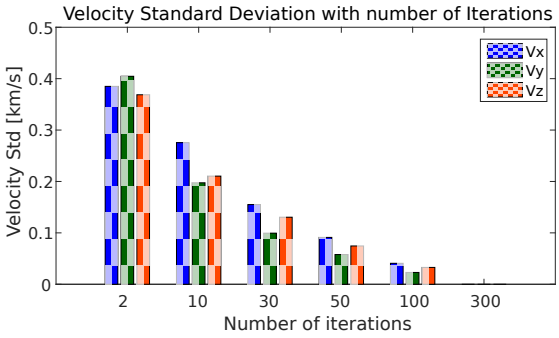


Figure 4.32: MC example x-state error compared to 300 iterations



(a) MC standard deviation for position. Legend shows iteration number.



(b) MC standard deviation for velocities

Figure 4.33: MC iteration numbers and standard deviation

As figure 4.32 shows, there is a large variation over the course of the simulation, further backed up by the standard deviations in figure 4.33.

Table 4.11: Percentage improvement in standard deviation over previous number of iterations

	Number of iterations			
Dimension	10	30	50	100
x	49.44242	42.00868	39.99453	55.21237
y	46.35462	46.6437	44.5542	59.16097
z	29.73132	45.97903	43.48064	56.40352
Vx	28.45702	43.71881	41.18528	55.26466
Vy	51.19604	49.6425	41.8782	60.22014
Vz	42.93046	37.99859	42.86912	56.02163

As table 4.11 shows, the largest relative difference comes from the 50 to 100 iteration mark. This Gives the largest impact compared to the lower number of iterations, and gives an acceptable relative standard deviation.

SOFTWARE ARCHITECTURE

As the CRLB has not been included in the first version of SAT-ANS, the analysis module has been simplified, to that shown in figure 4.34. The parameters mentioned above are generated from the simulation and also through the Statistics function. Additionally, the simulation parameters which are randomly varied according to the MC analysis are changed before starting another iteration.

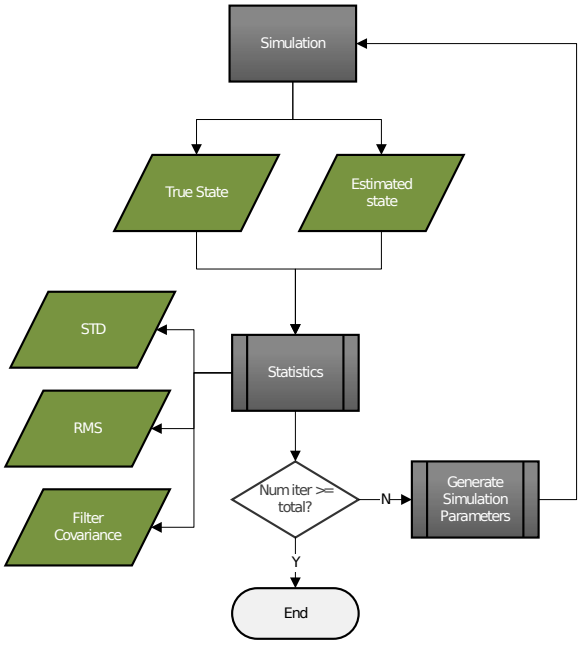


Figure 4.34: Simplified analysis module for SAT-ANS

4.6. SAT-ANS SUMMARY

The software tool SAT-ANS has been developed which allows different combinations of sensors to be tested with an implemented navigation filter over a user-defined trajectory. The key aspects of the different modules are briefly summarised here:

Orbit Module

- Define a Keplerian orbit with the Sun, Earth or Mars as the central body
- Choose a solver to calculate the true state of the s/c over the simulation time
- Provide a reference true s/c state every epoch
- Transform the output true s/c state from the planetocentric inertial coordinate system to barycentric
- Add noise to the true state

Sensor Module

- Define the observational equations of a sensor
- Read and interpret the library of observables for the sensor
- Add noise to the sensor observations
- Provide the sensor observations after the sensor integration period

Navigation Module

- Define a navigation filter

- Propagate the estimated s/c state with an internal dynamical model
- Format the observations coming from one or multiple sensors for use
- Combine the sensor observations with dynamics estimate of the state, while taking account of the noise characteristics of the dynamics and the sensors
- Provide an estimate of the state and state covariance when the filter is updated

Analysis Module

- Calculate the state and measurement equation derivatives
- Run MC analyses to assess the statistical properties of the navigation system

These factors combined, in addition to the relevant verification, allow the addition of further sensors such that navigation systems may be analysed. Pulsar navigation sensors will now be investigated and the observation algorithms implemented into SAT-ANS and the system will be assessed for the impact of sensor fusion.

4.6.1. UN-VALIDATED REQUIREMENTS

In the previous chapter, a series of requirements were generated and their validation methods proposed. However, during the course of the generation of the tool, not all of the requirements could be validated. Specifically the top-level, requirement TL-03 on the sizing capabilities of SAT-ANS was both not implemented, and therefore not validated. This was due to the time constraints of the design/generation phase of the tool. However, within the analysis module, this capability could be added using user requirements. These requirements should define the covariance or maximum permissible state error at specific epochs. In addition to user-defined constraints in sensor sizing aspects and sensor combinations. Using these inputs and the MC analysis, an optimum navigation system can be generated for the filter chosen. This could be extended further, in principle.

Another requirements which was not validated fully, is the creation of additional navigation filters (SAT-F-01.04). This was also due to the time constraints of the project. Although one navigation filter was implemented, the requirement stipulated multiple algorithms.

5

PULSAR NAVIGATION

SAT-ANS has been developed and tested with integrated radial velocity and angle sensors. The attention may now turn to the promising deep space autonomous navigation technology - pulsar navigation. This section researches the theory behind both x-ray and radio pulsar navigation. The generation of pulsar signals is investigated and the two sensor types are implemented into SAT-ANS. Two navigation methods - absolute navigation and delta correction, are assessed for use in the tool. The sensors are verified against literature and two separate pulsar libraries are generated as observables. Further, a noisy clock model is generated and added to SAT-ANS. Finally, a deep space test is generated, in addition to the planetocentric case used in the previous chapter to investigate the impact of sensor fusion on pulsar navigation in different orbital scenarios.

5.1. PULSAR NAVIGATION

Based on the detection wavelength band (radio or x-ray), pulsar navigation can be divided. Different methods of using pulsars were investigated and described in [17], and from this, two main methods of detection were established which could be used for deep space navigation: Delta Correction and Absolute Navigation - which use the timing characteristics of the pulsars. These methods are re-iterated here.

5.1.1. DELTA CORRECTION

Delta Correction uses the time of arrival (TOA) of a pulsar pulse and compares it to the predicted TOA from that pulsar. It has been considered for different X-ray pulsar navigation systems [25] [40].

To make an analogy with GNSS navigation, where the signal used is being generated by a satellite with precise clocks, in PNAV, the signal has an origin much further away. Additionally, the reference frame is also different. For GNSS the reference frame is generally Earth-centered. But for PNAV, it is the SSB. This is a motivation for including the functionality for coordinate transformations to the SSB into SAT-ANS. The precise 'clocks' of the pulsars allow the pulsar signal phase from their emission to be predicited at a future time using:

$$\Phi^{ref}(t) = \Phi^{ref}(T_0) + f(t - T_0) + \sum_{m=1}^{M} \frac{f^{(m)}(t - T_0)^{m+1}}{(m+1)!} \tag{5.1}$$

where $\Phi^{ref}(t)$ is the predicted phase at the SSB reference at time t, T_0 is an earlier time when the phase of the pulsar was known, f is the observation frequency and m is the differential order to which the phase change may be measured over longer time scales [12]. It has been found that order 4-5 is sufficient as this represents changes on the order of months to years [72] [17].

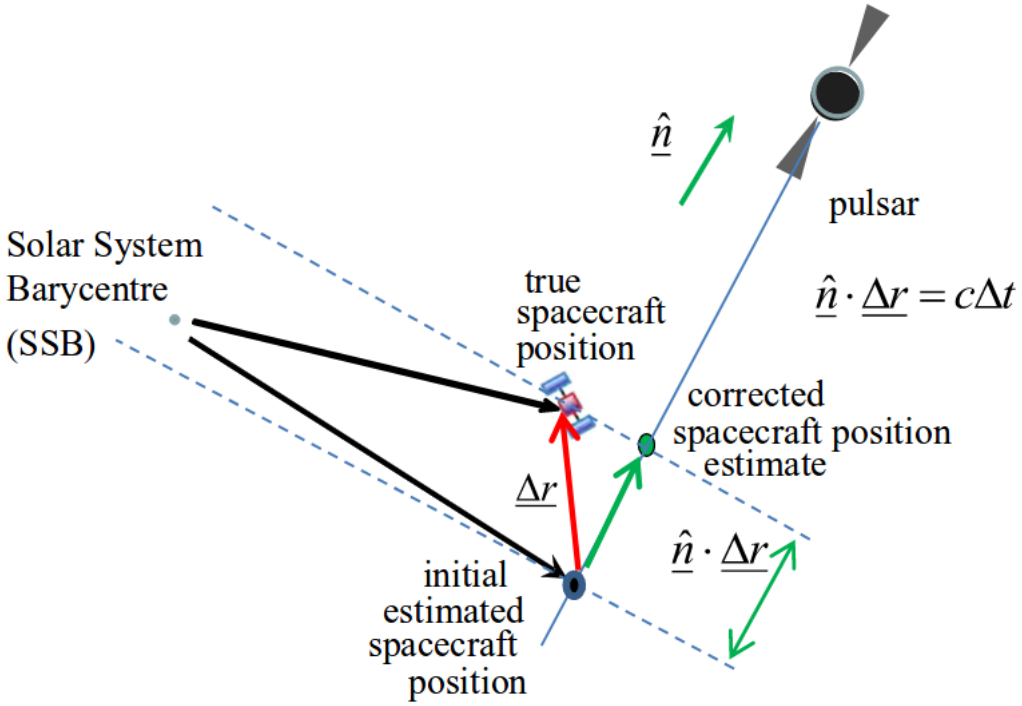


Figure 5.1: Representation of position correction in the direction of the observed pulsar - delta correction [8]

Pulsar TOA navigation, as mentioned above uses the predictability of the pulsar pulses and compares them to the detected arrival pulses. As the estimated state is not the true state, there is some time difference between the two, which corresponds to a position difference in the direction of the pulsar. However, to make this investigation, there must be common frame of reference for the comparison of the pulsar signals - the SSB. The time conversion equation, shown in 5.2.

$$\begin{aligned} t_{b_i} &= t_{obs_i} + \\ &\frac{1}{c} \times \left[\begin{array}{l} \hat{\mathbf{n}} \cdot \mathbf{r_i} - \frac{r_i^2}{2D_0} + \frac{(\hat{\mathbf{n}} \cdot \mathbf{r_i})^2}{D_0} + \frac{\mathbf{r_i} \cdot V \Delta t_i}{D_0} - \frac{(\mathbf{r_i} \cdot V \Delta t_i)(\hat{\mathbf{n}} \cdot \mathbf{r_i})}{D_0} - \frac{(\mathbf{b_i} \cdot \mathbf{r_i})}{D_0} \\ \\ + \frac{(\mathbf{b_i} \cdot \mathbf{r_i})(\hat{\mathbf{n}} \cdot \mathbf{r_i})}{D_0} \end{array} \right] \\ &+ \sum_{k=1}^{PBSS} \frac{2GM_k}{c^3} \ln \left| \frac{\hat{\mathbf{n}} \cdot \mathbf{r_{i_k}} + r_{i_k}}{\hat{\mathbf{n}} \cdot \mathbf{b_{i_k}} + b_{i_k}} + 1 \right| \end{aligned} \tag{5.2}$$

In equation 5.2, the barycentered time of the i-th pulsar pulse, t_{b_i}, is converted from the receiver's observed time t_{obs_i}. $\hat{\mathbf{n}}$ is the direction vector to the pulsar from the SSB, $\mathbf{r_i}$ is the spacecraft position relative to this point. D_0 is the distance to the pulsar at the transmission of the zeroth pulse, with $\mathbf{V}$ the pulsar's proper motion. Δt_i is the difference in transmission time between the zeroth pulse and and the N-th. Furthermore, $\mathbf{b_i}$ Sun's centre relative to the SSB origin. The fist term within the square brackets is the first order doppler delay, and the third and fourth terms are caused by annual parallax of the pulsar as observed at the SBB which when combined, form the Roemer delay. The final terms in the square bracket are known as Shapiro delay. The summation term accounts for the gravitational perturbation on the EM radiation due to solar system planetary bodies [19]. Note that effects due to the interstellar medium have been omitted here [17].

Once the two pulse arrival times are in the same reference frame through barycentering, the two TOAs may be subtracted from each other to form a timing residual. This timing residual is equaivalent to the light time delay between the estimated and true position. This may then be used to calculate the observer's position relative to the SSB in the direction of the observed pulsar.

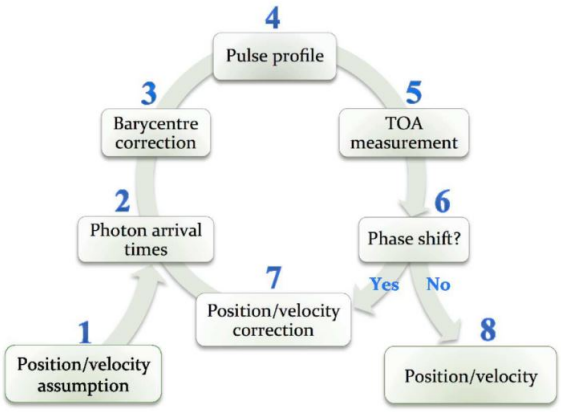


Figure 5.2: Pulsar TOA navigation flow chart [9]

Delta correction may be achieved in an iterative way as shown in 5.2, which follows the process described above. The aim in general is to minimise the timing residual such that the difference between estimated and true position is also minimised (assuming small errors in the pulsar model).

There are additional effects which may change the detection of the signal. Specifically Doppler shift which has been used in previous chapters for the radial velocity information and this in principle may also be done in pulsar detection - where the detected pulsar frequency changes. However, depending on the model, if not accounted for, the detection mechanism may not recognise the specific pulsar based on the rotation period alone.

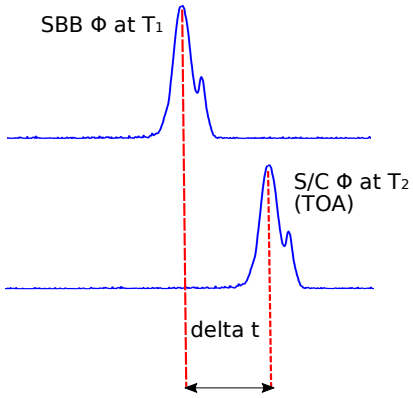


Figure 5.3: Pulsar TOA navigation - difference between reference phase and the arrival of phase at the s/c

Note how the term phase is defined at an arbitrary point - however the term Time of Arrival in this work is taken to mean the time of detection of the reference phase.

The observation of one pulsar will constrain the position solution in one dimension, therefore a minimum of three pulsars are required to get a solution for the observer.

5.1.2. ABSOLUTE NAVIGATION

A very desirable characteristic of an autonomous navigation system, is that it should be able to navigate without human intervention and little to no previous knowledge of its position. This case may occur after the s/c has recovered from a failure.

Contrary to delta correction which works to constrain the position in the direction of the single observed pulsar, absolute navigation requires the simultaneous observation of 3-4 pulsars to define the position. In GNSS the received pulses carry information, such that the integer number of phase cycles between the satellite and the observer may be inferred from the arrival time. This is however not the case for pulsars. Here, 3 or more pulsars are simultaneously observed and a unique solution exists with respect to the integer phase cycles which fits the observation. Once this has been identified, fractions of the phase are added to each of the pulsar observations to exactly match the observation and thereby define the observer's position [36].

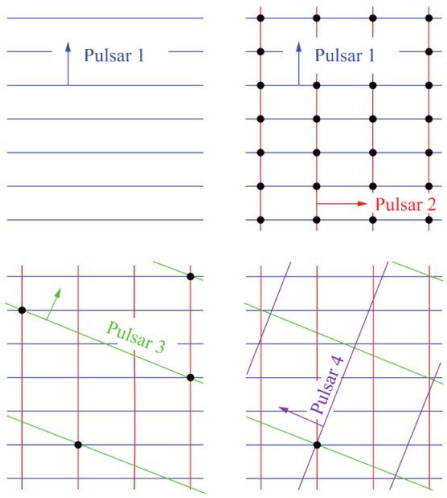


Figure 5.4: Pulsar observation to constrain the position solution [9]

Figure 5.4 shows that one unique solution exists for the integer cycles of the observed pulsars. Note, a fourth pulsar may be required for on-board clock calibration. Having mentioned the stability of millisecond pulsars in the introduction, these generally form the bulk of long term pulsar navigation beacons, but from a cold-start, having a group of pulsars with longer periods can lead to a navigation solution more quickly, due to having a smaller search space. For a SC measuring 3 pulsars, its position $[x, y, z]$ in inertial coordinates with origin at the SSB is:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 & y_2 & z_3 \\ x_1 & y_2 & z_3 \\ x_1 & y_2 & z_3 \end{bmatrix}^{-1} \begin{bmatrix} \frac{cP_1}{2\pi}\phi_1 \\ \frac{cP_2}{2\pi}\phi_2 \\ \frac{cP_3}{2\pi}\phi_3 \end{bmatrix} \tag{5.3}$$

where ϕ_i is the measured pulse phase of the i-th pulsar, P_i is the pulsar's period and $[x_i, y_i, z_i]$ are the pulsar's unit position coordinates.

The process of finding this unique set of integers is known as the ambiguity problem and is covered in the Navigation Problems section.

5.2. SOURCES OF ERROR/UNCERTAINTY

There are sources of error when observing pulsars, depending on the observation band. They may broadly be split into the following categories [73]:

- Pulsar emission and background noise
- Detector noise
- On-board clock noise
- Pulsar timing Irregularities

For this work, the errors such as Pulsar timing irregularities will not be considered. Further, the other errors may be combined in a linearised equation which considers only first order effects. The error in position $\delta\mathbf{r}$ is [19]:

$$c(\delta t_b - \delta t_{obs}) - \delta\hat{\mathbf{n}} \cdot \mathbf{r}' = \hat{n} \cdot \delta\mathbf{r} \tag{5.4}$$

This shows the error in position is strongly dependent on both the timing accuracy and the positional accuracy of the pulsar.

GEOMETRIC DILUTION OF PRECISION

using beacons as a navigational aid is geometric dilution of precision (GDOP). This refers to the contribution of information to the state estimate based on the positions of the beacons in space. Figure 5.5 shows this for two cases using foghorns as beacons. When, relative to the observer, the beacons are orthogonal, each beacon is contributing maximally to the two dimensions. However, when the beacons are close together, the contributions to different dimensions relative to the observer are less, leading to increased uncertainty in one of the two dimensions.

If the beacons are not isotropically distributed, GDOP will influence the navigation solution.

5.3. PULSAR NAVIGATION MODELS

There are in general three ways to model pulsars depending on their closeness to reality: High fidelity, an approximate model, and a low fidelity model. Each have an decreasing computational requirement respectively. The Approximate model, developed in [75] found a compromise between accuracy and computational use ability - showing that it is possible to simulate weeks of a spacecraft flight, with an accuracy that tended towards that of the high fidelity model. The different models will be assessed for use.

5.3.1. HIGH FIDELITY MODEL

Defined for X-ray pulsars, the high fidelity model [75] is as follows. The mechanism of pulsar detection, especially in x-ray is probabilistic, and defined by a Poisson process $P(t)$ which has units photons per second. The pulsar creates a periodic signal with period T_p and the the expected value of P will also be periodic:

$$E\langle P(t+T_p)\rangle = E\langle P(t)\rangle \tag{5.5}$$

where $E\langle\rangle$ is the expectation operator. As the pulsar will be observed for multiple periods, their observation must be done via periodic time-binning. A time bin vector of length n has elements B which is photon counts. The time bin B_j periodically repeats:

$$B_j(t+T_p) = B_j(t) \tag{5.6}$$

where $j \in \mathbb{N} \quad 1 \leq j \leq n$.

As pulsar navigation is a measure of timing differences, the time in the S/C frame must be known relative the reference point - in this case the barycentre:

$$\Delta t_{SSB} = \gamma \Delta t_{SC} \tag{5.7}$$

with

$$\gamma = \frac{1}{\sqrt{1-\left(\frac{v}{c}\right)^2}} \tag{5.8}$$

as the Lorentz factor. Now detector timing resolution δt_{SC} is introduced - as this is the limiting factor for determining arrival times of photons. This timing resolution therefore sets the simulation time step. However, if more than one photon arrives within one increment of the detector time resolution, then only one photon is detected.

$t_{SSB,E}$ is introduced as the effective time step in the barycentric reference frame and from its conversion the relative velocity of the s/c to the pulsar can be calculated:

$$t_{SSB,E} = t_{SSB}\left(\frac{\mathbf{v}\cdot\mathbf{x_p}+c}{c}\right) \tag{5.9}$$

$\mathbf{v}$ is the SC's velocity and $\mathbf{x_p}$ is a unit vector in the direction of the pulsar (assumed to be independent of position in the solar system).

With the bins defined, it is now possible to navigate using this information. To do this, first the bin duration Δt_b in the s/c frame must be corrected for the s/c motion:

$$\Delta t_b = \frac{T_P}{n}\gamma_e^{-1}\left(\frac{c}{\mathbf{v_e}\cdot\mathbf{x_p}+c}\right) \tag{5.10}$$

here the subscript $_e$ refers to estimated values. After the integration period, there is a cross correlation between the profile $E\langle P(t)\rangle$ and the bin counts b. The difference between the phase at maximum correlation and the phase at he expected profile peak (defined using some ephemeris) gives half the phase shift for whole the integration period:

$$\frac{1}{2}\Phi_p = \Phi_X - \Phi_E \tag{5.11}$$

Unmodelled dynamics will cause a larger difference between measured and estimated phase as photons will be detected at different times compared to the model (known as smearing). For unmodelled acceleration it may be assumed that this smearing effect is linear, and so the epoch of the half-phase point will occur at the mid-point of the integration period. The spatial phase shift over the integration period is:

$$\Delta\Phi_L = \Phi_p T_p c + (\mathbf{v_e}\cdot\mathbf{u}) T_I \gamma_e \tag{5.12}$$

where T_I is the integration time.

This also creates a convention with sign - if the detected pulse phase is delayed, then Φ_p is negative. To solve for position in 3 dimensions (and omitting for the moment the idea of dilution of position), 3 different pulsars must be observed simultaneously which lead to the following system of equations:

$$\begin{bmatrix} \mathbf{x_{p_1}} \\ \mathbf{x_{p_2}} \\ \mathbf{x_{p_3}} \end{bmatrix} (\mathbf{s}_p - \mathbf{s}_{p-1}) = \begin{bmatrix} \Delta\Phi_L^1 \\ \Delta\Phi_L^2 \\ \Delta\Phi_L^3 \end{bmatrix} \tag{5.13}$$

$$\bar{\mathbf{x_p}} \Delta\mathbf{s}_p = \Delta\bar{\Phi}_L \tag{5.14}$$

This can, in turn solve for the position of the spacecraft by taking the inverse of the position vectors on both sides of the equation 5.11, and then adding this difference to the position estimate at the start of the integration period.

SIGNAL DETECTION

If the pulsar signal is to be modelled in a realistic way, the detection of that signal must then also be modelled. This may be done in two ways: Signal folding and matched filtering

Signal Folding he pulsar signal is periodic with additive noise. As the noise is zero mean, if the periodicity of the signal is known (like for the pulsar) each period may be summed leading to the noise to tend to zero. This is shown in figure 5.7 [17] [72].

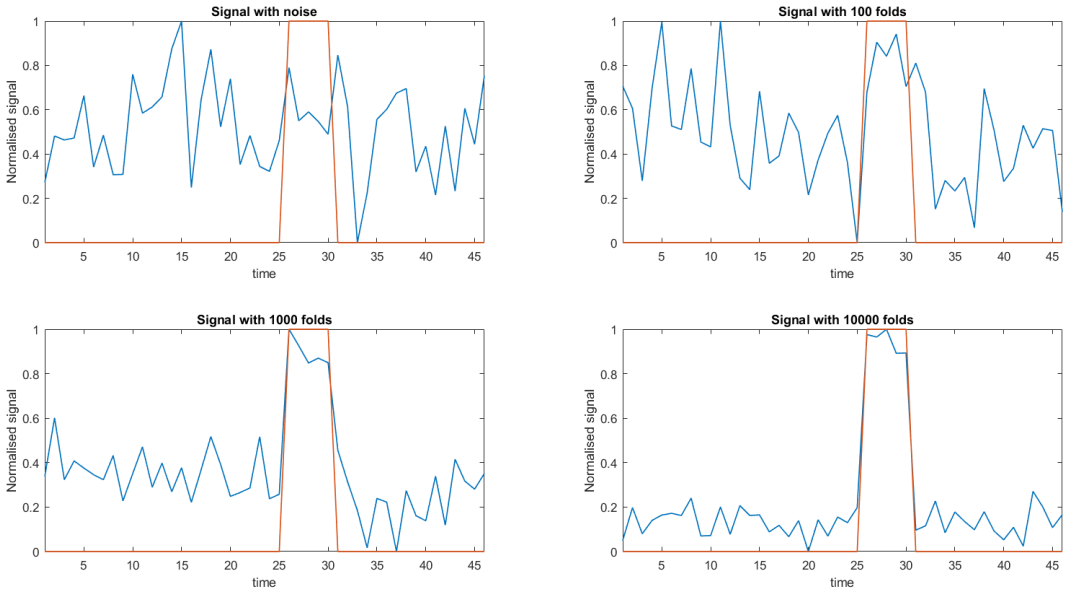


Figure 5.6: Example of folding. Blue is the simulated normalised detected noisy signal and red is the normalised pulsar 'profile', in this case a square wave with a period of 46 s. The number of folds increases from one to 10,000. After 1000 folds, the signal is stronger than the noise and very clearly distinguished after 10,000 folds.

If the s/c is moving relative to the pulsar, then the detected period of the pulsar will change, which would need to be incorporated into the signal folding.

Matched Filtering Matched filtering uses the profile of the pulsar to increase the SNR. This would need to included in a database - which would be required for the recognition of the pulsar. For matched filtering, this stored profile is used to correlate the detected signal and thereby decrease the required detection time [76].

LIMITATIONS

The high fidelity pulsar model is not perfect. It is very computationally invasive and only works if it is possible to observe three pulsars simultaneously for the whole integration period. However, this leaves a few questions:

- What is the optimum observation period for these three pulsars?

- What solution is possible if the pulsars are not able to be observed, or need to be observed sequentially?
- This method assumes perfect clock synchronisation between the S/C and the reference. How can clock error be accounted for?

These limitation may need to be addressed if this method is chosen. However, the main focus of this section is the generation of the pulsar model. To this end, the approximate model will now be investigated.

5.3.2. APPROXIMATE MODEL

The main limiting factor in the high fidelity model is the time-step. The NICER x-ray instrument on-board the ISS has a timing resolution of around 100 ns. For a deep space mission, where the navigation may need to be investigated over the course of weeks, a time step value of 1×10^{-6} s would require on the order of 1×10^{12} time steps. Although the evaluation of the computational speed and efficiency is beyond the scope of this work, this number is clearly too large for comparative testing of integrated sensors.

A way to overcome this, is to replace the process of generating the photons on the sensor bin level with the detection of a profile which would be created after each pulsar period. By doing so, the same statistics can be used, but created in an array. This would then set the lower limit of the simulation time step to the pulsar period (which would be over three orders of magnitude greater). In principle it may be possible to extend the time step to the whole integration period, however this will need to be investigated.

PULSAR PROFILES

Both the high fidelity and the approximate model require the pulsar light profiles for their implementation. In the introduction, it was stated that the pulsar's emission profile/light curve (the detected pulsar's emission radiation as a function of period/rotational phase) changes over the course of the pulsar's emission spectrum. For many pulsars which have been timed for many years, there exists a repository for these profiles as data sets for specific energy bands (mostly in radio). However, for pulsar navigation, a parameterised version of the profile is helpful as it reduces the required computational time of interpolating over data set. To date there are no specific databases/programs which will parameterise these data sets.

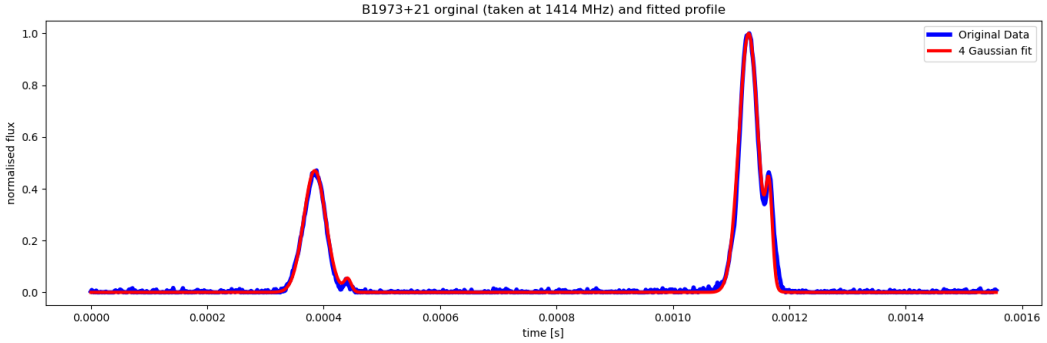


Figure 5.7: Comparison of original pulse profile from database (with reference to [10]) and a four-Gaussian fit

The approach to parametarising the profiles is to assume that they are based on Gaussian curves. To fit the curves, it was found that a least-squares approach where an approximation of the Gaussian is initially given, was very sensitive to the initial estimate of the parameter set. The Gaussian parameters are therefore first tuned by eye and then a least squares optimisation is performed. Figure 5.7 shows a pulsar often used in PNAV, B1937+21 with the original profile from the database, and the Gaussian fit superimposed. For the first iteration of this model, this approach would be considered acceptable if one of the above methods for PNAV is chosen, but later iterations may consider a more rigorous method.

5.3.3. LOW FIDELITY MODEL

The low fidelity model calculates only the phase of the pulsar and the TOA from this navigation can be done. This method is the simplest to implement, and uses the fundamental equations of pulsar navigation as mentioned previously As a first attempt into pulsar navigation, the low fidelity model will be used. This is because the basic equations remain the same - which will allow specific trends to be seen. The higher fidelity models include greater complexities, leading to more realistic results.

http://www.epta.eu.org/epndb/ (as of 19/07/2018)

With the model chosen, the two different PNAV methods may be described within the framework of this model, and the initial implementation into SAT-ANS chosen.

ABSOLUTE NAVIGATION

If multiple pulsars can be observed simultaneously, then this method can be employed. for a first order phase model [11]: The signals emitted by the pulsars are periodic, and if these signals are to be used for navigation, there is an unknown whole number of pulses that exist - phase ambigutiy as mentioned previously. This can be incorporated into the signal model. Equation 5.1 can be defined to first order:

$$\Phi_k(t) = [\Phi(t_0) + f_k(t - t_0)]_1 \tag{5.15}$$

$\Phi_k(t)$ represents the phase of the k-th pulsar at time t, based on the phase at t_0 and the pulsar frequency $f_k = \frac{1}{T_k}$. $[x]_1 = x + m$ is a modulo 1 reduction where m is the unknown integer number of pulses which must be found. Note, how as this is a first order model, only the frequency of the pulsar is required, and the specific profile characteristics are omitted. The spacecraft produces an estimate of the phase,

$$\hat{\Phi}_k(t) = [\Phi_k^{SSB}(t - \tau_k) + n_k]_1 \tag{5.16}$$

Φ_k^{SSB} is the SSB-relative phase, $\tau_k = \frac{\mathbf{u}_k^T \mathbf{x}}{c}$ is the light-delay due to the position of the s/c $\mathbf{x}$ relative to the SSB, and the pulsar $\mathbf{u}_k^T$. n_k is the noise associated with the estimation of k-th pulsar phase.

This leads to the following equation for the phase estimation:

$$\hat{\Phi}_k(t) = \Phi_k^{SSB}(t_0) + f_k(t - t_0 - \tau_k) + n_k + m \tag{5.17}$$

$$= \Phi_k^{SSB}(t_0) + f_k\left(t - t_0 - \frac{\mathbf{u}_k^T \mathbf{x}}{c}\right) \tag{5.18}$$

From this, the following variables may be considered unknown and require estimation:

$\mathbf{x}$: The s/c position

$t - t_0$: The time difference from the reference epoch

m : The integer number of phases between the SSB and the s/c

This sets the minimum number of pulsars to be observed to four. However, if it is assumed that there is no clock error between "true" time and on-board clock time, then $t - t_0$ is known and this number is reduced to three.

DELTA CORRECTION

Delta correction can be done independently of the pulsar profile, as described by equation 5.2. This makes it an attractive starting point for the implementation of pulsar navigation. The time of arrival of a specific pulse is transformed and compared against an estimate based on the navigation system s/c state. If the barycentric arrival time is omitted, delta correction is in principle independent of the pulsar timing parameters, and based only on the positional parameters. The barycentric arrival time, which in reality would be defined some look-up table or high order pulsar timing model [25], could be estimated based on the start time of the simulation and the propagation of time.

Delta correction is chosen for the algorithm used in the pulsar detection for SAT-ANS.

5.4. PULSAR DETECTION

As mentioned previously, there are (in general) two different electromagnetic bands in which pulsars are detected: Radio and X-ray.

5.4.1. RADIO

For a radio pulsar, the detected signal is dependent on the area of the receiver, the frequency and bandwidth of the receiver, the flux and spectral index around that frequency (over the bandwidth), and finally the polarization of the detector. A radio detector generally consists of two components - an antenna and the detection mechanism. The antenna is generally the easiest to scale, and this impact is shown in figure 5.9.

Figure 5.8 relates the SNR of the brightest millisecond pulsars to the effective antenna area.

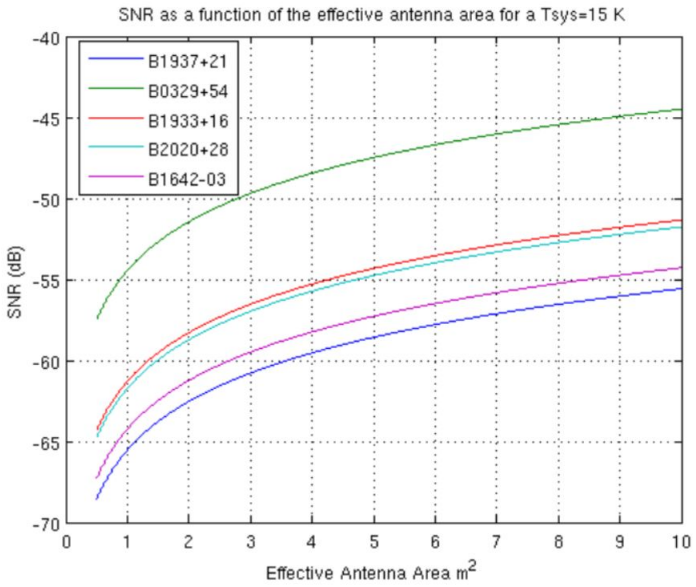


Figure 5.8: Signal to Noise ratio (with system noise temperature at 15 K) as a function of antenna area for a series of millisecond pulsars [11]

For a specific pulsar ,i, the received signal may be defined as [77]:

$$S_i = \alpha A_e 10^{-26} S_i^p \left(\frac{\nu_{rec}}{\nu_{ref}}\right)^{\beta_i} B \tag{5.19}$$

where the parameters are described in table 5.1.

Table 5.1: Receiver and pulsar parameters for the detection in the radio spectrum

Parameter	Definition	Unit
α	Polarization parameter ($0 < \alpha < 1$)	[-]
A_e	Effective area of the receiver	m^2
ν_{rec}	Receiver central frequency	Hz
B	Receiver bandwidth	Hz
ν_{ref}	Reference frequency	Hz
S^p	Pulsar flux at ν_{ref}	Jy
β	Pulsar spectral index	[-]

The receiver noise power can be defined as:

$$S_n = k_B(T_{rec} + T_{back} + T_{galaxy} + T_{solar})B \tag{5.20}$$

Table 5.2: Radio pulsar detection noise parameters, the receiver frequency units in these expressions is GHz

Parameter	Definition	Value
k_B	Boltzmann constant	1.3806×10^{23} [J/K]
T_{rec}	Noise temperature of the receiver	[-] [K]
T_{back}	Noise temperature of the background cosmos	2.7 [K]
T_{galaxy}	Noise temperature of the galaxy	$6\nu_{rec}^{-2.2}$[K]
T_{solar}	Noise temperature of the solar system	$(72\nu_{rec} + 0.058) A_e 10^{A/10} d^{-2}$[K][a]

[a] d (AU) distance of observer from Sun, A is the antenna main sidelobe attenuation

The signal to noise ratio can then written as:

$$SNR_i = \frac{S_i}{S_n} = \left(\frac{\alpha A_e 10^{-26}}{k_B(2.7 + 6\nu_{rec}[\text{GHz}]^{-2.2} + (72\nu_{rec}[\text{GHz}] + 0.058) A_e 10^{A/10} d^{-2}}\right) S_i^p \left(\frac{\nu_{rec}}{\nu_{ref}}\right)^{\beta_i} \tag{5.21}$$

The error in the signal (from a radio perspective), may be defined based on a CRLB analysis from [12]:

$$\sigma_t^2 \geq \frac{1}{(2\pi)^2 SNR_i^2 Q_{timing} B t_{int}} \tag{5.22}$$

Where Q_{timing} relates the stability of the pulsar timing signal. It is generated by analysing the Fourier transform of the pulsar mean power profile. An anlysis done in [77] found the following pulsars to be the best for navigation, based on the above equation:

Table 5.3: Ten best pulsars for radio pulsar navigation

Pulsar	Period [s]	S_p [mJy]	Q_p [dB]
B1937+21	0.0016	15.8	44.41
B0329+54	0.7145	202.8	33.74
B1933.16	0.3587	42	26.54
B2020+28	0.3434	37.9	23.19
B1642-03	0.3877	21.3	22.54
B0950+08	0.2531	85.3	22.33
B1929+10	0.2265	36.0	22.02
B1133+16	1.1879	31.6	18.16
B0740-28	0.1667	14.9	17.79
B1749-28	0.5625	17.9	17.48

where $Q_p = S_p^2 Q_{timing}$ This allows the pulsar phase to be calculated (to first order) and the signal to be detected using radio detectors.

TIMING TO DISTANCE ACCURACY

For a continuous signal from an x-ray source, the 1-sigma timing error in the signal is defined in equation 5.24 and the timing error can then be converted to a position error:

$$\sigma_D = c\,\sigma_{TOA} \tag{5.23}$$

If the error in TOA is known, this gives a simple analysis of RNAV.

Outside this method, the ESA study found the lower limit of accuracy for specific pulsars [12]. Figure 5.9 shows these timing accuracies possible for the 10 best pulsars.

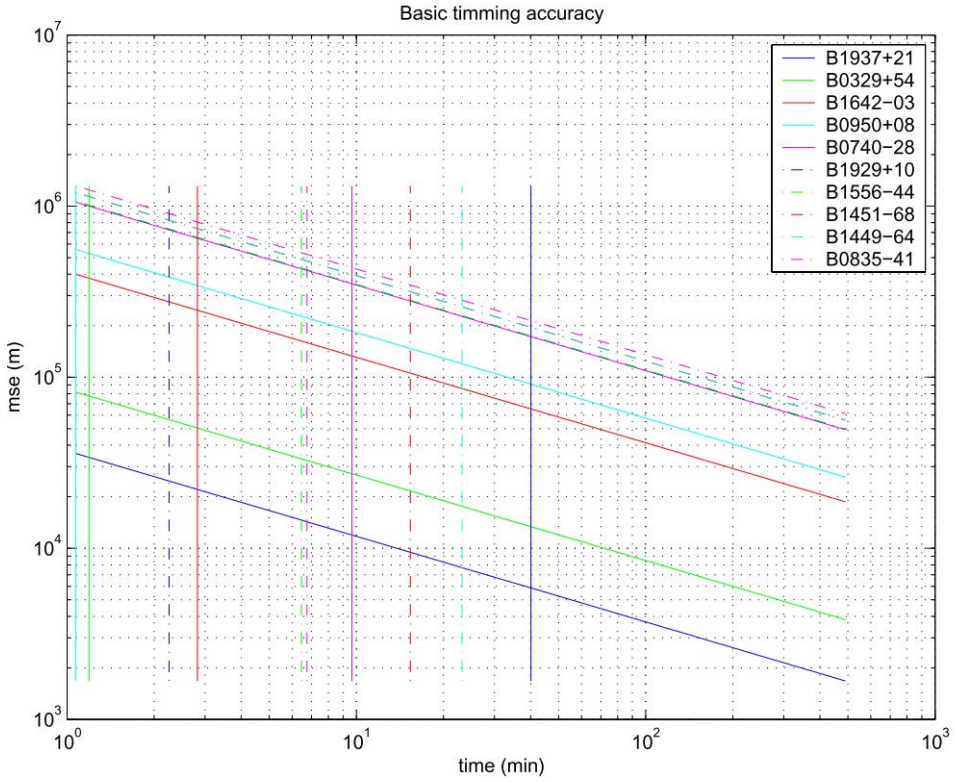


Figure 5.9: Timing accuracies for the 10 best pulsars [12]

5.4.2. X-RAY

Similar to RNAV, the x-ray pulsar navigation timing equation and error are also dependent on the SNR and the pulsar parameters. The 1-sigma TOA error has been shown to be [19]:

$$\sigma_{TOA} = \frac{W}{2SNR} \tag{5.24}$$

This has similar characteristics to the radio error equation. The SNR may be defined with respect to integration time as follows [19]:

$$SNR = \frac{F_p A p_f t_{obs}}{\sqrt{[B + F_p(1-p_f)](A t_{obs} d) + F_p A p_f t_{obs}}} \tag{5.25}$$

where F_p is the observed signal from the pulsar, A is the (effective)detector area, p_f is the faction of the signal which is pulsed, t_{obs} is the observation time and d is the duty cycle; $d = \frac{W}{P}$ with P the period [17]. As shown by the equation, a lower duty cycle and a higher SNR will lead to a lower error in the TOA error.

A list of the best pulsars for navigation in the x-ray spectrum are given in 5.4

Table 5.4: [17] Best Millisecond pulsars (and the Crab) for XNAV using a NICER-style detector [18].
[a] data taken from [19]
[b] data taken from [20]
[c] data taken from [21]
[d] data taken from [22]. For pulsed fraction for J0437-4715, the Boron data set was used.
[e] data taken from [23]
[f] data taken from [24] unless otherwise stated
[g] data taken from [25] where the ratio is of the signal to noise

Name	Energy [erg/s/cm^2][f]	Noise count rate ratio (-)[g]	Background energy [erg/s/cm^2]	Period [ms]	Pulsed fraction [%]	Pulse Width [s]
B0531+21 (Crab)	9.93×10^{-9}	21.0	2.09×10^{-7}	33.51	70[a]	1.7×10^{-3}
B1937+21	3.70×10^{-13}	8.28	3.064×10^{-12}	1.56	86 [a]	2.1×10^{-5}
B1821-24	1.25×10^{-12}	2.366	2.29×10^{-12}	3.05	98 [a]	5.5×10^{-5}
J0218+4232	4.1×10^{-13}[78]	2.44	1.0×10^{-12}	2.32	73 ± 12 [b]	6.9×10^{-4}
J0030+0451	1.27×10^{-13}	1.036	1.316×10^{-13}	4.87	69 ± 18 [c]	7.31×10^{-4}
J1012+5307	1.25×10^{-14}	4.345	5.43×10^{-14}	5.26	75[d]	6.9×10^{-4}
J0437-4715	4.30×10^{-13}	2.191	9.42×10^{-13}	5.76	26 ± 9 [d]	1.41×10^{-4}
J2124-3358	8.26×10^{-14}	2.703	2.23×10^{-13}	4.93	33 ± 8 [d]	5.24×10^{-4}
J2214+3000	4.365×10^{-14}[79]	8.97	3.92×10^{-13}	3.12	72 (mean assumed)	3.12×10^{-4}(est)
J0751+1807	4.29×10^{-14}	8.8	3.78×10^{-13}	3.48	70 [d]	7.0×10^{-4}
J1024-0719	8.86×10^{-15}	13.33	1.18×10^{-13}	5.16	52 [e]	2.69×10^{-3}

5.5. IMPLEMENTATION

5.5.1. SOFTWARE FLOW

The general mechanism for pulsar delta correction as implemented in SAT-ANS is shown below. First the (estimated) state is transformed using the on-board estimated time, after that the pulsar TOA's are generated from the pulsar library.

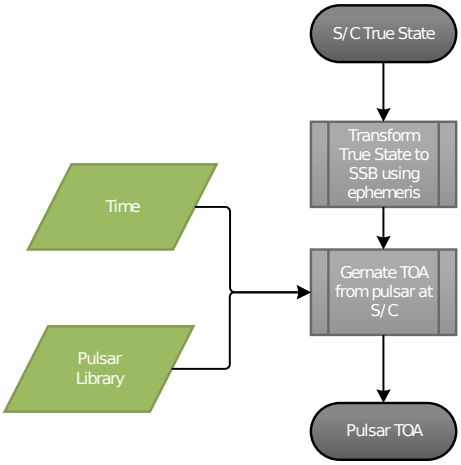


Figure 5.10: PNAV flow chart implemented in SAT-ANS: PNAV phase generation

5.5.2. PULSAR LIBRARY

Similar to the implementation of the angle sensors, true pulsars may not be considered to be fixed on the celestial sphere but have proper motion, and should be based on an ephemeris.

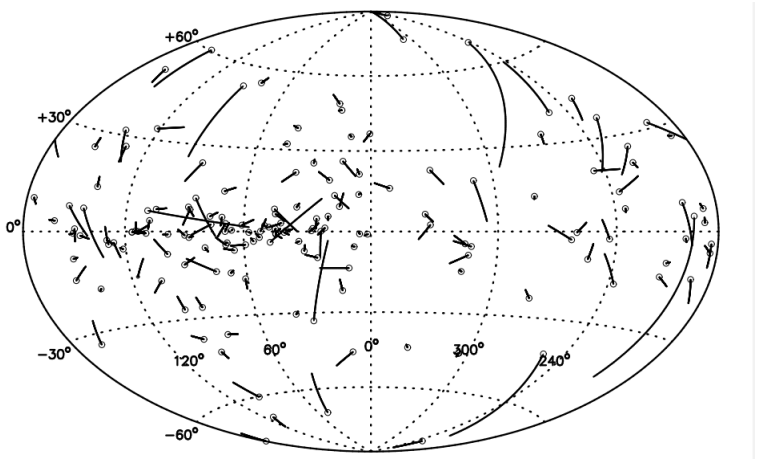


Figure 5.11: Proper motion of a subset of known pulsars over the last million years [13]

However, considering project time constraints, and that the pulsar's move of the order of tens of milli-angular seconds per year across the celestial sphere and so of the order of micro-angular seconds per day[13]. The error in the knowledge of the angular position of pulsars is generally higher, as mentioned previously. The pulsars may therefore be considered stationary with respect to the solar system over short-term simulations. The angular position of the pulsars is however required, and will be added to the library.

Table 5.5: Radio pulsar Positions in galactic reference frame

Pulsar	Period (s)	Galactic Longitude [deg]	Galactic Latitude [deg]	distance [kpc]
B1937+21	0.0016	57.51	-0.29	3.50
B0329+54	0.7145	145	-1.22	1.0
B1933+16	0.3587	52.44	-2.09	3.7
B2020+28	0.3434	68.86	-4.67	2.10
B1642-03	0.3877	14.11	26.06	1.32
B0950+08	0.2531	228.91	43.70	0.26
B1929+10	0.2265	47.38	-3.88	0.31
B1133+16	1.1879	8.56	0.33	0.35
B0740-28	0.1667	243.77	-2.44	2.00
B1749-28	0.5625	1.54	-0.96	0.20

Table 5.6: X-ray pulsar positions in galactic reference frame

Pulsar	Period [s]	Galactic Longitude [deg]	Galactic Latitude [deg]	distance [kpc]
B0531+21	0.033	184.56	-5.78	2
B1937+21	0.0016	57.51	-0.29	3.50
B1821-24	0.0031	7.8	-5.58	5.1
J0218+4232	0.00232	139.51	-17.53	3.15
J0030+0451	0.00486	113.14	-57.61	0.32
J1012+5307	0.00525	160.35	50.86	0.7
J0437-4715	0.00575	253.39	-41.96	0.16
J2124-3358	0.00493	10.93	-45.44	0.41
J2214+3000	0.00311	86.86	-21.67	0.60
J0751+1807	0.00347	202.73	21.09	1.11
J1024-0719	0.00516	251.70	40.52	1.22

The galactic reference frame (from which the latitude and longitude are given) is a sun-centered reference frame. given the distances (order of kiloparsecs 10^{19} m) the pulsar angular position is considered independent of the position within the solar system (a change of 0.1 ” from the Sun to Neptune m)

a kiloparsec is assumed for the average distance to a pulsar and the Sun-Neptune distance is of the order of 5×10^{12}

RADIO

For the radio spectrum, the best pulsars for timing estimation (and so navigation) are listed in table 5.3. These are added to the pulsar library. In addition to that, the sensor-independent parameters from equation 5.29 are required. The spectral index for radio pulsars has a mean value of around -1.8 ± 0.2. This law holds for the majority of pulsars and for frequencies above 100 MHz, below which a turnover is often seen [34]. This value of -1.8 will be assumed for all the pulsars

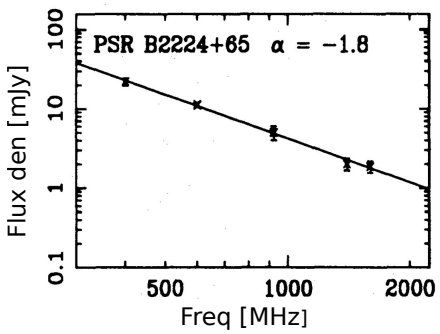


Figure 5.12: B2224+65 example flux density spectrum. X-axis is frequency (MHz) and Y-axis is flux density (mJy) [14]

Interstellar Medium Issues The space between star systems in galaxies contains the interstellar medium. Composed of gas, dust and ionized plasma, it acts to slow the propagation of electromagnetic signals. This retardation of signals is frequency dependent, thereby causing a delay in the signal over the bandwidth of the received signal, as shown in figure 5.13.

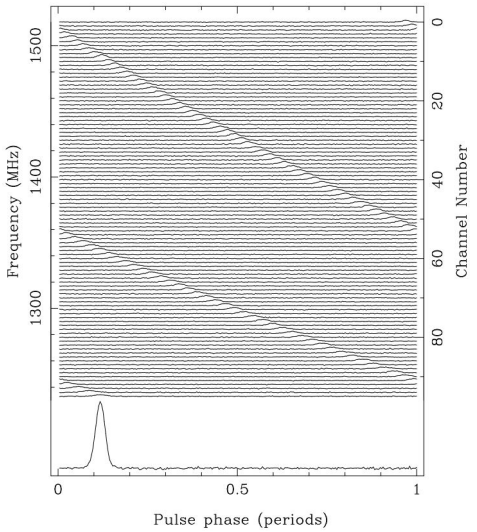


Figure 5.13: Pulse dispersion shown in B135-60 observation [15]

For the proper treatment of the detection of the pulsar signal, this would need to be included in the model, however as the output of this model is the final detected phase, the noise temperatures should account for this.

X-RAY

The propagation of X-rays are unaffected by the interstellar medium, and the factors required for the library are all included in equation 5.25

Units The fluxes in the previous section are all quoted in ergs/cm^2/s (1 erg = 1×10^{-7} J = 1.609×10^{-9} KeV), however the energy unit must be converted a photon flux for use. For this an energy bandwidth is required and the majority

of the bandwidths listed in table 5.4 are 8 keV. The following conversion is then used:

$$S_i(E) = E \cdot N_i(E) \tag{5.26}$$

Where E is the energy bandwidth, S_i is the flux, and N_i is the photon count

5.5.3. TOA GENERATION

There is some ambiguity when referring to the detection of signals in PNAV - TOA and phase are used quite interchangeably. To specify, an arbitrary phase is generated per pulsar at the start of every simulation (each start at zero phase at the SSB side and increase with time). The TOA is the time of arrival at the s/c of a specific phase referenced at the SSB at a specific instance in time. For the delta correction algorithm

The time transfer equation 5.2 is very detailed for this implementation - accounting for the movement of the pulsars, in addition to time delays due to all the gravitational bodies in the solar system. This would be very time consuming to implement as written. However, some simplifications can be made [26], namely as the distance from the SSB to the i-th pulsar is much greater than the change in position of the pulsar due to proper motion over the course of an integration period (and that the proper motion of the pulsars is being omitted):

$$\begin{aligned} t_{SC_i} = t_{b_i} + \frac{1}{c}\hat{\mathbf{n}}_i \cdot \mathbf{r} + \frac{1}{2cD_{0_n}}\left[-r^2 + (\hat{\mathbf{n}}_i \cdot \mathbf{r})^2 + b^2 - (\mathbf{b} \cdot \hat{\mathbf{n}}_i)^2\right] \\ + \frac{2GM_s}{c^3}\ln\left|\frac{(\mathbf{r}-\mathbf{b}) \cdot \hat{\mathbf{n}}_i + ||\mathbf{r}-\mathbf{b}||}{\mathbf{b} \cdot \hat{\mathbf{n}}_i + b} + 1\right| \end{aligned} \tag{5.27}$$

additionally, the dependence due to different gravitational effects has been reduced to just the Sun. This equation has been implemented into the model for the transformation of generated phase at the SSB to the S/C side.

5.6. VERIFICATION

5.6.1. X-RAY

Using the data for the three best pulsars found in [19] and for a detector of 1800 cm^2, the range error was plotted as a function of integration time, as shown in figure 5.14. For this analysis, the flux was provided in 5.4.

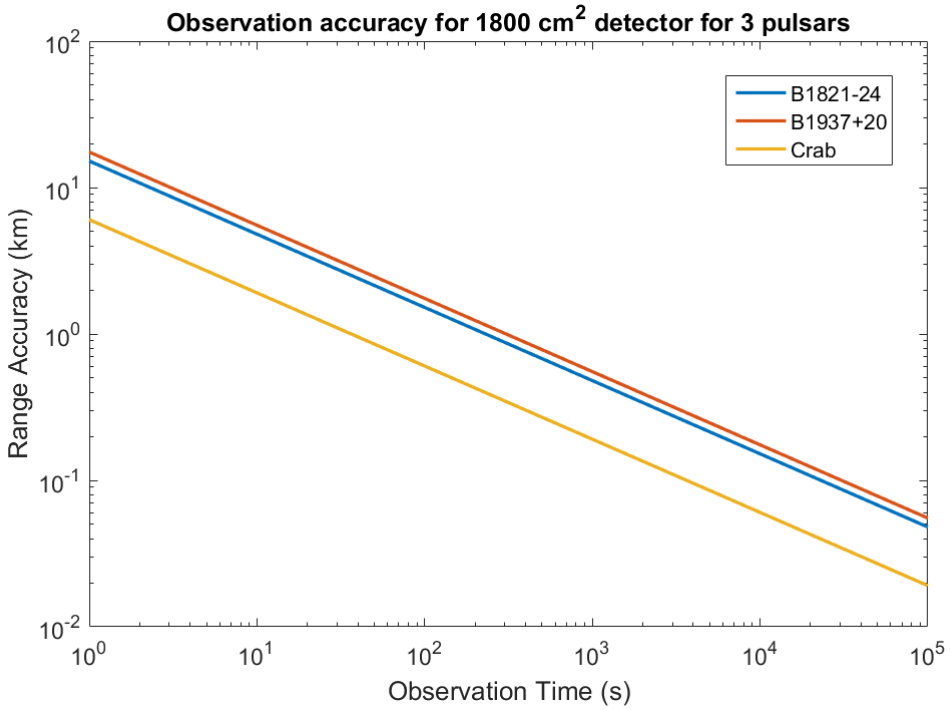


Figure 5.14: The range error to B0531+21 (Crab), B1937+21 and B1821-24 as a function of observation time using an 1800 cm^2 detector in the energy range of 2-10 keV, assuming a constant x-ray background of 7.87 photons/s. Data taken from [16].

As figure 5.14 shows, the range accuracy to the three pulsars each is lower than 1 km after 1000 s of observation time. This compares well with [80]. Additionally, by rearranging equations 5.24 to 5.23 it may be interesting to see the

integration time as a function of detector area assuming a range accuracy error of 1 km to give an idea of the limits of accuracy on detection area for possible small satellite applications.

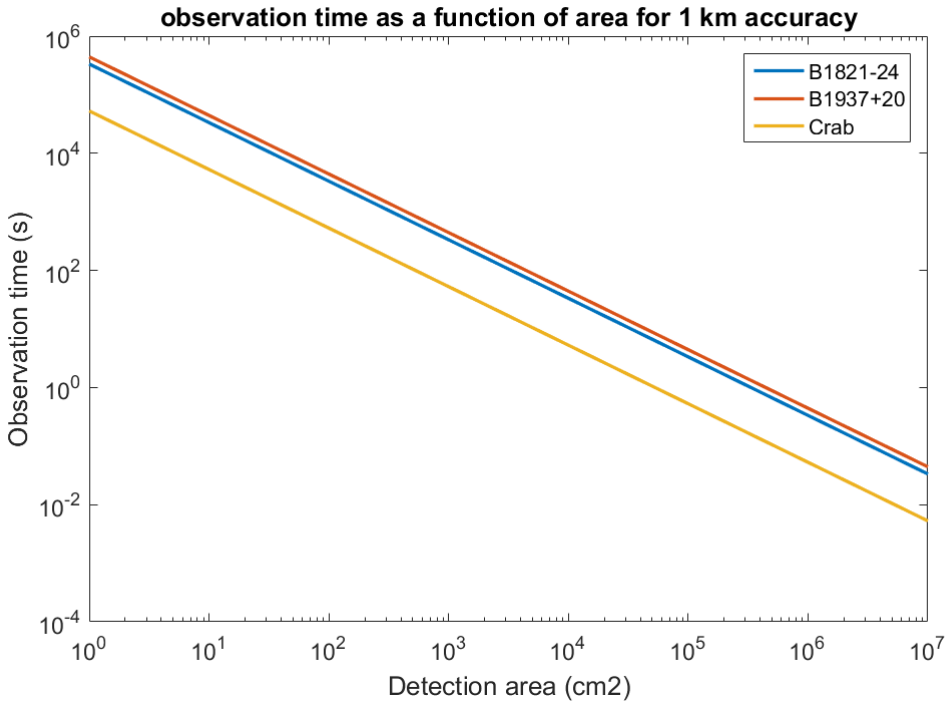


Figure 5.15: The observation time with detection area of B0531+21 (Crab), B1937+21 and B1821-24 with a 1 km range error. Data taken from [16].

5.6.2. RADIO

For the verification of the radio pulsar navigation, first the implementation of the timing error is verified along equation 5.21. The following parameters are used in addition to the library in table 5.3:

Table 5.8: Radio pulsar SNR verification

Parameter	Value
α	0.5
A_e	100 m^2
B	400 MHz
ν_{rec}	1.4 GHz
ν_{ref}	1.4 GHz
β	-1.8
A	-40 dB
d	1 AU
T_{rec}	15 K

Pulsar	SNR [dB]
B1937+21	-45.8
B0329+54	-34.7
B1933+16	-41.5
B2020+28	-42.0
B1642-03	-44.5
B0950+08	-38.4
B1929+10	-42.2
B1133+16	-42.8
B0740-28	-46.0
B1749-28	-45.2

which matches well with [77]. This then leads to a distance error as a function of integration time, as shown in figure 5.16

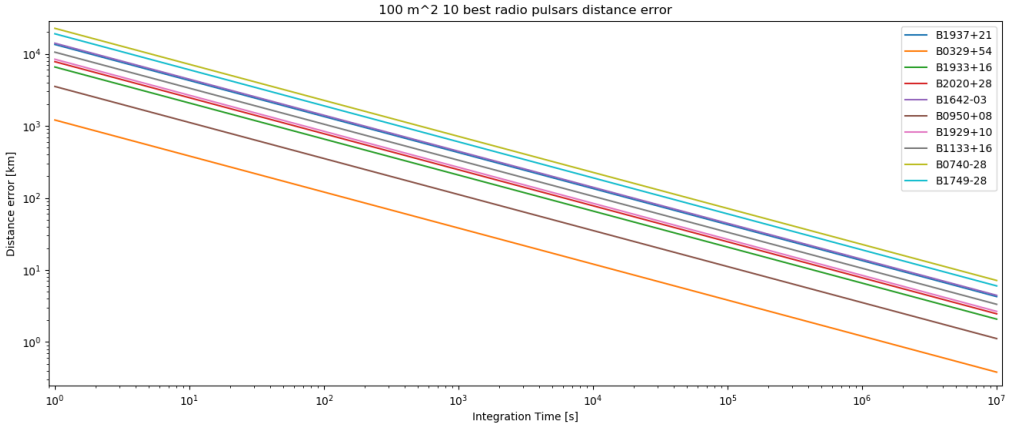


Figure 5.16: distance error with integration time for RNAV pulsars

5.6.3. NOISE

Two main noise sources of pulsar navigation exist: Pulsar position error and on-board clock error.

PULSAR POSITION ERROR

The position of every pulsar is not known to arbitrary precision, which in turn will impact the TOA estimations. The pulsar position error is not included here, but the application can be found in F.

ON-BOARD CLOCK

Based on previous research [17], it may be assumed that the clocks on-board S/C using PNAV may be modelled by atomic clocks. However, even the best clocks have some noise, which causes their time to drift relative to 'true' time. To give a more accurate representation of PNAV, a clock model will be added to SAT-ANS.

Clock error may be represented by a 3rd order discrete state model defined as:

$$\begin{aligned} dX_1 &= (X_2(t) + \mu_1)dt + \sigma_1 dW_1(t) \\ dX_2 &= (X_3(t) + \mu_2)dt + \sigma_2 dW_2(t) \\ dX_3 &= \mu_3 dt + \sigma_3 dW_3(t) \end{aligned} \tag{5.28}$$

where the three clock error state elements represent the clock error itself, the clock error drift, and the rate of change in clock drift. The differential equation can be written discretely as:

$$\mathbf{X}_{k+1} = \Phi\mathbf{X}_k + \mathbf{BM} + \mathbf{W}_k \tag{5.29}$$

Where

$$\Phi = \begin{bmatrix} 1 & dt & \frac{dt^2}{2} \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \tag{5.30}$$

$$\mathbf{B} = \begin{bmatrix} dt & \frac{dt^2}{2} & \frac{dt^3}{6} \\ 0 & dt & \frac{dt^2}{2} \\ 0 & 0 & t \end{bmatrix} \tag{5.31}$$

$$\mathbf{M} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} \tag{5.32}$$

$$\mathbf{Q}_k = \begin{bmatrix} q_1 dt + \frac{1}{3}q_2 dt^3 + \frac{1}{20}q_3 dt^5 & \frac{1}{2}q_2 dt^2 + \frac{1}{8}q_3 dt^4 & \frac{1}{6}q_3 dt^3 \\ \frac{1}{2}q_2 dt^2 + \frac{1}{8}q_3 dt^4 & q_1 dt + \frac{1}{3}q_2 dt^3 & \frac{1}{2}q_3 dt^2 \\ \frac{1}{6}q_3 dt^3 & \frac{1}{2}q_3 dt^2 & q_3 dt \end{bmatrix} \tag{5.33}$$

For a clock with the following specifications, figure XX shows the accumulated contribution to position error due to the clock

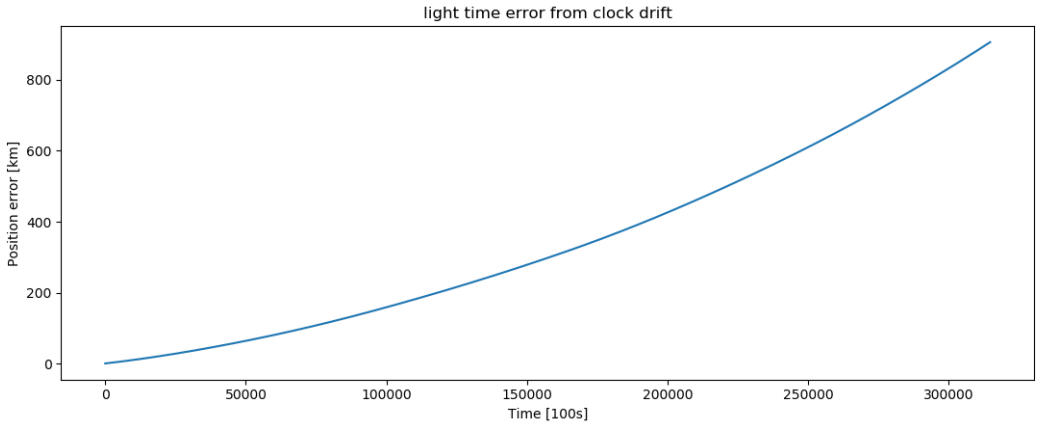


Figure 5.17: Position error due to clock drift over the course of a year

Result Reproducibility and Random Numbers in Python The clock module has its own random number generator for the addition of noise to the clock signal. This affects the reproducibility of results in SAT-ANS when comparing the addition of clock noise to navigation systems without. Random number 'seeds' are used to add pseudorandom stochastic nature to systems such that they may be analysed in a reproducible way. However, the sequence of generated random numbers is ordered (hence the term 'pseudorandom'), which means if the number of operations in the system changes, the specific number chosen in the ordered sequence will change and therefore so will the final result. The the clock adds additional operations as well as redefining the random number seed (although the impact on the specific pseudorandom nature of the system due to this is unclear). This means that singular simulation runs between a navigation system using the noisy clock and those not using it cannot be directly compared. Only a Monte Carlo analysis can be done to properly compare the two.

This result was found when comparing the addition of a noisy clock for sensors not using a clock, and a difference in navigation performance was seen when directly comparing two single simulation runs with the same sensors. This clearly shows that additional computational evaluations of the pseudorandom numbers has an impact on the system.

5.7. INVESTIGATION

With the system defined and pulsar navigation sensors and algorithms implemented, the system can be used to investigate navigation systems using integrated pulsar navigation. To limit the scope of the investigation somewhat, rather than designing a navigation system and testing, the following questions are of note and may be answered with SAT-ANS:

- Does the performance of PNAV change depending on whether in a planetary orbit or in deep space (orbiting the Sun in this case)?
- In these cases, how does the addition of an angle sensor or radial velocity sensor affect the navigation performance?
- What impact does clock noise have on navigation performance?

6

RESULTS

In this chapter the questions asked previously may be answered. Firstly, the influence of sensor noise is investigated on the navigation performance. After this, PNAV is analysed. For this, two test orbital cases are generated - a deep space case, where the Sun is the central orbital body; and a planetocentric orbital case, where the Earth is the central orbital body, with the s/c having the same orbital characteristics as the test case used in the software development section. Further the addition of clock error is compared (specifically for PNAV) to a case where clock error is completely accounted for - the upper limit on performance due to time.

6.1. NON PULSAR NAVIGATION - INFLUENCE OF SENSOR NOISE

Firstly, a comparison is made between a navigation system using very noisy sensors, and those which are very precise. This will show the influence of sensor noise - particularly on the covariance estimation. The

Parameter	Low noise	High Noise
Integration time [s]	200,000	200,000
Timestep [s]	100	100
Filter sigma	1	1
radial velocity sensor std [m]	1×10^{-4}	1×10^{-14}
Angle std [rad]	1×10^{-3}	1×10^{-13}

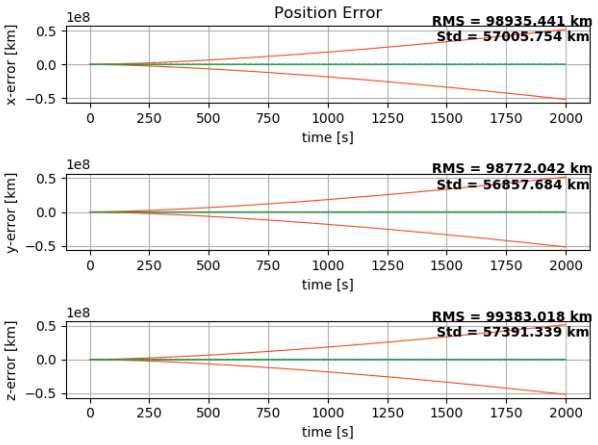
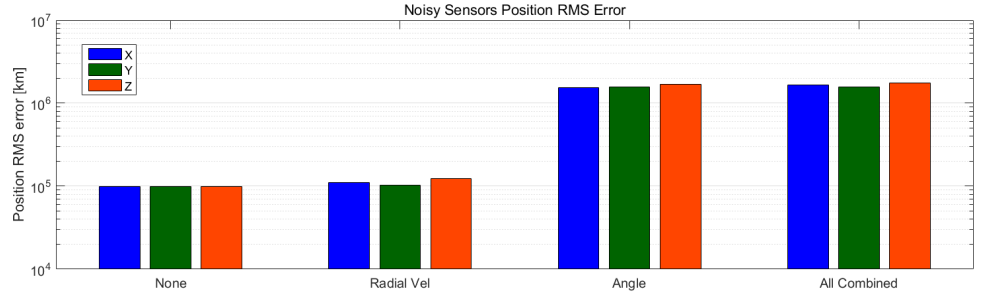
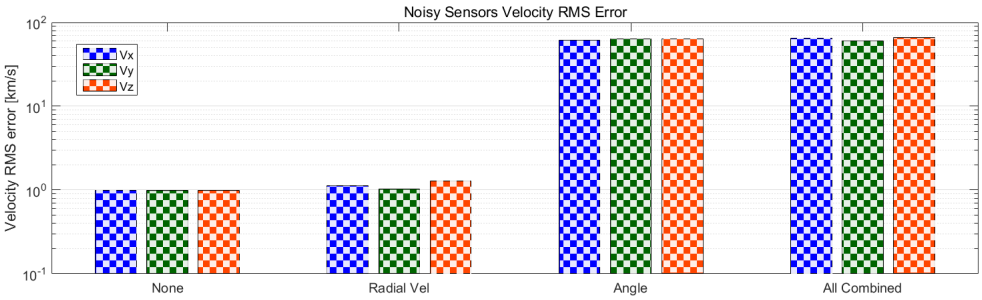


Figure 6.1: Propagated filter dynamical equation error for deep space case

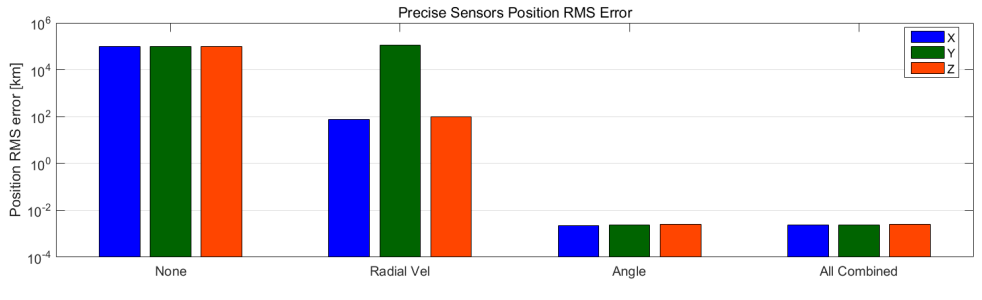


(a)

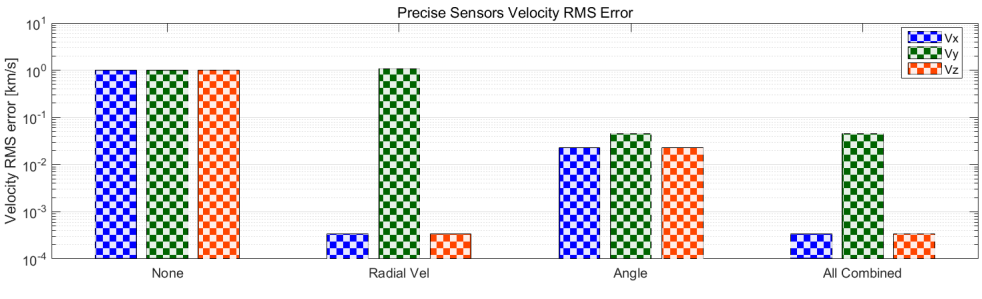


(b)

Figure 6.2: Position and velocity RMS error for deep space case with noisy sensors.



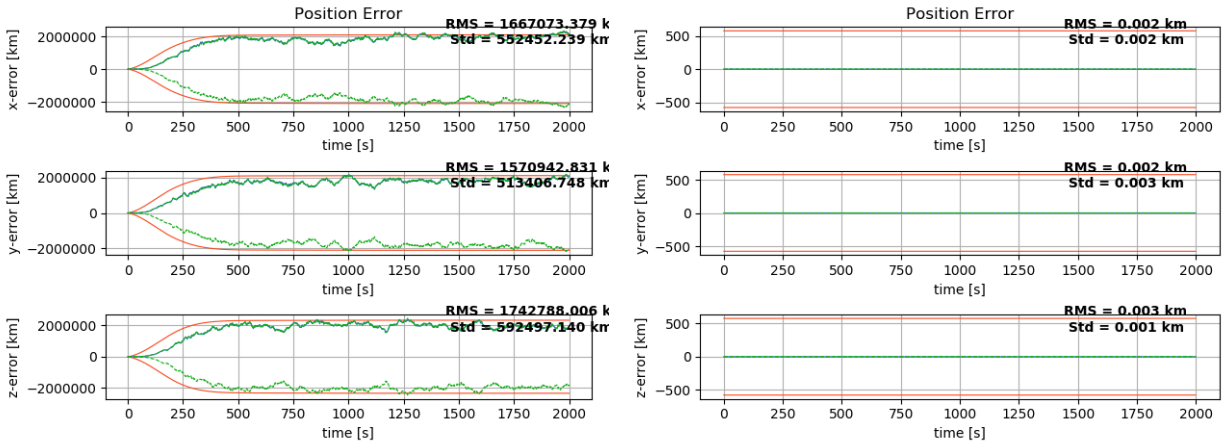
(a)



(b)

Figure 6.3: Position and velocity RMS error for deep space case with precise sensors.

As may be expected, for the same filter process noise covariance, reducing the noise due to the sensors, dramatically improves the navigation performance. However, notice that the error in the Y direction is not improved with additional accuracy with the radial velocity sensor. This is due to the absence of an observable to provide observation information in this direction.



(a) Integrated navigation with noisy sensors

(b) Integrated navigation with low noise sensors

Figure 6.4: Integrated navigation with noisy and non-noisy angle sensor and radial velocity sensor

Further, when the sensor noise is very small, the covariance estimate becomes limited by the process noise, even though the state error is much less than this.

6.2. PULSAR TESTS

There are two cases tested in this work. The orbital conditions used in the simulations are the following:

Parameter \ Case	LEO	Deep Space
Orbital Body	Earth	Sun
Simulation date [tdb]	2018-01-01 00:00	2018-01-01 00:00
Semimajor axis	7136.6 km	1 AU
Eccentricity [-]	0.3	0.0167
Inclination [deg]	90	0
Right ascension of ascending node [deg]	175	0
Argument of perigee [deg]	90	0
True Anomaly [deg]	178	0

For the PNAV tests with additional sensors, the influence of sensor noise is assessed in addition to clock noise. For this, the noise two cases of varying noise levels of sensors are implemented:

Table 6.1: Two cases for sensor noise in the PNAV tests

Sensor	Nominal case	Low noise case
radial velocity sensor	1 nm	0.1 fm
Angle Sensor	1×10^{-6} rad	1×10^{-13} rad
Navigation process noise	1×10^{-5}	1×10^{-5}

The higher noise level is chosen to somewhat reflect the quality of the sensors which are available today [REF], and the low noise should show the impact of pulsar navigation with excellent additional measurements.

For the XNAV sensor, a detection area of 1 m^2 and an integration time of 1500 s are used.

For the RNAV sensor, the following sizing parameters have been used:

Table 6.2: RNAV sensor sizing parameters

Parameter	Value
polarizaton parameter	0.5
Detection area	100 m^2
Receiver central frequency	1.4 GHz
Bandwidth	400 MHz
Main sidelobe attenuation	-40 dB
Receiver noise temperature	15 K
Integration time	1500 s

To analyse the impact of clock noise, the following clock parameters are used:

Table 6.3: Clock model parameters

State	Error	Error rate	Rate drift
Initial State	6×10^{-6} s/s	3×10^{-9} s/s^2	6×10^{-11} s/s^3
Noise spectral index	1.11×10^{-9} s/s	2.22×10^{-20} s/s^2	6.66×10^{-22} s/s^3

The navigation system outputs a large amount of data which is difficult to display in a concise manner. To overcome this, the data displayed in this section, will show representative cases, in addition to the mean of the normalised RMS state position and velocity components for the different sensor combinations tested. Due to the periodic nature of especially the LEO case, the RMS is deemed to be a better parameter for the comparison of the different cases. Further, this allows the immediate comparison of the different sensor noise groups as well as the addition of clock error.

The full representative graphs and tablular data is be available in appendix F for reference.

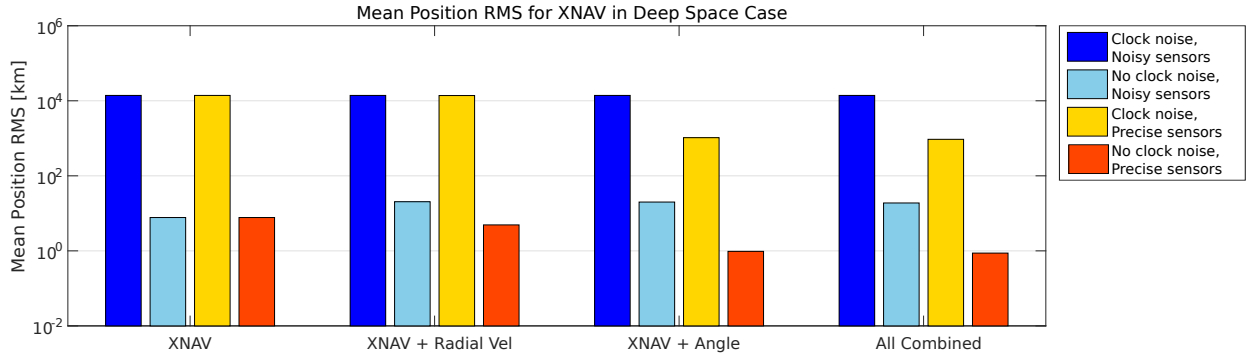
6.2.1. DEEP SPACE CASE

The first case is the deep space or interplanetary case. Note that the observables for the sensors as mentioned previously are assumed to remain fixed relative to the s/c

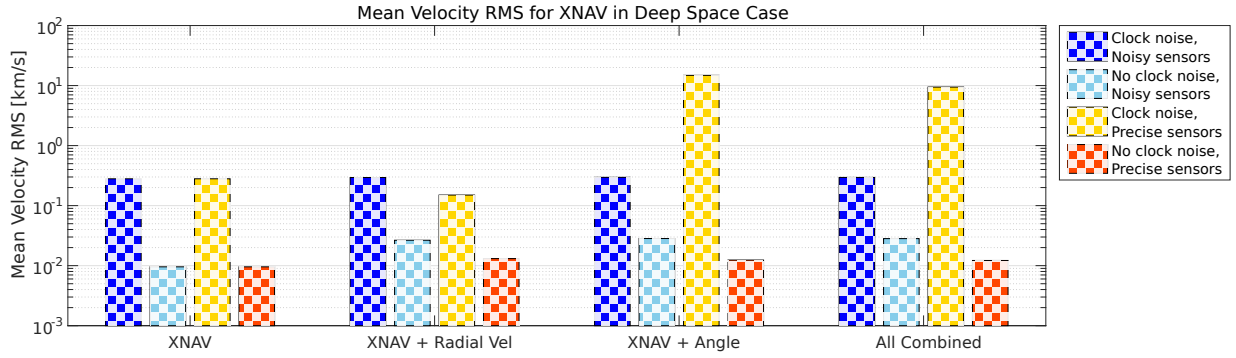
XNAV

There are points of note from the graphs shown in figure 6.5. There is no clear trend that with additional sensors comes a better navigation solution. Specifically for noisy sensors, the performance actually degrades with additional sensors - both for position and velocity estimation. Indeed, when not considering clock noise, the position estimation with XNAV alone is 7.73 km, and with integrated angle and radial velocity sensors, it is 18.9 km, which is over 144% worse. A similar trend is seen for velocity - from 9.56×10^{-3} km/s to 2.82×10^{-2} km/s (194% worse). The degradation is somewhat limited when clock noise is considered - 0.05% for position and 5% for velocity.

When precise sensors are used, the position estimation improves with additional sensors - both with clock errors (89%) and without (93%). Velocity estimation on the other hand sees some strange behaviour. Both see a performance degradation. Without clock error it is 27% worse, and with clock error it is over 3000% worse. This is an interesting result and shows that the RMS graphs alone perhaps do not make the best comparative metric. If the errors with covariance as a function of time are considered, as shown in figures 6.6 which shows velocity RMS for precise sensors with clock error. It can be seen that every update time of the XNAV sensor, the error in velocity greatly changes. After an update, and in the absence of the XNAV measurement, the velocity estimation has input from only the angle and radial velocity sensor, and returns to a more reasonable value (preventing the propagation of the position error). This indicates that a precise sensor with unmodelled noise may not completely be accounted for by improving the accuracy of the additional sensors. However, the question still remains why this effect is so prominent when precise angle sensors are used.



(a)



(b)

Figure 6.5: XNAV mean RMS errors for the deep space case

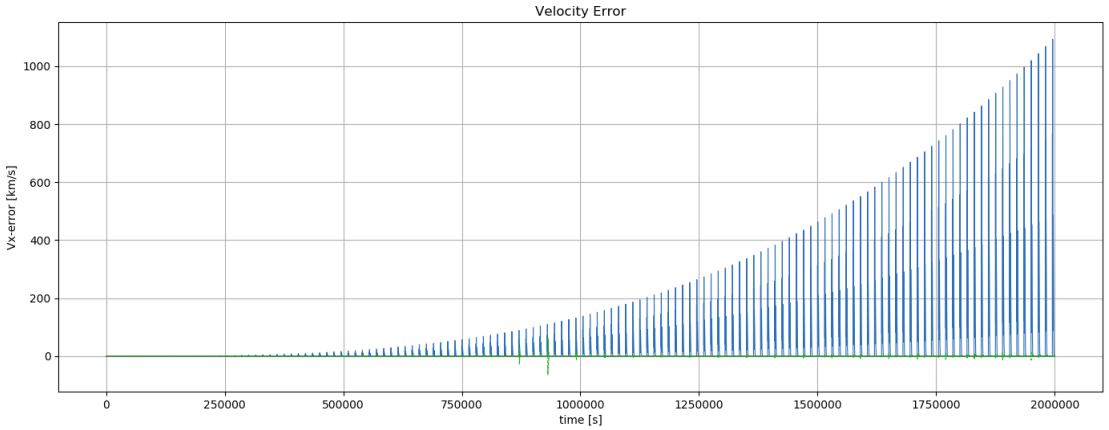
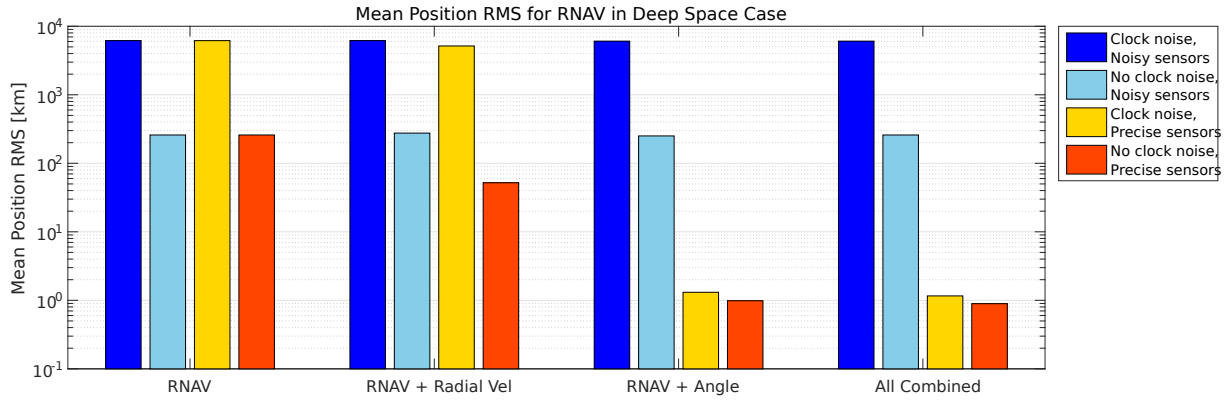
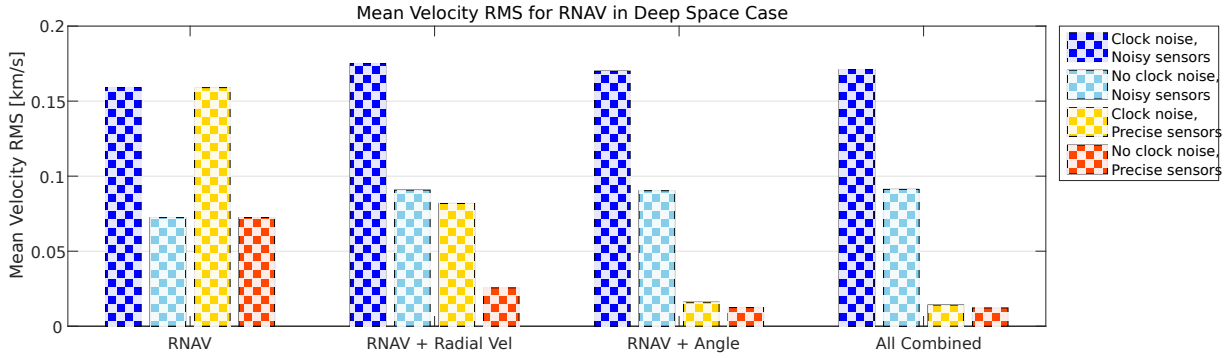


Figure 6.6: Velocity RMS for Integrated XNAV for deep space case with clock error. The increase of the velocity error is periodic with the update rate of the XNAV sensor.

RNAV



(a)



(b)

Figure 6.7: RNAV mean RMS errors for deep space case

For RNAV, noisy sensors also lead to a performance degredation in general. When clock noise is not considered, the mean RMS velocity estimate increases from 7.42 $\times 10^{-2}$ km/s, to 9.12 $\times 10^{-2}$ km/s (26%), although the position estimation remains broadly similar with a loss using integrated sensors compared to RNAV alone of 0.1%. When clock noise is considered on the other hand, the position estimation error improves by 1.7%, with a velocity degredation of 7.8%. For the precise sensors, large performance improvements are seen in both position and velocity, both with clock noise (99.98% position and 91.1% velocity), and without (99.7% position and 83.0% velocity). It is clear from these results that the RNAV sensor is less accurate than the XNAV sensor and from this, that additional sensors will lead to larger improvements (or smaller degredations).

6.2.2. CLOCK NOISE

Although clock noise is added to the graphs in the previous section, it is interesting to see the impact it has over the mean profiles of the XNAV and RNAV cases for deep space.

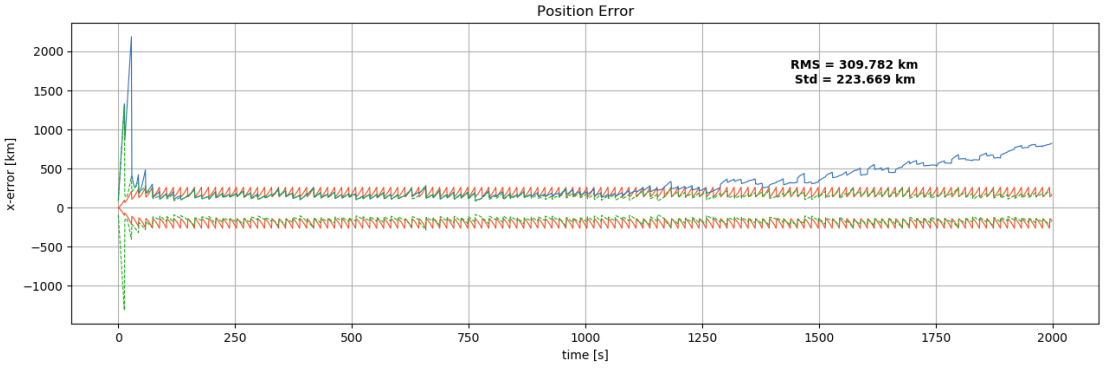
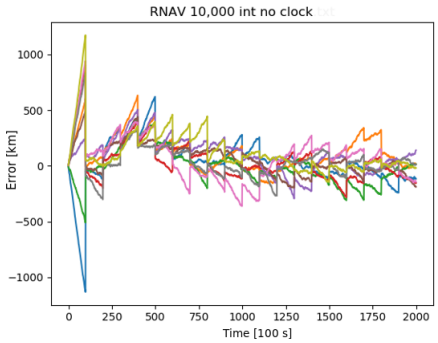
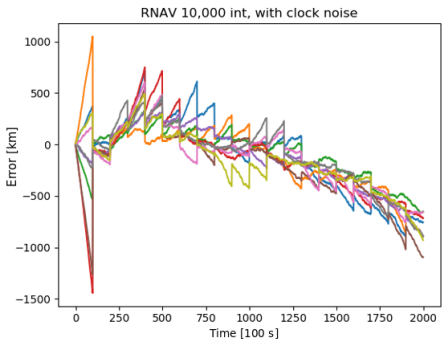


Figure 6.8: RNAV mean x-state error as a function of time

Figure 6.8 shows that the filter-estimated covariance (red) does not account for the increased error.

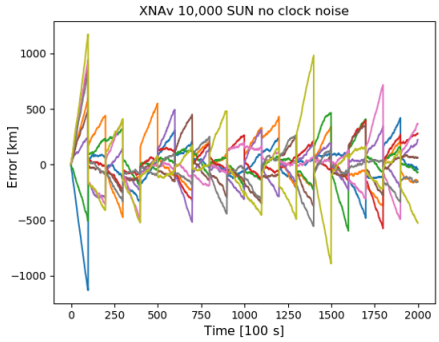


(a) RNAV sensor in a 1 AU orbit without clock noise

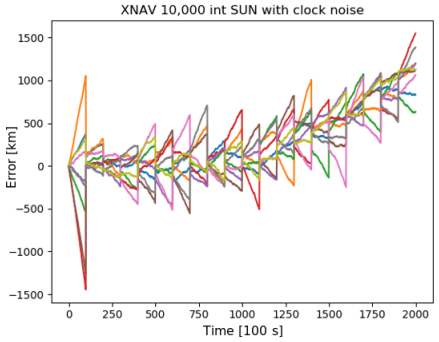


(b) RNAV sensor in a 1 AU orbit with clock noise

Figure 6.9: Example of addition of clock noise for RNAV



(a) XANV sensor in a 1 AU orbit without clock noise



(b) XANV sensor in a 1 AU orbit with clock noise

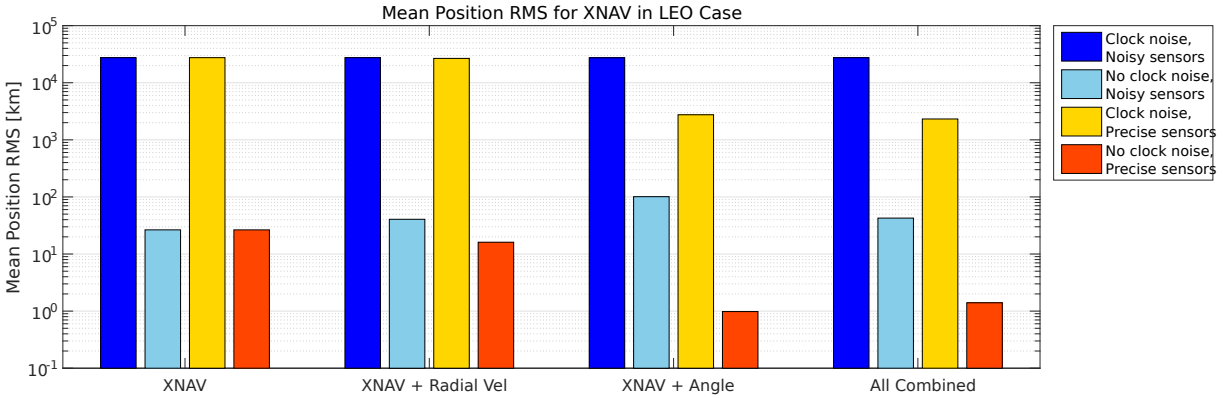
Figure 6.10: Example of addition of clock noise for XNAV

As the figures above show, the state error drifting. In this case the x-component is graphed, but all positional components drift over time when pulsar navigation is used in combination is additional clock error. An interesting observation is the sign difference in drift between XNAV and RNAV.

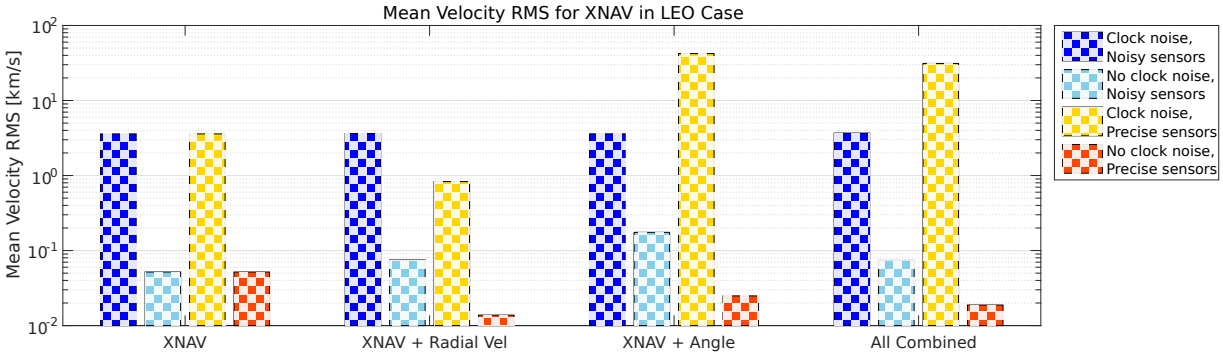
6.2.3. PLANETARY ORBIT CASE

The planetary orbit case was chosen to to be a low Earth orbit case.

XNAV



(a)

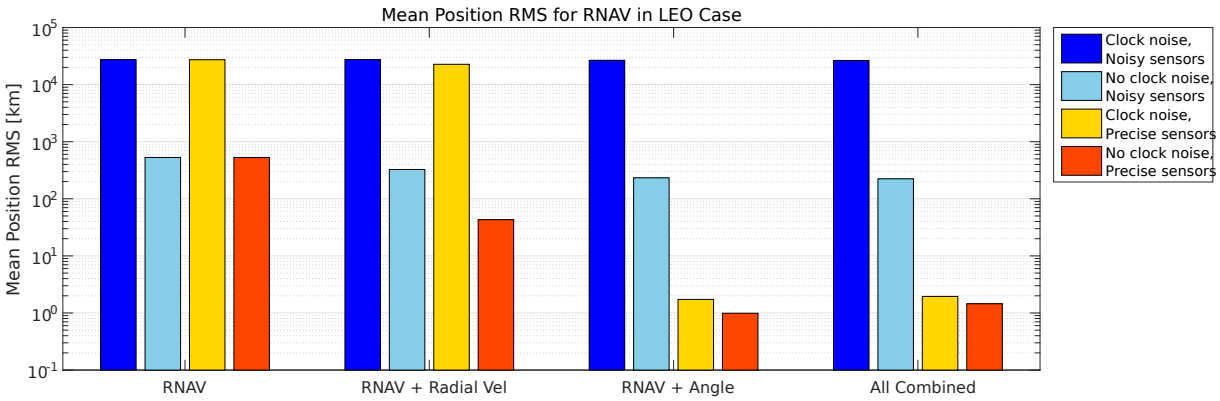


(b)

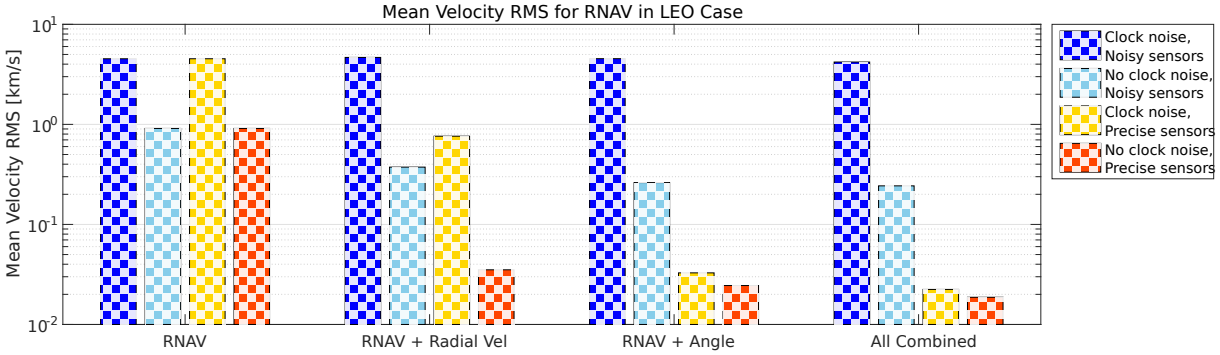
Figure 6.11: XNAV mean RMS errors for the LEO case

Compared to the Deep space case, the degredation when considering clock noise is similar with noisy sensors - 0.1 % position and 2.5% for velocity. The degredation is much lower when clock noise is not considered - 60.6% loss for position and 42.6% velocity. For precise sensors, there is a large performance improvement when clock noise is not considered - 94.7% position and 63.8% velocity, and again when clock noise is considered the position solution improves by 91.6%, but the velocity estimation degrades by 766%. This is an improvement albeit still a very large difference from the XNAV-only case. Again this change is attributed to the update rate differences between the pulsar sensor and the angle sensor - but it requires further investigation.

RNAV



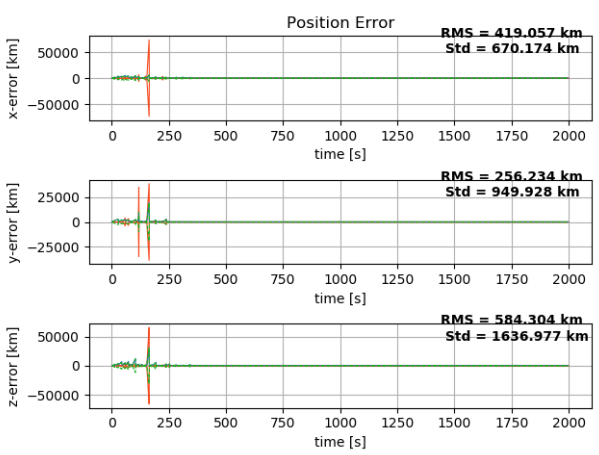
(a)



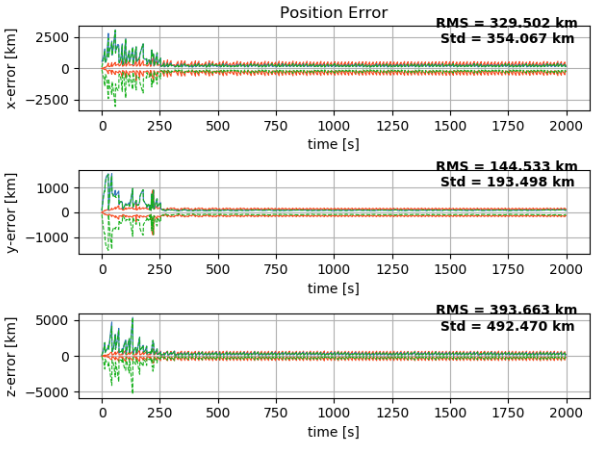
(b) RNAV mean normalised RMS velocity error

Figure 6.12

For the LEO case with RNAV shows an improvement of 2.9% and 7.3% for position and velocity respectively when noisy sensors are added with included clock noise. larger improvements are seen when clock noise is not added - 57.9% and 73.3% respectively. Similar large improvements are seen as with the Deep space case for the precise sensors.



(a) RNAV sensor only



(b) Integrated RNAV with radial velocity sensor and angle sensor

Figure 6.13: RNAV position errors for the LEO space case

Similar to the XNAV case, the RNAV-only test compared to integrated navigation with noisy sensors again shows the perturbations, which is somewhat abated in the integrated case. Notice however that there is some oscillation at the

start of the simulation before the filter seems to converge to a steady state.

7

DISCUSSION, CONCLUSIONS AND RECOMMENDATIONS

7.1. DISCUSSION

This section will discuss both the investigation questions and further observations made in the results section.

7.1.1. INFLUENCE OF ORBIT ON PNAV

Two cases were investigated - a reference planetary orbit, in this case a low Earth orbit; and a deep space orbit - a s/c in a solar orbit with the same orbital characteristics as the Earth. In both XNAV and RNAV the navigation system was found to be better in the deep space case. Specifically, for XNAV-and RNAV-only the following results were found:

Table 7.1: Mean PNAV-Only results

Case	LEO		Deep Space	
	Mean Pos error [km]	Mean Velocity error [km/s]	Mean Pos error [km]	Mean Velocity error [km/s]
XNAV-Only	26.5	0.052	7.7	0.0096
XNAV-Only with clock	27500	3.6	13900	0.28
RNAV-Only	530	0.91	259	0.072
RNAV-Only with clock	27300	4.53	6170	0.159

This result can be attributed to the smaller forces acting on the s/c in the deep space environment, and from this the smaller gradients in velocity. This in turn could allow for a longer integration time from the PNAV sensor - thereby leading to a more precise navigation result. Additionally, as the PNAV sensor beacons are effectively position independent - their use for deep space is well motivated. The long integration times required in an environment where the state is rapidly changing, the accumulation of error due to dynamical propagation is more difficult to offset with precise observation. What was however further established is that for the sizing parameters chosen, the XNAV sensor performed better.

Note however, that only a single algorithm has been implemented in SAT-ANS and indeed with the lowest fidelity pulsar model. If techniques such as phase tracking of pulsars are used or absolute navigation, then different performances may be achieved.

7.1.2. INFLUENCE OF ADDITIONAL SENSORS

The orbital environment also changes the influence of additional sensors in the integrated navigation. Specifically for noisy sensors - those in this case were a spectrometer with a 1 nm accuracy and an angle sensor with 1 μrad accuracy, the performance gain when in a deep space environment was shown to be in general detrimental to the navigation.

This reduction in the performance with the XNAV sensor was also observed with the integration of noisy sensors in the LEO environment. However with additional clock noise - the additional sensors had a minimal positive impact. Compared to RNAV, this may be explained by the general better performance of this sensor - leading to minimal performance degradation with additional sensors. This implies then that the performance attained by the XNAV sensor is below the noise threshold of the additional sensors and that their implementation only serves to worsen the performance.

However for RNAV performance improvements are seen (and in the clock case with XNAV as mentioned). Further, the state estimation is drastically improved with the precise additional sensors - also in the deep space case. This

then shows that the impact of the additional sensors is dependent on their noise contribution relative to the other sensors, and that arbitrarily adding more sensors will not necessarily improve performance.

From this, it may be possible to generate a design space based on the capabilities of sensors to show that for certain navigation performance requirements, certain threshold sensor noises must be met - thereby defining the sensors required for use.

7.1.3. CLOCK NOISE

The addition of clock noise when combined with the use of pulsar navigation leads to a drift in the estimated state - leading to increasing errors as a function of time. This error comes from the pulsar sensor bias in the detection of the arrival pulses. There may be an additional effect which is the barycentering of the spacecraft state. When the estimated s/c state is transformed from the planetocentric inertial frame, to the barycentric one - for use in pulsar detection, the current time is used. If the clock drifts, the position of the reference orbital body relative to the barycentre will be incorrect - thereby leading to another cumulative error. Although further investigation of this effect is required to quantify, the effect should be more pronounced for an Earth-centered orbit than for a deep space one - as the Earth has a larger velocity relative to the SSB compared to the Sun. This cannot be verified using the data acquired due to the influence of filter parameters and orbital noise.

Further, the covariance of the filter does not change when clock error is added. This is understandable as the clock error is not estimated in the state. Due to this, the weighting on the PNAV sensors remains the same - further degrading the navigation performance. Although precise additional measurements can counter this effect (although possibly leading to velocity estimation issues as with XNAV)- it then leads the PNAV sensor to be superfluous. An interesting aspect would be to add the clock error state to the overall state estimation and to investigate performance with additional sensors. In the LEO case especially - the performance could tend to that without clock noise.

7.2. CONCLUSION

For this work, the following research aims were generated:

Develop a general autonomous navigation simulator testbed for spacecraft, capable of quantifying positional navigation accuracy.

Identify the most likely candidates for autonomous navigation sensors and quantify the positional navigation accuracy of a user defined subset of models of these.

Extend the tool to be capable of user-defined navigation filters and orbital conditions.

Further, as pulsar navigation is likely to be part of future autonomous navigation systems, an investigation using the tool is performed, and the aim generated:

Develop pulsar navigation sensor models and use the software tool to investigate the impact of fusing additional sensor information on positional navigation performance.

To answer the first research aim, the Spacecraft Analysis Tool for Autonomous Navigation and Sizing was developed.

7.2.1. SAT-ANS

A software tool SAT-ANS has been developed to aid in the design and analysis of autonomous navigation systems. Designed in a modular way, the tool contains four main components:

- Orbital Module
- Sensor Module
- Navigation Module
- Analysis Module

which allow a user to design a Keplerian orbit around the Sun, Mars or the Earth at a specific date and time. This orbit can then be propagated with a choice of integrator with optional additive noise to provide the reference true state of the spacecraft. The navigation system of the s/c is comprised of sensors and a navigation filter, each defined in their own module. The sensor module houses the defining equations, sizing parameters and noise characteristics of the sensors and the navigation module houses the filter. The filter combines a dynamical model with the multi-sensor input to provide an estimated state. The implemented filter to date is a verified unscented kalman filter Additionally, the navigation module contains a model of the spacecraft on-board clock which can drift over time -

allowing time-based navigation techniques to be assessed. The analysis module is a current work in progress and will seek to calculate the CRLB for the chosen sensor combination, however to date it runs Monte Carlo simulations of the chosen navigation system and trajectory.

The quantification of accuracy was chosen to be the mean RMS normalised position and velocity error with respect to the true state of the navigation system from a Monte Carlo analysis. Further, the extenability of the system has been shown at least with respect to the sensors through the addition of pulsar sensors.

For the second aim, two sensors were chosen: an angle sensor and a radial velocity sensor.

For the third aim, different orbital conditions can be used in SAT-ANS, for example the orbital bodies of the Sun, Earth and Mars have been implemented for Keplerian orbits.

7.2.2. PNAV

The second aim was investigated through the implementation of pulsar sensors and a detextion algorithm:

SAT-ANS was tested with Pulsar navigation. Two separate detection methods were developed according to the common detection EM bands: X-ray and radio. Libraries of pulsars were constructed and a low fidelity model was then developed for use with the delta-correction algorithm to test PNAV performance. It was found that the orbit of the s/c affects the impact of additional sensors to both XNAV and RNAV. Further the impact of clock noise was shown to be an issue which will degrade performance.

Overall, this tool has shown the efficacy of integrated pulsar navigation in a planetary orbital environment and in deep space. With some further development, SAT-ANS has the potential for use in preliminary studies of autonomous navigation system design, and can be expanded for further functionality.

7.3. RECOMMENDATIONS FOR FUTURE WORK

There are two parts to the recommendations for future work: those which are relevant to development in SAT-ANS in general for further functionality, and those which further investigate pulsar navigation with the tool.

7.3.1. POSITION-VELOCITY DISCREPANCY IN XNAV

It was found that the addition of low noise sensors to the XNAV sensors with the addition of clock noise caused a very large change in the velocity estimation. This was attributed to the weighting of the XNAV sensor which remained constant in the presence of clock noise. However, it is not understood why there is such a large difference between the high and low sensor case when an angle sensor is added. This should be investigated.

7.3.2. INSTABILITIES IN THE FILTER

It was observed both in some RNAV LEO cases that glitches occurred in the PNAV-only sensor. These glitches however were no longer present when additional sensors were added. The addition of the sensors is then attributed to the removal of the glitching. It is then postulated that if an intermittent noisy sensor input is given to the navigation filter and a state estimate based on the noisy sensor input is propagated, this could lead to instabilities. Interestingly these glitches are only observed at the beginning of the simulation run. As the navigation filter reaches a steady state, it may then be more sensitive to these sorts of irregularities between sensor input and propagation. This idea is backed-up by the reduction of the perturbations in the RNAV-case, as there is higher frequency sensor update, leading to the correction of any irregularities in propagation.

7.3.3. SAT-ANS DEVELOPMENT

The next step for this model, is an overall verification campaign which uses a true trajectory of a spacecraft, its sensors and navigation filter, and the navigation performance it achieved. Once this has been confirmed, its use as a system analysis tool will be confirmed.

SYSTEM DESIGN

The Sizing component of SAT-ANS has not been developed in this work, and could be an interesting avenue for investigation. This may be done against specific requirements for system sizing parameters such as mass and power in addition to navigation performance. For this, more sizing parameters should be included with the sensors themselves with scaling factors, such that the system scales with the sizing parameters. This, in turn would validate requirement **SAT-F-02**.

Further, if mission being analysed has multiple phases, all phases of the mission could be analysed with the chosen sensor combinations and the optimal configuration chosen. With the inclusion of boundary conditions for sensors

and system parameters, a list of real sensors could be generated which best fits the system requirements set by the user.

The future development for the tool may be broken into their constituent modules.

ORBITAL MODULE

For this work, only Keplerian orbits have been considered for use without perturbations. To generate a more realistic true state of the s/c, true perturbations may be added such as the J2 and higher order terms. Additionally, sensors are implemented which observe beacons fixed on the surface of an orbital body, additional reference frames may need to be added, such as the planet-centered, planet fixed. Further along these lines, a model for the s/c attitude could be included such that positions and pointing of sensors could be modelled - which could then constrain the observation of specific observables.

SENSOR MODULE

For the first implementation of the sensor observables, a simple library was created, from which a series of observables were chosen before the simulation. A more realistic scenario is that the full list of observables is available to the sensor-navigation system combination and that based on the current (estimated) state an perhaps attitude, only certain observables will be visible. In line with this, the central orbital body is currently considered a point mass, however in reality there will be eclipses of observables due to the reference body. If this is implemented, then the optimisation of specific observables may be done - such the the most ideal element is chosen for the state estimation.

Further and specifically for the angle sensor but to a lesser extent the pulsar sensors; the observables have been considered stationary over the course of the simulation. This was motivated for the pulsar sensors, but for the angle sensors - the position of the beacons will change over time. To account for this, an ephemeris of the observables may be added, or specific models of their motion.

For the spectrometer, rather than just observing a single spectral line, modelling the spectrum of stellar sources within some bandwidth would be more realistic.

Finally for the extension of the tool to additional sensors, with the combination of including attitude to the model (and then in principle the state estimation), the a type of sensor which would be interesting to investigate is the true vision-based sensor - where the features are being observed. Although the implementation of this would require some work, it would extend the capabilities of the tool such that advancements in optical space navigation could be tested.

NAVIGATION MODULE

The navigation module has been implemented with just one filter - the unscented Kalman filter. This was chosen as a good compromise between navigation performance, the ability to cope with non-linearities and efficiency (the time to produce a state estimate). However, for such a general filter, ideal navigation performance may not be reached. The next steps could be to investigate the impact of different filters on navigation performance. Additionally, to validate requirement **SAT-F-01.03**, an additional filter must be added verified and tested.

As was shown in the PNAV section, if the on-board clock is noisy, then time-based navigation techniques will degrade over time. A solution to this may be to include the clock state into the state estimation within the filter, and assess the impact that this will have.

ANALYSIS MODULE

The analytical posterior CRLB was attempted to be implemented into SAT-ANS, however this was not possible, and thought to be due to the validity of the assumptions made with respect to noise. As the CRLB would be a valuable upper-performance bound indicator, this would be a worthwhile avenue for further investigation. Although the MC analysis technique seems the most likely to produce a result, if an analytical solution could be found, this would reduce operational time.

7.3.4. PULSAR NAVIGATION

For pulsar navigation, the low fidelity model has been implemented, and from this the delta correction method. The next logical step would be to look at the absolute navigation with this model. After this, the modelling of more realistic pulsar signals could be done, using the pulsar profiles. This would allow the true phase of specific signals to be calculated from the start of the simulation, and propagated in time and space from the SSB to the s/c.

With the implementation of a signal model for the pulsars, more realistic detection method could be implemented to include signal folding. Additionally timing noise could be added to the pulsar signals themselves. With respect to

the determination of which pulsars to observe, an optimisation algorithm could be implemented based on the visibility of specific pulsars at certain times, the covariance of certain state components, and the relationship between observation time and PNAV sensor covariance.

Finally, with the result of the varying impact of additional sensors depending on the orbital situation, the investigation of a design space for a improvement factor over PNAV-only navigation (with certain sensor and navigation sizing parameters), what sensor noise would be required, would be an interesting avenue for further research.

REFERENCES

[1] V. S. Beskin, S. V. Chernov, C. R. Gwinn, and A. A. Tchekhovskoy, *Radio pulsars,* Space Science Reviews **191**, 207 (2015).

[2] R. N. Manchester, G. Hobbs, A. Teoh, and M. Hobbs, *The australia telescope national facility pulsar catalogue,* (2005), `http://www.atnf.csiro.au/people/pulsar/psrcat/`.

[3] ECSS, *ECSS-E-TM-10-25A 2010 Engineering design model data exchange CDF,* , 31 (2010).

[4] M. Bandecchi, B. Melton, B. Gardini, and F. Ongaro, *The ESA / ESTEC Concurrent Design Facility,* EuSEC 2000 , 330 (2000).

[5] P. D. L. B. integration_kalman Groves, *Principles of GNSS, inertial, and multisensor integrated navigation systems* (Artech house, 2013).

[6] E. A. Wan and R. Van Der Merwe, *The unscented Kalman filter for nonlinear estimation,* in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000* (Ieee, 2000) pp. 153–158.

[7] T. E. Bell, *Planets Amidst the Noise,* AstroShort (2013).

[8] S. Shemar, G. Fraser, L. Heil, D. Hindley, A. Martindale, P. Molyneux, J. Pye, R. Warwick, and A. Lamb, *Towards practical autonomous deep-space navigation using x-ray pulsar timing,* Experimental Astronomy **42**, 101 (2016).

[9] W. Becker, M. G. Bernhardt, and A. Jessner, *Autonomous spacecraft navigation with pulsars,* arXiv preprint arXiv:1305.4842 (2013).

[10] I. Stairs, S. Thorsett, and F. Camilo, *Coherently dedispersed polarimetry of millisecond pulsars,* The Astrophysical Journal Supplement Series **123**, 627 (1999).

[11] D. Brito, G. Tavares, J. Fernandes, A. Noroozi, and C. Verhoeven, *Radio pulsar receiver systems for pulsar navigation,* (2015).

[12] J. Sala, A. Urruela, X. Villares, R. Estalella, and J. Paredes, *Feasibility study for a spacecraft navigation system relying on pulsar timing information,* European Space Agency Advanced Concepts (2004).

[13] G. Hobbs, D. R. Lorimer, A. G. Lyne, and M. Kramer, *A statistical study of 233 pulsar proper motions,* Monthly Notices of the Royal Astronomical Society **360**, 974 (2005).

[14] D. Lorimer, J. Yates, A. Lyne, and D. Gould, *Multifrequency flux density measurements of 280 pulsars,* Monthly Notices of the Royal Astronomical Society **273**, 411 (1995).

[15] D. R. Lorimer, *Binary and millisecond pulsars,* Living Reviews in Relativity **11**, 8 (2008).

[16] J. Liu, E. Wei, and S. Jin, *Mars cruise orbit determination from combined optical celestial techniques and x-ray pulsars,* The Journal of Navigation , 1 (2017).

[17] A. J. Jongschaap, *Literature Review: Pulsars - Timing and Navigation,* (2017).

[18] J. W. Mitchell, M. A. Hassouneh, L. M. Winternitz, J. E. Valdez, S. R. Price, S. R. Semper, W. H. Yu, Z. Arzoumanian, P. S. Ray, and K. S. Wood, *Sextant—station explorer for x-ray timing and navigation technology,* AIAA Guidance, Navigation, and Control Conference , 2015 (2015).

[19] S. I. Sheikh, D. J. Pines, P. S. Ray, K. S. Wood, M. N. Lovellette, and M. T. Wolff, *Spacecraft navigation using x-ray pulsars,* Journal of Guidance, Control, and Dynamics **29**, 49 (2006).

[20] T. Mineo, G. Cusumano, L. Kuiper, W. Hermsen, E. Massaro, W. Becker, L. Nicastro, B. Sacco, F. Verbunt, and A. Lyne, *The pulse shape and spectrum of the millisecond pulsar psr j0218+ 4232 in the energy band 1-10 kev observed with bepposax,* Astronomy and Astrophysics **355**, 1053 (2000).

[21] W. Becker, J. Trümper, A. N. Lommen, and D. C. Backer, *X-rays from the nearby solitary millisecond pulsar psr j0030+ 0451: The final rosat observations,* The Astrophysical Journal **545**, 1015 (2000).

[22] W. Becker and J. Trümper, *The x-ray emission properties of millisecond pulsars,* arXiv preprint astro-ph/9806381 (1998).

[23] V. E. Zavlin, *Xmm-newton observations of four millisecond pulsars,* The Astrophysical Journal **638**, 951 (2006).

[24] A. Possenti, R. Cerutti, M. Colpi, and S. Mereghetti, *Re-examining the X-ray versus spin-down luminosity correlation of rotation powered pulsars,* Astronomy & Astrophysics **387**, 993 (2002).

[25] L. M. Winternitz, M. A. Hassouneh, J. W. Mitchell, J. E. Valdez, S. R. Price, S. R. Semper, H. Y. Wayne, P. S. Ray, K. S. Wood, and Z. Arzoumanian, *X-ray pulsar navigation algorithms and testbed for sextant,* in *Aerospace Conference, 2015 IEEE* (IEEE) pp. 1–14.

[26] N. Xiaolin, Y. Yuqing, G. Mingzhen, W. Weiren, F. Jiancheng, and L. Gang, *Pulsar navigation using time of arrival (toa) and time differential toa (tdtoa),* Acta Astronautica (2017).

[27] ESA, *Estrack ground stations,* Accessed on 05/10/2018.

[28] B. W. Ashman, J. J. Parker, F. H. Bauer, and M. Esswein, *Exploring the limits of high altitude gps for future lunar missions,* (2018).

[29] J. Yim, J. Crassidis, and J. Junkins, *Autonomous orbit navigation of interplanetary spacecraft,* in *Astrodynamics Specialist Conference* (2000) p. 3936.

[30] F. H. Bauer, K. Hartman, and E. G. Lightsey, *Spaceborne gps current status and future visions,* in *Aerospace Conference, 1998 IEEE*, Vol. 3 (IEEE) pp. 195–208.

[31] X. P. Deng, G. Hobbs, X. P. You, M. T. Li, M. J. Keith, R. M. Shannon, W. Coles, R. N. Manchester, J. H. Zheng, X. Z. Yu, D. Gao, X. Wu, and D. Chen, *Interplanetary spacecraft navigation using pulsars,* Advances in Space Research **52**, 1602 (2013).

[32] C. L. Thornton and J. S. Border, *Radiometric tracking techniques for deep-space navigation,* (2003).

[33] J. Berner, T. Pham, A. Bhanji, and C. Scott, *Deep space network services catalog,* Deep Space Network, Jet Propulsion Laboratory, California Institute of Technology, Rev. F (2015).

[34] D. R. Lorimer and M. Kramer, *Handbook of pulsar astronomy,* **4** (2005).

[35] Y. Wang, W. Zheng, S. Sun, and L. Li, *X-ray pulsar-based navigation using time-differenced measurement,* Aerospace Science and Technology **36**, 27 (2014).

[36] S. I. Sheikh, A. R. Golshan, and D. J. Pines, *Absolute and relative position determination using variable celestial x-ray sources,* in *30th Annual AAS Guidance and Control Conference*, pp. 855–874.

[37] J. Sala, A. Urruela, X. Villares, J. Romeu, S. Blanch, R. Estalella, and J. M. Paredes, *Pulsar navigation,* Acta Futura **3**, 94 (2008).

[38] R. Regulations, *International telecommunication union,* Radiocommunication Sector. ITU-R. Geneva (2016).

[39] W. H. Yu, *Application of X-ray pulsar navigation: A characterization of the Earth orbit trade space*, Thesis (2015).

[40] Y. Wang, W. Zheng, X. An, S. Sun, and L. Li, *Xnav/cns integrated navigation based on improved kinematic and static filter,* The Journal of Navigation **66**, 899 (2013).

[41] Y. Wang, W. Zheng, and S. Sun, *X-ray pulsar-based navigation system/sun measurement integrated navigation method for deep space explorer,* Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering **229**, 1843 (2015).

[42] J. Liu, E. Wei, and S. Jin, *Mars cruise orbit determination from combined optical celestial techniques and x-ray pulsars,* Journal of Navigation **70**, 719 (2017).

[43] Y. Wang, W. Zheng, X. An, S. Sun, and L. Li, *XNAV/CNS integrated navigation based on improved kinematic and static filter,* Journal of Navigation **66**, 899 (2013).

[44] F. Torre, M. Vasile, R. Serra, and S. Grey, *Autonomous Navigation of a Formation of Spacecraft in the Proximity of a Binary Asteroid,* in *International Symposium on Space Technology and Science* (2017).

[45] S. R. Steffes and G. Barton, *Deep Space Autonomous Navigation Options for Future Missions,* in *AIAA SPACE and Astronautics Forum and Exposition* (2017) p. 5369.

[46] E. Secretariat, *ECSS-M-ST-40C Rev. 1*, Tech. Rep. March (ESA-ESTEC, 2009).

[47] A. Fedele, G. Guidotti, G. Rufolo, G. Malucchi, A. Denaro, F. Massobrio, S. Dussy, S. Mancuso, and G. Tumino, *The Space Rider Programme: End user's needs and payload applications survey as driver for mission and system definition,* Acta Astronautica , 0 (2018).

[48] F. Gandía, A. Paoletti, A. Tomassini, M. Sagliano, and F. Ankersen, *GNCDE: exploiting the capabilities of development environments for GNC design,* in *4th International Conference On Spacecraft Formation Flying Mission & Technologies (SFFMT)* (2011).

[49] J. M. R. Martin, F. Torre, M. Vetrisano, and M. Vasile, *ATHENA: Astrodynamics Toolbox for High-Fidelity Error and Navigation Analysis,* .

[50] R. F. Sunseri, H.-C. Wu, S. E. Evans, J. R. Evans, T. R. Drain, and M. M. Guevara, *Mission analysis, operations, and navigation toolkit environment (monte) version 040,* (2012).

[51] ECSS-E-ST-40 Working Group, *ECSS-E-ST-40C Space engineering Software,* , 206 (2009).

[52] K. Kumar, Y. Abdulkadir, P. W. L. van Barneveld, F. Belien, S. Billemont, E. Brandon, M. Dijkstra, D. Dirkx, F. Engelen, and D. Gondelach, *Tudat: a modular and robust astrodynamics toolbox,* in *Fifth ICATT, International Conference on Astrodynamics Tools and Techniques* (ESA Noordwijk, 2012) pp. 1–8.

[53] N. C. Mohanty, *Autonomous Navigation for High Altitude Satellites,* Information Sciences **30**, 125 (1983).

[54] Y. Gao, *Contemporary Planetary Robotics: An Approach Toward Autonomous Systems* (John Wiley and Sons, 2016).

[55] L. Q. Gothard and J. Rosen, *Encyclopedia of physical science,* (2010).

[56] G. R. Gladstone, S. C. Persyn, J. S. Eterno, B. C. Walther, D. C. Slater, M. W. Davis, M. H. Versteeg, K. B. Persson, M. K. Young, G. J. Dirks, *et al.*, *The ultraviolet spectrograph on nasa's juno mission,* Space Science Reviews **213**, 447 (2017).

[57] A. Kucherov and V. Kurenkov, *Use of cluster analysis for development of star tracker mass statistical model,* Procedia engineering **185**, 227 (2017).

[58] A. Valade, P. Acco, P. Grabolosa, and J.-Y. Fourniols, *A Study about Kalman Filters Applied to Embedded Sensors,* Sensors (Basel, Switzerland) **17**, 2810 (2017).

[59] H. Musoff and P. Zarchan, *Fundamentals of Kalman filtering: a practical approach* (American Institute of Aeronautics and Astronautics, 2009).

[60] M. Veth, *Nonlinear estimation techniques for navigation,* NATO STO Lecture Series SET-197, Navigation Sensors and Systems in GNSS Degraded and Denied Environments (2013).

[61] K. Röbenack and K. J. Reinschke, *An efficient method to compute lie derivatives and the observability matrix for nonlinear systems,* in *Proc. 2000 International Symposium on Nonlinear Theory and its Applications (NOLTA'2000), Dresden, Sept. 17*, Vol. 21 (2000) pp. 625–628.

[62] E. D. Sontag, *Mathematical control theory: deterministic finite dimensional systems*, Vol. 6 (Springer Science & Business Media, 2013).

[63] M. Šimandl, J. Královec, and P. Tichavský, *Filtering, predictive, and smoothing Cramér–Rao bounds for discrete-time nonlinear dynamic systems,* Automatica **37**, 1703 (2001).

[64] G. Einicke, *Nonlinear prediction, filtering and smoothing,* in *Smoothing, Filtering and Prediction*, edited by G. A. Einicke (IntechOpen, Rijeka, 2012) Chap. 10.

[65] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing* (Cambridge university press, 2007).

[66] D. A. Vallado, *Fundamentals of astrodynamics and applications*, Vol. 12 (Springer Science & Business Media, 2001).

[67] A. G. Cornejo, *The rotating reference frame and the precession of the equinoxes,* Lat. Am. J. Phys. Educ. Vol **7**, 591 (2013).

[68] P. K. Seidelmann, B. A. Archinal, M. F. A'hearn, A. Conrad, G. J. Consolmagno, D. Hestroffer, J. L. Hilton, G. A. Krasinsky, G. Neumann, and J. Oberst, *Report of the IAU/IAG Working Group on cartographic coordinates and rotational elements: 2006,* Celestial Mechanics and Dynamical Astronomy **98**, 155 (2007).

[69] W. George and I. Collins, *The foundations of celestial mechanics,* The Pachart Foundation dba Pachart Publishing House and reprinted by permission, US (2004).

[70] K. F. Wakker, *Fundamentals of astrodynamics,* (2015).

[71] J. Gurland and R. C. Tripathi, *A simple approximation for unbiased estimation of the standard deviation,* The American Statistician **25**, 30 (1971).

[72] P. J. Buist, S. Engelen, A. Noroozi, P. Sundaramoorthy, S. Verhagen, and C. Verhoeven, *Overview of pulsar navigation: Past, present and future trends,* Navigation **58**, 153 (2011).

[73] J. Hanson, S. Sheikh, P. Graven, and J. Collins, *Noise analysis for x-ray navigation systems,* in *Position, Location and Navigation Symposium, 2008 IEEE/ION* (IEEE) pp. 704–713.

[74] E. Kaplan and C. Hegarty, *Understanding GPS: principles and applications* (Artech house, 2005).

[75] I. Jovanovic and J. Enright, *An approximate model for pulsar navigation simulation,* Acta Astronautica **119**, 101 (2016).

[76] C. Kabakchiev, V. Behar, P. Buist, I. Garvanov, D. Kabakchieva, M. Bentum, and J. Fernandes, *Improvement in snr of signal detection using filtering in pulsar-based navigation systems,* in *Radar Symposium (IRS), 2017 18th International* (IEEE) pp. 1–10.

[77] D. B. Goncalo Tavares and Jorge Fernandes, *A Study on the Accuracy of Radio Pulsar Navigation Systems,* (2015).

[78] T. Mineo, G. Cusumano, L. Kuiper, W. Hermsen, E. Massaro, W. Becker, L. Nicastro, B. Sacco, F. Verbunt, and A. G. L. B. J. Lyne, *The pulse shape and spectrum of the millisecond pulsar PSR J0218+ 4232 in the energy band 1-10 keV observed with BeppoSAX,* Astronomy and Astrophysics **355**, 1053 (2000).

[79] P. Arumugasamy, G. G. Pavlov, and G. P. Garmire, *X-ray Emission From J1446–4701, J1311–3430, and Other Black Widow Pulsars,* The Astrophysical Journal **814**, 90 (2015).

[80] X. Zhang, P. Shuai, L. Huang, S. Chen, and L. Xu, *Mission overview and initial observation results of the x-ray pulsar navigation-i satellite,* International Journal of Aerospace Engineering **2017** (2017).

A

TEST-BED ARCHITECTURE FLOW CHARTS

A.1. ORBITAL MODULE

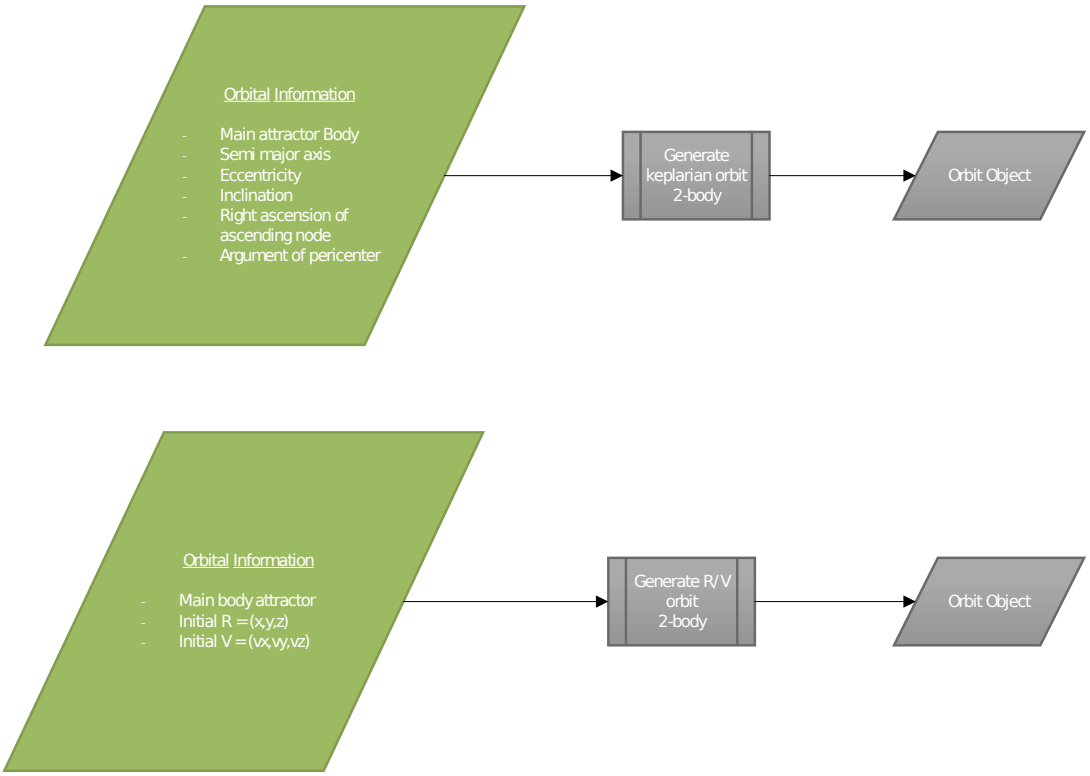


Figure A.1: Generation of the ephemeris

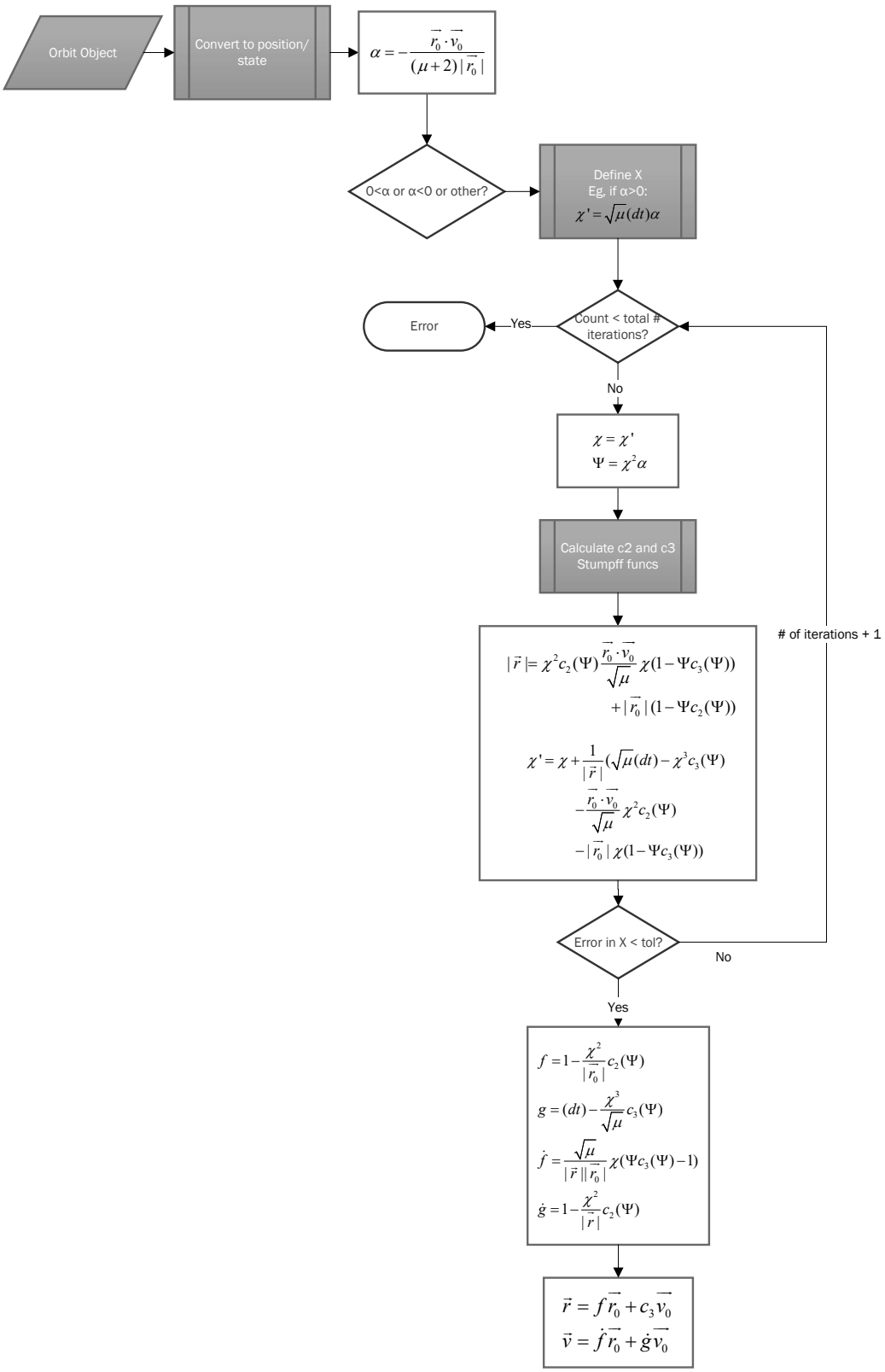


Figure A.2: Orbital propagation using Stumpff functions flow chart

B

Software Unit Tests

B.1. Orbit Module

Make Ephemeris

```
import OrbitModule
from OrbitModule import Orbit
from astropy.time import Time
from astropy import units as u
from OrbitModule import Earth, Mars, Sun
ref_body_mapper = {'sun': Sun, 'earth': Earth, 'mars': Mars}

class Test(object):

    def __init__(self, ref_time, true_state, ref_body, orbit_info, orbit_type, dist_units, vel_units, sim_length):
        self.refTime = ref_time
        self.true_state = true_state
        self.refBody = ref_body_mapper[ref_body]
        self.orbitInfo = orbit_info
        self.orbitType = orbit_type
        self.simStates = []
        self.i = 0
        self.dist_units = dist_units
        self.vel_units = vel_units
        self.simLength = sim_length

    def makeEphem(self):
        """
        Generates the ephemeris which will be propagated.
        Ephemeris type is dependent on the type of orbit. Currently only 2-body orbits are implemented, so all orbit
        types are converted to 2-body ephemerides
        """

        orbit_epoch = Time(self.refTime[0], scale=self.refTime[1], format=self.refTime[2])

        if self.true_state is None:
            if self.orbitType is 'kepler':
                self.ephem = Orbit.from_classical(self.refBody, self.orbitInfo[0], self.orbitInfo[1],self.orbitInfo[2],
                                                  self.orbitInfo[3], self.orbitInfo[4], self.orbitInfo[5], epoch=orbit_epoch)
                #plot(self.ephem)
                self.simStates.append([self.ephem.r.value, self.ephem.v.value])

            elif self.orbitType is 'position_velocity':
                self.ephem = Orbit.from_vectors(self.refBody, self.orbitInfo[0], self.orbitInfo[1], epoch=orbit_epoch)
                self.interim = self.ephem.state.to_classical()
                # self.ephem = Orbit.from_classical(self.refFrame, self.interim)
                self.ephem = Orbit.from_classical(self.refBody, self.interim.a, self.interim.ecc,
                                                  self.interim.inc, self.interim.raan, self.interim.argp,
                                                  self.interim.nu, epoch=orbit_epoch)
                #plot(self.ephem)

            else:
                print("error")
        else:

            self.state =[self.true_state[self.i,0]*self.dist_units, self.true_state[self.i,1]*self.vel_units]
            self.i+=1
```

```
        self.startEpoch = orbit_epoch
        self.endEpoch = orbit_epoch + self.simLength
        self.time = self.startEpoch
        #print(self.ephem.epoch.iso)
        #print(self.endEpoch.iso)

        # self.ephem_kernel = EphemerisModule._get_kernel(self.ref_ephem)

dt = 100. * u.s  # choice of s, min, hour, day, year
Simulation_length = 200000. * u.s
refTime = ["2018-01-01 00:00", 'tdb', 'iso']  # start epoch, timing reference, format]

a = 7136.6 * u.km  # semi-major axis [km]
ecc = 0.3 * u.one# eccentricity [-]
inc = 90. * u.deg# inclination [deg]
raan = 175. * u.deg   # Right ascension of the ascending node [deg]
argp = 90. * u.deg # Argument of perigee [deg]
nu = 178. * u.deg # True anaomaly [deg]
kep = [a, ecc, inc, raan, argp, nu]

R_V_state = [[-6045, -3490, 2500]* u.km, [-3.457, 6.618,
         2.533]* u.km / u.s]  # position then velocity

Orbit_Type = 'ephemeris' # 'kepler' 'position_velocity' 'ephemeris'
Reference_Body = 'earth'  # earth, mars, sun

case1 = [refTime, None, 'earth', kep, 'kepler', u.km, (u.m/u.s), Simulation_length]
case2 = [refTime, None, 'mars', R_V_state, 'position_velocity', u.km, (u.km/u.s), Simulation_length]

C = [case1, case2]

for case in C:
    t = Test(case[0],case[1],case[2],case[3],case[4],case[5], case[6], case[7])
    t.makeEphem()
    print(t.ephem)
```

Update State

```
import OrbitModule
import numpy as np
from astropy.time import Time
from astropy import units as u

class Test(object):

    def __init__(self, dt, sim_length):

        self.length = sim_length
        self.dt = dt

        if self.dt > self.length:
            raise('Timestep greater than the simulation length')

        # orbit_epoch = Time(self.refTime[0], scale=self.refTime[1], format=self.refTime[2])
        a = 7136.6 * u.km  # semi-major axis [km]
        ecc = 0.3 * u.one  # eccentricity [-]
        inc = 90. * u.deg  # inclination [deg]
        raan = 175. * u.deg  # Right ascension of the ascending node [deg]
        argp = 90. * u.deg  # Argument of perigee [deg]
        nu = 178. * u.deg  # True anaomaly [deg]
        kep = [a, ecc, inc, raan, argp, nu]
        self.ephem = OrbitModule.Orbit.from_classical(OrbitModule.Earth, a, ecc, inc,
                                        raan, argp, nu, epoch=Time("2018-01-01 00:00", scale='tdb', format='iso
    '))

    def updateState(self):
        """
        propagates the ephemeris by the timestep and updates the s/c state
        :return: True state of the spacecraft in cartestian coords
        """
        # print("start update")
        self.ephem = self.ephem.propagate(self.dt, method=cowell, rtol=1e-15)
        r = self.ephem.r.value
        v = self.ephem.v.value
        self.state = np.array([r[0], r[1], r[2], v[0], v[1], v[2]])
```

```
37
38          return (self.state)
39
40
41
42  case1 = [1*u.s, 10*u.s]
43  case2 = [1*u.min, 10*u.min]
44  case3 = [1*u.hr, 10*u.min]
45
46
47  C = [case1, case2, case3]
48
49  for case in C:
50      t = Test(case[0], case[1])
51      i = 0
52      time = 0 * (case[1].unit)
53      while time.value < case[1].value:
54          t.updateState()
55          i += 1
56          time = i * (case[0])
57          print(t.state)
```

Transform to SSB

```
1   import OrbitModule
2   import numpy as np
3   from astropy.time import Time
4   from astropy import units as u
5   import EphemerisModule
6   from astropy.coordinates import solar_system_ephemeris
7   class Test(object):
8
9       def __init__(self, dt, sim_length, planet):
10
11          solar_system_ephemeris.set("de430")
12          self.ref_ephem = "de430"
13          self.ephem_kernel = EphemerisModule._get_kernel(self.ref_ephem)
14
15          self.length = sim_length
16          self.dt = dt
17          self.body = planet
18
19          if self.dt > self.length:
20              raise('Timestep greater than the simulation length')
21
22          # orbit_epoch = Time(self.refTime[0], scale=self.refTime[1], format=self.refTime[2])
23          a = 7136.6 * u.km  # semi-major axis [km]
24          ecc = 0.3 * u.one  # eccentricity [-]
25          inc = 90. * u.deg  # inclination [deg]
26          raan = 175. * u.deg  # Right ascension of the ascending node [deg]
27          argp = 90. * u.deg  # Argument of perigee [deg]
28          nu = 178. * u.deg  # True anaomaly [deg]
29          kep = [a, ecc, inc, raan, argp, nu]
30          self.ephem = OrbitModule.Orbit.from_classical(self.body, a, ecc, inc,
31                                                        raan, argp, nu, epoch=Time("2018-01-01 00:00", scale='tdb', format='iso
    '))
32          self.time = Time("2018-01-01 00:00", scale='tdb', format='iso')
33
34      def TransformSSB(self):
35          """
36          propagates the ephemeris by the timestep and updates the s/c state
37          :return: True state of the spacecraft in cartestian coords
38          """
39          # print("start update")
40          self.ephem = self.ephem.propagate(self.dt, method=OrbitModule.cowell, rtol=1e-15)
41          r = self.ephem.r.value
42          v = self.ephem.v.value
43          self.state = np.array([r[0], r[1], r[2], v[0], v[1], v[2]])
44          self.SSB_state = EphemerisModule.body_centered_to_icrs(self.ephem.r, self.ephem.v, self.body, self.time,
45                                                                 ephemeris=self.ref_ephem, Ephemkernel=self.ephem_kernel)
46          self.SSB_X = [self.SSB_state[0][0].value, self.SSB_state[0][1].value, self.SSB_state[0][2].value, self.
    SSB_state[1][0].value,
47                        self.SSB_state[1][1].value, self.SSB_state[1][2].value]
48          self.time += self.dt
49
```

```
        return (self.SSB_state)



case1 = [1*u.s, 10*u.s, OrbitModule.Sun]
case2 = [1*u.min, 10*u.min, OrbitModule.Earth]
case3 = [1*u.hr, 10*u.hr, OrbitModule.Mars]


C = [case1, case2, case3]

for case in C:
    print()
    t = Test(case[0], case[1], case[2])
    i = 0
    time = 0 * (case[1].unit)
    while time.value < case[1].value:
        t.TransformSSB()
        i += 1
        time = i * (case[0])
        print(t.state)
        print(t.SSB_X)
```

B.2. SENSOR MODULE

Format Parameters

```
import SensorModule
import numpy as np
from astropy import units as u
import xml.etree.cElementTree as ElementTree

class Xml2Dict(dict):
    """
    Reads an XML file and creates a dictionary from this
    - if units are present in the attributes of the observables - add the units flag
    """
    def __init__(self, parent, units=False):
        if not units:
            for child in parent.getchildren():
                dictA = {}
                for grandchild in child.getchildren():
                    dictA.update({grandchild.get('name'): eval(grandchild.get('type'))(grandchild.get('value'))})
                self.update({child.get('name'): dictA})
        else:
            for child in parent.getchildren():
                dictA = {}
                for grandchild in child.getchildren():
                    dictA.update({grandchild.get('name'):[eval(grandchild.get('type'))(grandchild.get('value')),
                                                          str(grandchild.get('units'))]})
                self.update({child.get('name'): dictA})


def format_params(sensor):
    xml_tree_root = sensor.tree.getroot()
    lib_params = Xml2Dict(xml_tree_root)  # TO DO: CONVERSION OF POSITION VECTORS TO OTHER REF FRAMES
    for key in lib_params:  # create a numpy array from the xml file
        pos = np.array([lib_params[key]['direction.x'], lib_params[key]['direction.y'],
                        lib_params[key]['direction.z']])
        del (lib_params[key]['direction.x'])
        del (lib_params[key]['direction.y'])
        del (lib_params[key]['direction.z'])
        lib_params[key]['direction'] = pos
        return(lib_params)



input = [1*u.s, [1e-9], 'spectrometer_test.xml' ]
spec = SensorModule.Spectrometer(input)
print(format_params(spec))

input = [1*u.s, [1e-6, 1e-6], 'ang_sensor_lib.xml' ]
ang = SensorModule.AngSensor(input)
print(format_params(ang))
```

Make Observations

```
import SensorModule
import numpy as np
from astropy import units as u
import xml.etree.cElementTree as ElementTree


input = [1*u.s, [1e-9], 'spectrometer_test.xml' ]
spec = SensorModule.Spectrometer(input)
X = [100, 100, 100, 10, 10, 10]
spec.h(X)
observed_wavelength = {}
for key in spec.lib_params:
    dot_prod = np.dot(X[3:6], spec.observables_SCframe[key]['direction'])
    observed_wavelength[key] = 1 / (1 + dot_prod / (3e8 / 1000)) * spec.lib_params[key][
        'base_wavelength_nm']  # Doppler equation scaled to nanometer wavelength
    print(observed_wavelength[key] - spec.observed_wavelength[key] )



input = [1*u.s, [1e-6, 1e-6], 'ang_sensor_lib.xml' ]
ang = SensorModule.AngSensor(input)
X = [100, 100, 100, 10, 10, 10]
ang.h(X)

for key in ang.lib_params:
    # run through the elements in the library and take a measurement from the s/c's persepctive
    dX = [ang.lib_params[key]['direction'][0] - X[0], ang.lib_params[key]['direction'][1] - X[1],
          ang.lib_params[key]['direction'][2] - X[2]]
    r_true = (np.sqrt((dX[0]) ** 2 + (dX[1]) ** 2 + (dX[2]) ** 2))
    theta_true = (np.arccos(dX[2] / np.sqrt((dX[0]) ** 2 + (dX[1]) ** 2 + (dX[2]) ** 2)))
    phi_true = (np.arctan2(dX[1], dX[0]))

    mix = np.array([float(theta_true),
                    float(phi_true)])
    print(mix[0] - ang.output[key][0])
    print(mix[1] - ang.output[key][1])
```

AWGN

```

import matplotlib.pyplot as plt
import numpy as np

def AWGN(obs, std):
    obs2 = []
    for i in obs:
        obs2.append(i + (np.random.normal(0, std)))
    return(obs2)

N = 10000
T = 5

t = np.linspace(0,T,N)
w1 = 0.1
w2 = 0.5
x = np.sin(2*np.pi*w1*t) + np.sin(2*np.pi*w2*t)


np.random.seed(20)
plt.figure(1)
x_N = AWGN(x,0.1)
plt.plot(t, x_N, 'r')
plt.plot(t,x)
plt.show()
```

B.3. NAVIGATION MODULE

Format Observations

```
import SensorModule
import numpy as np
from astropy import units as u
```

```

def obs_func(sensors):
    """
    Collects all of the observation equations from the sensors, checks if the observation is ready and then
     stores
    the corresponding results. The noise matrices needed for the UKF are dynamically updated every call.

    :param sensors:
    """
    # if the observations are available, the relevant observation equations are added for the UKF
    h_func = []
    num_obs = []
    all_measurements = []
    residual_h = []
    z_mean = []
    R_params = []
    for sens in sensors:
        h_func.append(sens.h)  # holds the observation equations
        all_measurements.append(sens.measurements)  # holds the measurements
        residual_h.append(sens.residual_h)
        z_mean.append(sens.z_mean)
        num_obs.append(len(sens.measurements.values()) * sens.vec_length)
        R_params.extend(np.array(sens.R_params).flatten())  # creates an array containing the values

        print(all_measurements)
        print(R_params)



input = [1*u.s, [1e-9], 'spectrometer_test.xml' ]
spec = SensorModule.Spectrometer(input)

input = [1*u.s, [1e-6, 1e-6], 'ang_sensor_lib.xml' ]
ang = SensorModule.AngSensor(input)

X = [100, 100, 100, 1, 1, 1]
spec.observe(X)
ang.observe((X))
obs_func([spec,ang])
```

Generate Sigma Points

This is just the code rather than a unit test as the functionality is difficult to test

```
import numpy as np
import Filter

def generate_sigma_points(x, P, alpha, n, kappa):

    if np.isscalar(x):
        x = np.asarray([x])

    if np.isscalar(P):
        P = np.eye(n) * P
    else:
        P = np.asarray(P)

    lambda = alpha**2 * (n+kappa)-n
    # print(P)
    # print()
    U = Filter.cholesky((lambda + n) * P)
    # print(U)
    # print()
    sigmas = np.zeros((2*n + 1, n))
    sigmas[0] = x
    for k in range(n):
        sigmas[k + 1] = np.subtract(x, -U[k])
        sigmas[n + k + 1] = np.subtract(x, U[k])

    return sigmas
```

Unscented Transform

```
def unscented_transform(num_measurements, sigmas, Wm, Wc, noise_cov=None,
```

```
                        mean_fn=None, residual_fn=None):

    kmax, n = sigmas.shape
    x = []

    if mean_fn is None:
        # new mean is just the sum of the sigmas * weight
        x = np.dot(Wm, sigmas)     # dot = \Sigma^n_1 (W[k]*Xi[k])
    else:
        # for func in mean_fn:
        #     x.append(func(sigmas, Wm))
        i = 0
        j = 0
        for func in mean_fn:
            x.extend(func(sigmas[:,i:i+num_measurements[j]], Wm))
            i += num_measurements[j]
            j += 1
    x = np.array(x)


    # new covariance is the sum of the outer product of the residuals
    # times the weights

    if residual_fn is None:
        y = sigmas - x[np.newaxis,:]
        P = y.T.dot(np.diag(Wc)).dot(y)
    else:
        P = np.zeros((n, n))
        for k in range(kmax):
            y = []
            i = 0
            j = 0
            if type(residual_fn) != np.ufunc:
                for func in residual_fn:
                    y.extend(func(sigmas[k][i:i+num_measurements[j]], x[i:i+num_measurements[j]]))
                    i += num_measurements[j]
                    j += 1
            else:
                y.extend(residual_fn(sigmas[k][i:i + num_measurements], x[i:i + num_measurements]))
                i += num_measurements
            y = np.array(y)
            P += Wc[k] * np.outer(y, y)

    if noise_cov is not None:
        P = P + noise_cov

    return (x, P)
```

B.4. ANALYSIS MODULE

Fourier Transform

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft
from scipy.signal import convolve


N = 100000
T = 50

t = np.linspace(0,T,N)
w1 = 0.1
w2 = 0.5
x = np.sin(2*np.pi*w1*t) + np.sin(2*np.pi*w2*t)
plt.figure(1)
plt.plot(t,x)


freq_data = np.array(fft(x))[:N//2]
plt.figure(2)
plt.plot(np.linspace(0.0, 1.0 / (2.0 * (T/N)), N // 2), abs(freq_data))
plt.xlim(0,0.6)

```

```
23 plt.show()
```

C

Verification Code and Results

C.1. Orbit

C.1.1. Code

```
import numpy as np
import matplotlib.pyplot as plt
import EphemerisModule
from astropy import units as u
from poliastro.bodies import Earth, Mars, Sun
from poliastro.twobody import Orbit
from poliastro.twobody.propagation import cowell, kepler, RK4, mean_motion
from astropy.time import Time
from astropy.coordinates import solar_system_ephemeris
from poliastro import constants
import pickle as pkl


def norm(vec):
    return np.sqrt(np.dot(vec,vec))

ref_body_mapper = {'sun': Sun, 'earth': Earth, 'mars': Mars}
ref_time_mapper = {'s': u.s, 'min': u.min, 'hr': u.hr, 'day': u.day, 'yr': u.yr}


class MakeOrbit(object):
    """ MakeOrbit class uses AstroPy and Poliastro to define and propagate an orbit in keplarian or r/v coords with a choice of
    central body. Output in r/v
    Parameters

    in_orbit_type : the type of orbit definition 'kepler' 'position_velocity' 'ephemeris'
    in_orbit_info : the parameters which define the orbit (kep elements, r/v state or ephemeris [TO DO])
    in_ref_body : the reference orbital body
    in_ref_time : the reference start time (in J2000 ref frame)
    in_timestep : the desired time step of propagation (in units of s, min, hr, yr)
    in_sim_length :the simulation length in specified units

    """
    def __init__(self, in_orbit_type, in_orbit_info, in_ref_body, in_ref_time, in_timestep, in_sim_length, noise, intergrator):

        solar_system_ephemeris.set("de430")
        self.ref_ephem = "de430"


        self.orbitType = in_orbit_type
        self.orbitInfo = in_orbit_info
        self.refBody = ref_body_mapper[in_ref_body]
        self.refTime = in_ref_time
        self.dt = in_timestep
        self.simLength = in_sim_length
        # if(self.dt/self.simLength).decompose() == u.one:
        self.simStates = []
        self.endEpoch = 0
        self.startEpoch = 0
        self.noise = np.diag(noise) if noise != 0 else None

        self.i = 0
        self.barycentric_state = []
        self.inter = intergrator
        self.M = []
```

```
56          self.e = []
57
58      def makeEphem(self):
59          """
60          Generates the ephemeris which will be propagated.
61          Ephemeris type is dependent on the type of orbit. Currently only 2-body orbits are implemented, so all orbit
62          types are converted to 2-body ephemerides
63          """
64
65          orbit_epoch = Time(self.refTime[0], scale=self.refTime[1], format=self.refTime[2])
66
67          if self.orbitType is 'kepler':
68              self.ephem = Orbit.from_classical(self.refBody, self.orbitInfo[0], self.orbitInfo[1],self.orbitInfo[2],
69                                                self.orbitInfo[3], self.orbitInfo[4], self.orbitInfo[5], epoch=orbit_epoch)
70
71          elif self.orbitType is 'position_velocity':
72              self.ephem = Orbit.from_vectors(self.refBody, self.orbitInfo[0], self.orbitInfo[1], self.orbitInfo[2],
73                                              self.orbitInfo[3], self.orbitInfo[4], self.orbitInfo[5], epoch=orbit_epoch)
74              self.interim = self.ephem.state.to_classical()
75              # self.ephem = Orbit.from_classical(self.refFrame, self.interim)
76              self.ephem = Orbit.from_classical(self.refBody, self.interim.a, self.interim.ecc,
77                                                self.interim.inc, self.interim.raan, self.interim.argp,
78                                                self.interim.nu, epoch=orbit_epoch)
79
80          else:
81              print("error")
82          self.startEpoch = self.ephem.epoch
83          self.endEpoch = self.ephem.epoch + self.simLength
84          #print(self.ephem.epoch.iso)
85          #print(self.endEpoch.iso)
86
87          self.ephem_kernel = EphemerisModule._get_kernel(self.ref_ephem)
88
89      def add_noise(self, r,v):
90          """
91          Adds zero-mean gaussian noise to the ephemeris state with the user-defined noise parameters
92
93          :param r: ephemeris range [x,y,z] with units
94          :param v: ephemeris velocity
95          :return:
96          """
97          r += [np.random.normal(0, self.noise[0]),np.random.normal(0, self.noise[1]),np.random.normal(0, self.noise[2])]*self.ephem.state.r.unit
98          v += [np.random.normal(0, self.noise[3]),np.random.normal(0, self.noise[4]),np.random.normal(0, self.noise[5])] * self.ephem.state.v.unit
99
100     def updateState(self, noise=False):
101         """
102         propagates the ephemeris by the timestep and updates the s/c state
103         :return: True state of the spacecraft in cartestian coords
104         """
105         self.ephem = self.ephem.propagate(self.dt, method=self.inter, rtol=1e-15)
106         self.add_noise(self.ephem.r, self.ephem.v) if noise is True else None
107         r = self.ephem.r.to(u.m).value
108         v = self.ephem.v.to(u.m/u.s).value
109         self.state = np.array([r[0], r[1], r[2], v[0], v[1], v[2]])
110         self.simStates.append(self.state)
111         return(self.state)
112
113
114     def cart_to_mean_motion(self, t):
115         r = self.state[0:3]
116         v = self.state[3:]
117         self.e.append(norm(((norm(v)**2 - self.ephem.attractor.k.value/norm(r))*r - np.dot(r,v)*v)/self.ephem.attractor.k.value))
118         a = (-self.ephem.attractor.k.value)/(2*((norm(v)**2)/2 - self.ephem.attractor.k.value/norm(r)))
119         # E = np.arccos((norm(r)/a - 1)/e)
120         self.M.append(np.sqrt(self.ephem.attractor.k.value/abs(a)**3)*t)
121
122
```

```
Orbit_Type = 'kepler'  # 'kepler' 'position_velocity' 'ephemeris'
Reference_Body = 'earth'  # earth, mars, sun
Reference_Time = ["2018-01-01 00:00", 'tdb', 'iso']  # start epoch, timing reference, format]

# If the type kepler is chosen, change the values below:
a = 8000e3 * u.m  # semi-major axis [km]
ecc = 0.75 * u.one# eccentricity [-]
inc = 65. * u.deg# inclination [deg]
raan = 0. * u.deg   # Right ascension of the ascending node [deg]
argp = 0. * u.deg # Argument of perigee [deg]
nu = 0. * u.deg # True anaomaly [deg]
kep = [a, ecc, inc, raan, argp, nu]

dt = 10. * u.s  # choice of s, min, hour, day, year
Simulation_length = 200000. * u.s



if Reference_Body is 'sun':
    mu = constants.GM_sun
elif Reference_Body is 'earth':
    mu = constants.GM_earth
elif Reference_Body is 'mars':
    mu = constants.GM_mars
else:
    raise NameError(Reference_Body)

n = np.sqrt(mu.value/a.value**3)
M_true = []
time = []
times = [1*u.s, 10*u.s, 100*u.s, 1000*u.s, 10000*u.s]
times_graph = [1,10,100, 1000,10000]

mean_cowell_err_M = []
mean_kep_err_M = []
mean_RK4_err_M = []
mean_mean_motion_err_M = []

std_cowell_err_M = []
std_kep_err_M = []
std_RK4_err_M = []
std_mean_motion_err_M = []

mean_cowell_err_e = []
mean_kep_err_e = []
mean_RK4_err_e = []
mean_mean_motion_err_e = []

std_cowell_err_e = []
std_kep_err_e = []
std_RK4_err_e = []
std_mean_motion_err_e = []

for dt in times:

    M_true = []
    time = []

    ephem_cowell = MakeOrbit(Orbit_Type, kep, Reference_Body, Reference_Time, dt
                                            , Simulation_length, 0, cowell)
    ephem_kep = MakeOrbit(Orbit_Type, kep, Reference_Body, Reference_Time, dt
                                            , Simulation_length, 0, kepler)
    ephem_RK4 = MakeOrbit(Orbit_Type, kep, Reference_Body, Reference_Time, dt
                                            , Simulation_length, 0, RK4)
    ephem_mean_motion = MakeOrbit(Orbit_Type, kep, Reference_Body, Reference_Time, dt
                                            , Simulation_length, 0, mean_motion)
    ephem_cowell.makeEphem()
    ephem_kep.makeEphem()
    ephem_RK4.makeEphem()
    ephem_mean_motion.makeEphem()

    for i in range(1,int(Simulation_length.value/dt.value)):
        print("%d / %d"%(i, int(Simulation_length.value/dt.value)))
        time.append(i*dt.value)
        M_true.append(n*i*dt.value)
```

```
198             ephem_cowell.updateState()
199             ephem_cowell.cart_to_mean_motion(i*dt.value)
200             ephem_kep.updateState()
201             ephem_kep.cart_to_mean_motion(i*dt.value)
202             ephem_RK4.updateState()
203             ephem_RK4.cart_to_mean_motion(i*dt.value)
204             ephem_mean_motion.updateState()
205             ephem_mean_motion.cart_to_mean_motion(i*dt.value)
206
207
208     M_true = np.array(M_true)
209     ephem_cowell.M = np.array(ephem_cowell.M)
210     ephem_kep.M = np.array(ephem_kep.M)
211     ephem_RK4.M = np.array(ephem_RK4.M)
212     ephem_mean_motion.M = np.array(ephem_mean_motion.M)
```

C.1.2. NUMERICAL RESULTS

CIRCULAR ORBIT

Table C.1: Mean motion RMS error circ orbit

Time [s]	1	10	100	1000	10,00
Cowell	3.76E-12	1.13E-12	1.55E-12	2.65E-12	2.26E-12
Kepler	1.71E-09	3.45E-11	6.68E-10	4.16E-07	3.75E-12
RK4	5.12E-05	5.12E-03	5.12E-01	4.75E+01	6.31E+04

Table C.2: Mean motion end error circ orbit

Time [s]	1	10	100	1000	10,000
Cowell	-1.67E-11	-4.32E-12	-1.99E-12	8.13E-12	6.03E-12
Kepler	2.27E-13	1.37E-13	5.66E-10	6.44E-09	2.31E-15
RK4	2.34E-04	2.29E-03	7.56E-03	3.28E-01	9.00E+00

Table C.3: Eccentricity RMS error circ orbit

Time [s]	1	10	100	1000	10,000
Cowell	2.23E-14	1.09E-14	4.05E-15	7.85E-15	5.00E-15
Kepler	1.67E-13	5.13E-14	4.24E-11	4.00E-09	2.34E-15
RK4	5.60E-04	5.60E-03	5.61E-02	5.73E-01	9.00E+00

Table C.4: Eccentricity end error circular orbit

Time [s]	1	10	100	1000	10,000
Cowell	4.29E-14	2.11E-14	8.50E-15	4.33E-15	5.42E-15
Kepler	2.27E-13	1.37E-13	5.66E-10	6.44E-09	2.31E-15
RK4	2.34E-04	2.29E-03	7.56E-03	3.28E-01	9.00E+00

INCLINED ORBIT

Table C.5: Mean motion RMS error eccentric, inclined orbit

Time [s]	1	10	100	1000	10,000
Cowell	1.19E-11	5.16E-12	7.49E-11	7.28E-11	6.16E-11
Kepler	5.34E-04	1.40E-04	1.86E-05	9.66E-08	1.62E-08
RK4	1.30E-01	1.89E+00	8.63E+01	3.04E+05	6.30E+08

Table C.6: Mean motion end error eccentric, inclined orbit

Time [s]	1	10	100	1000	10,000
Cowell	-3.37E-11	1.30E-11	2.33E-10	2.27E-10	1.82E-10
Kepler	-1.76E-06	-4.67E-07	-6.23E-08	1.81E-09	1.52E-11
RK4	4.09E-04	1.89E-03	2.33E-01	9.29E+00	1.27E+02

Table C.7: Eccentricity RMS error eccentric, inclined orbit

Time [s]	1	10	100	1000	10,000
Cowell	6.21E-15	5.90E-15	9.92E-14	8.83E-14	7.32E-14
Kepler	8.82E-07	2.31E-07	3.08E-08	4.73E-10	9.59E-12
RK4	2.84E-04	4.20E-03	2.33E-01	9.29E+00	1.27E+02

Table C.8: Eccentricity end error eccentric, inclined orbit

Time [s]	1	10	100	1000	10,000
Cowell	7.55E-15	-1.78E-14	-2.14E-13	-1.90E-13	-1.58E-13
Kepler	-1.76E-06	-4.67E-07	-6.23E-08	1.81E-09	1.52E-11
RK4	4.09E-04	1.89E-03	2.33E-01	9.29E+00	1.27E+02

C.2. RE-ENTRY

C.2.1. CODE

```
import Filter
from scipy.linalg import inv
from numpy.random import rand
from numpy.random import normal
import numpy as np
import matplotlib.pyplot as plt
import decimal
import csv
import copy

constants = [-0.59783, 13.406, 398600, 6374.]

tracker_noise = 0.17e-2**2 #rad
tracker_rng_noise = 1e-3**2

dt = 0.1
kf_update_rate = 0.1
sensor_sampling_time = 0.1


qx = 0
qv = 2.4064e-5
qb = 1e-6
np.random.seed(5000)
P = np.diag([1e-6, 1e-6, 1e-6, 1e-6, 1.])
Q = np.diag([qx, qx, qv, qv, qb])
std_true_object = np.array([np.sqrt(qv), np.sqrt(qv), 0])
est_starting_conditions = np.array([6500.4, 349.14, -1.8093, -6.7967, 0])

observer_position = np.array([[6374., 0]])
observer_var = np.array([copy.deepcopy(tracker_noise), copy.deepcopy(tracker_rng_noise)])




#parameters = np.concatenate([[tracker_noise, dt], std_true_object, est_starting_conditions,
    true_starting_conditions, P, Q])

def get_val_from_range(range, plus_minus=None):
    t = rand(1,1)
    t = -1 if t[0][0]<0.5 else 1
    x = rand(1,1)
```

```
    result = (x[0][0]*(range[1]-range[0]) + range[0])*t if plus_minus is True else x[0][0]*(range[1]-range[0]) +
     range[0]
    return result

def norm_angles(ang):
    ang_temp = np.mod(ang, 2*np.pi)
    # print("\n\n\n\n\n")
    # print(ang-ang_temp)
    # print("\n\n\n\n\n")
    if ang_temp > np.pi and abs(ang-(ang_temp-2*np.pi)) > 1e-4:
        ang = ang_temp - 2*np.pi
    return ang


class falling_object(object):
    def __init__(self, X, std_mdl, b0, H0, mu, R0):
        self.X = X
        self.std_model = std_mdl
        self.meas = [0, 0, 0]
        self.b0 = b0
        self.H0 = H0
        self.mu = mu
        self.R0 = R0

    def update(self, dt):
        #Re-entry dynamics equations
        self.R = np.sqrt(self.X[0] ** 2 + self.X[1] ** 2)
        self.V = np.sqrt(self.X[2] ** 2 + self.X[3] ** 2)
        self.b = self.b0*np.exp(self.X[4])
        self.D = self.b*np.exp((self.R0 - self.R)/self.H0)*self.V
        self.G = -self.mu/(self.R**3)


        self.X[2] += (self.D*self.X[2] + self.G*self.X[0])*dt + np.random.normal(0, self.std_model[0])
        self.X[3] += (self.D*self.X[3] + self.G*self.X[1])*dt + np.random.normal(0, self.std_model[1])
        self.X[4] += np.random.normal(0, self.std_model[2])
        self.X[0] += self.X[2] * dt
        self.X[1] += self.X[3] * dt

        self.F_Dot(dt)

        return self.X

    def F_Dot(self, dt):
        self.f_bar = []
        dxdX = np.array([1,0,dt,0,0])
        self.f_bar.append(dxdX)

        dydX = np.array([0,1,0,dt,0])
        self.f_bar.append(dydX)

        dvxdX= np.array([-((self.X[0]*self.D*self.X[2])/self.R*self.H0 + (3*self.mu*self.X[0]**2)/self.R**5 -
     self.mu/self.R**3)*dt,
                         -(self.X[1]*self.D*self.X[2]/self.R*self.H0 + 3*self.mu*self.X[0]*self.X[1]/self.R**5)*
    dt,
                         (self.D*self.X[2]**2/self.V**2 + self.D)*dt + 1,
                         (self.D*self.X[2]*self.X[3]/self.V**2)*dt,
                         (self.D*self.X[2])*dt])
        self.f_bar.append(dvxdX)

        dvydX= np.array([-(self.X[0]*self.D*self.X[3]/self.R*self.H0 + 3*self.mu*self.X[0]*self.X[1]/self.R**5)*
    dt,
                        -((self.X[1]*self.D*self.X[3])/self.R*self.H0 + (3*self.mu*self.X[1]**2)/self.R**5 - self
    .mu/self.R**3)*dt,
                         (self.D*self.X[2]*self.X[3]/self.V**2)*dt,
                         (self.D * self.X[3] ** 2 / self.V ** 2 + self.D) * dt + 1,
                         (self.D*self.X[3])*dt])
        self.f_bar.append(dvydX)

        dBdX = np.array([0,0,0,0,1])
        self.f_bar.append(dBdX)


        self.f_bar = np.array(self.f_bar)

```

```
112
113 def f_cv(X, dt):
114     """ state transition function for a 3D
115     accelerating object under gravity"""
116
117     b0, H0, mu, R0 = constants
118
119     b = b0 * np.exp(X[4])
120     R = np.sqrt(X[0] ** 2 + X[1] ** 2)
121     V = np.sqrt(X[2] ** 2 + X[3] ** 2)
122     D = b * np.exp((R0 - R) /H0) * V
123     G = -mu / R ** 3
124
125     X[2] += (D * X[2] + G * X[0]) * dt
126     X[3] += (D * X[3] + G * X[1]) * dt
127     X[0] += X[2] * dt
128     X[1] += X[3] * dt
129
130     return X
131
132
133 def h_observer(x, marks):
134     """Measurement function -
135     measuring only position"""
136     estimated_obs_angles = []
137     for i in range(0, len(marks)):
138         dX = [x[0] - marks[i][0],x[1] - marks[i][1]]
139         r = np.sqrt(dX[0]**2 + dX[1]**2)
140         # theta = np.arccos(dX[2]/r)
141         phi = np.arctan2(dX[1], dX[0])
142         estimated_obs_angles.extend([phi, r]) if ANGLES_ONLY is False else estimated_obs_angles.extend(
143             [phi])
144         # print(np.array(estimated_obs_angles))
145
146     return np.array(estimated_obs_angles)
147
148 def residual_h(a, b):
149     if any(isinstance(el, list) for el in a):
150         a = [item for sublist in a for item in sublist]
151     if any(isinstance(el, list) for el in b):
152         b = [item for sublist in b for item in sublist]
153
154     y = a - b
155
156     if ANGLES_ONLY is False:
157         # data in format [[theta1, r], [theta2, r], ...]
158         j = 0
159         for i in range(0, len(y)):
160             if j is not 1:
161                 q = y[i]
162                 y[i] = norm_angles(y[i])
163                 j += 1
164             else:
165                 j = 0
166             # if q != y[i]:
167             #     print("NORMALISED ANGLES!", b, "!=", y[i])
168     else:
169         for i in range(0, len(y)):
170             q = y[i]
171             y[i] = norm_angles(y[i])
172             # if q != y[i]:
173             #     print("NORMALISED ANGLES!", b, "!=", y[i])
174     return y
175
176 % # def residual_x(a, b):
177 % #     y = a - b
178 % #     for i in range(0, len(y)-1):
179 % #         y[i] = norm_angles(y[i])
180 % #     return y
181
182 def z_mean(sigmas, Wm):
183     z_count = sigmas.shape[1]
184     x = np.zeros(z_count)
185
186     if ANGLES_ONLY is False:
```

```
        # data in format [[theta1, phi1, r], [theta2, phi2, r], ...]

        j = 0
        for z in range(0, z_count):
            if j is not 1:
                sum_sin1 = np.sum(np.dot(np.sin(sigmas[:, z]), Wm))
                sum_cos1 = np.sum(np.dot(np.cos(sigmas[:, z]), Wm))

                x[z] = np.arctan2(sum_sin1, sum_cos1)
                j += 1
            else:
                x[z] = np.sum(np.dot(sigmas[:, z], Wm))
                j = 0

    else:
        for z in range(0, z_count):
            sum_sin1 = np.sum(np.dot(np.sin(sigmas[:, z]), Wm))
            sum_cos1 = np.sum(np.dot(np.cos(sigmas[:, z]), Wm))

            x[z] = np.arctan2(sum_sin1, sum_cos1)

    return x


class AngSensor(object):
    def __init__(self, observer_pos, stds, update):
        self.pos = observer_pos
        self.r_std = 0 if ANGLES_ONLY else stds[2]#stds[0]
        self.theta_std = stds[0]
        self.phi_std = stds[1]
        self.r = []
        self.r_true = np.zeros([len(self.pos), 1])
        self.theta_true = np.zeros([len(self.pos), 1])
        self.phi_true = np.zeros([len(self.pos), 1])
        self.storage_phi = [[0]]
        self.theta = []
        self.phi = []
        self.time = 0
        self.update_rate = update
        self.measure_ready = False

    def observe(self, x):
        for i in range(0, len(self.pos)):
            self.dX = [x[0] - self.pos[i][0], x[1]-self.pos[i][1]]
            self.r_true[i] = np.sqrt((self.dX[0]) ** 2 + (self.dX[1]) ** 2)
            self.phi_true[i] = np.arctan2(self.dX[1], self.dX[0])
            self.H_bar(x)


    def H_bar(self,x):
        self.h_bar = []
        drdx = [x[0]/self.r_true[0][0], x[1]/self.r_true[0][0],0,0,0]
        self.h_bar.append(drdx)

        dthetadx = [-self.dX[1]/(self.dX[0]**2 + self.dX[1]**2), self.dX[0]/(self.dX[0]**2 + self.dX[1]**2),
                    0,0,0]
        self.h_bar.append(dthetadx)
        self.h_bar = np.array(self.h_bar)
        # print(self.h_bar)

    def noisy_observe(self, x, time):
        self.measure_ready = False
        self.observe(x)
        self.measure = []
        self.theta = []
        self.phi = []
        #print(np.mod(time, self.update_rate))
        if self.time >= self.update_rate:
            # print("TRUE")
            self.time = 0
            for i in range(0, len(self.pos)):
                self.r = self.r_true[i] + np.random.normal(0, self.r_std)
                self.phi.append(self.phi_true[i] + np.random.normal(0, self.phi_std))
                #print(self.theta, self.phi)
                if ANGLES_ONLY:
```

```
                    mix = [float(self.phi[i])]
                else:
                    mix = [float(self.phi[i]), float(self.r)]
                self.measure.extend(mix)
            for i in range(0, len(self.pos),2):
                self.storage_phi[i].append(self.measure[i+1])
            self.measure = np.array(self.measure)
            self.measure_ready = True

x_error_storage = []
v_error_storage = []
b_error_storage = []
bounded_points = []

mean_err_store = [0,0,0,0,0]
covar_store = 0

for iterations in range(0,100):
    print(iterations)
    true_starting_conditions = np.array(
        [normal(6500.4, np.sqrt(P[0, 0])), normal(349.14, np.sqrt(P[1, 1])), normal(-1.8093, np.sqrt(P[2, 2])),
         normal(-6.7967, np.sqrt(P[3, 3])), 0.6932])


    # for i in range(0,100):
    #     print(i)
    #     true_starting_conditions = np.array([0., 0., 3000 * np.cos(np.deg2rad(45)), 3000 * np.sin(
    #         np.deg2rad(45))])  # np.array([500., -1000., 1e5, 100., 100., 500.])
    #     est_starting_conditions = np.array([10, 0, 3010 * np.cos(np.deg2rad(45)), 3010 * np.sin(np.deg2rad(45))
     ])
    h_observer.pos = observer_position

    dec = decimal.Decimal(str(dt))
    dec = abs(dec.as_tuple().exponent)
    points = Filter.MerweScaledSigmaPoints(n=5, alpha=1, beta=0., kappa=-2)
    mult = 1 if ANGLES_ONLY else 2
    ukf = Filter.UKF(dim_x=5, dim_z=mult * len(observer_position), fx=f_cv, hx=h_observer, dt=kf_update_rate,
     points=points,
                z_mean_fn=z_mean, residual_z=residual_h)
    ukf.x = np.array([est_starting_conditions])
    if ANGLES_ONLY:
        observer_std = np.array([np.sqrt(tracker_noise), np.sqrt(tracker_noise)])
        ukf.R_store = np.diag([observer_var[0]])
    else:
        observer_std = np.array([np.sqrt(tracker_noise), np.sqrt(tracker_noise), np.sqrt(tracker_rng_noise)])
        ukf.R_store = np.diag([(observer_var[0]), (observer_var[1])])
    ukf.R = ukf.R_store
    ukf.P = P


    uxs = []
    trux = []
    truv = []
    measx = []
    time = []
    std_est_x = []
    covary = []
    covar_storage = []
    std_est_v = []
    std_est_b = []
    kalman_t = []
    meas_pos = []
    update_time = []
    ukf.Q = Q
    f = falling_object(true_starting_conditions, std_true_object, constants[0], constants[1], constants[2],
     constants[3])
    station = AngSensor(observer_position, observer_std, sensor_sampling_time)
    t = 0
    i = 0
    # J = inv(ukf.P)
    J_storage = []
    J = ukf.P
    while t < 200:
        f.update(dt)
        # f.X = X_t[i, 0:4]
```

```
334         station.time += dt
335         station.observe(f.X)
336         station.noisy_observe(f.X, t)
337         ukf.predict()
338         if station.measure_ready is True:
339             # print("UPDATE")
340             ukf.update(station.measure, hx_args=(observer_position,))
341             kalman_t.append(t)
342             measx.append([station.r, station.phi])
343             x_m = station.r * np.cos(station.phi) + observer_position[0][0]
344             y_m = station.r * np.sin(station.phi) + observer_position[0][1]
345             meas_pos.append(np.array([(f.X[0]-x_m).flatten(), (f.X[1] - y_m).flatten()]).flatten())
346             update_time.append(t)
347             uxs.append(ukf.x.copy())
348             trux.append(f.X.copy())
349             time.append(t)
350             std_est_x.append(np.sqrt(np.mean([ukf.P[0][0], ukf.P[1][1]])))
351             covary.append(ukf.P[2][2])
352             std_est_v.append(np.sqrt(np.mean([ukf.P[2][2], ukf.P[3][3]])))
353             std_est_b.append(np.sqrt(ukf.P[4][4]))
354             covar_storage.append(np.diag(ukf.P))
355             % # J = inv(ukf.Q) + np.dot(station.h_bar.T, inv(ukf.R)).dot(station.h_bar) - np.dot(np.dot(inv(ukf.Q
    ), f.f_bar),
356             % #     inv(J + np.dot(f.f_bar.T, inv(ukf.Q)).dot(f.f_bar)), np.dot(inv(ukf.Q), f.f_bar))
357             % # #
358             % # J_storage.append(np.diag(J))
359
360
361         t = round(t+dt, dec)
```

C.2.2. RESULTS

Are shown in paper, section 4.3.3

D

INTEGRATION TEST PLOTS

This section shows examples of single simulation runs using the set up defined in section

D.1. NO MEASUREMENT

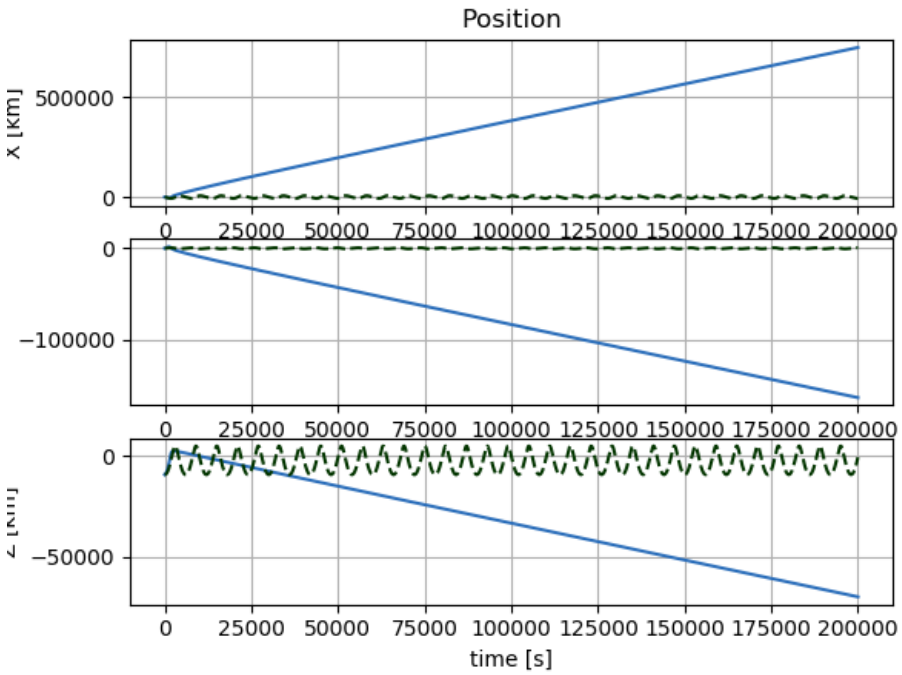


Figure D.1: Position using no measurements

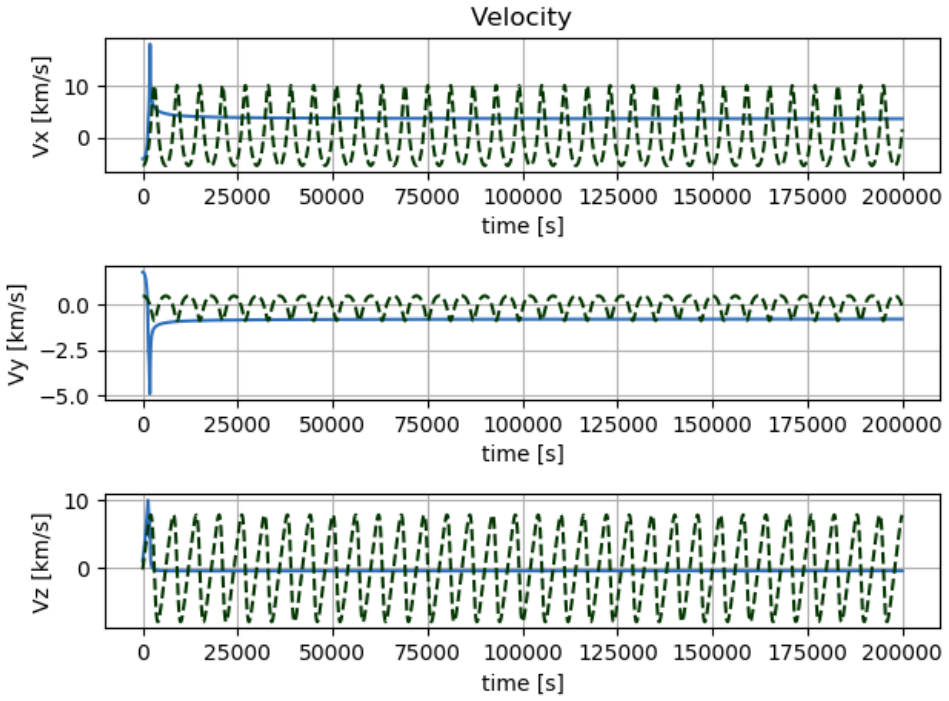


Figure D.2: Velocity using no measurements

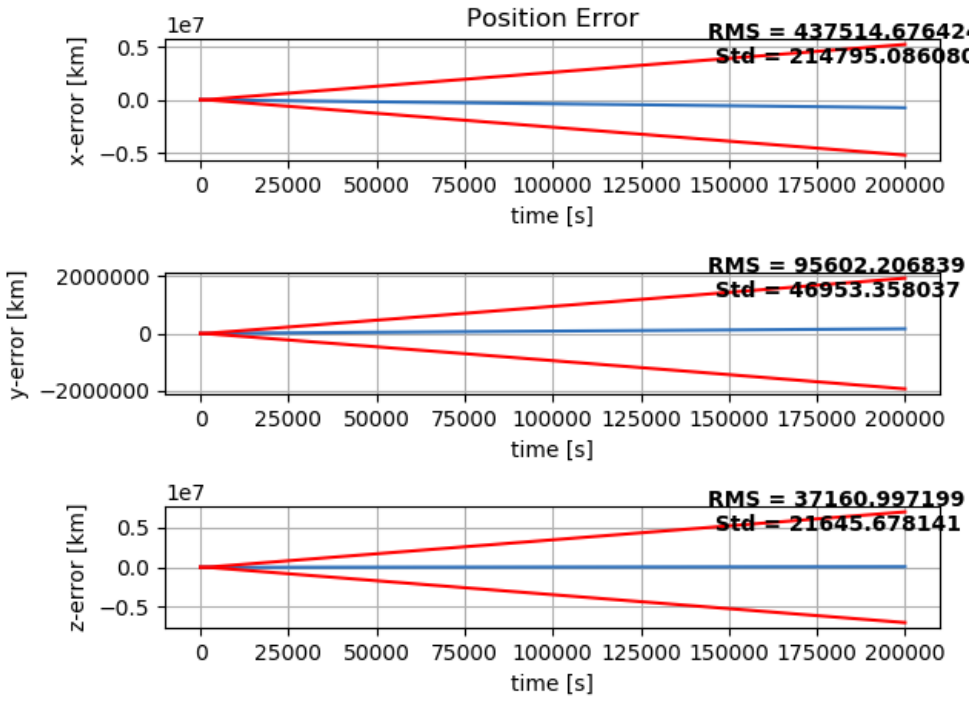


Figure D.3: Position errors using no measurements

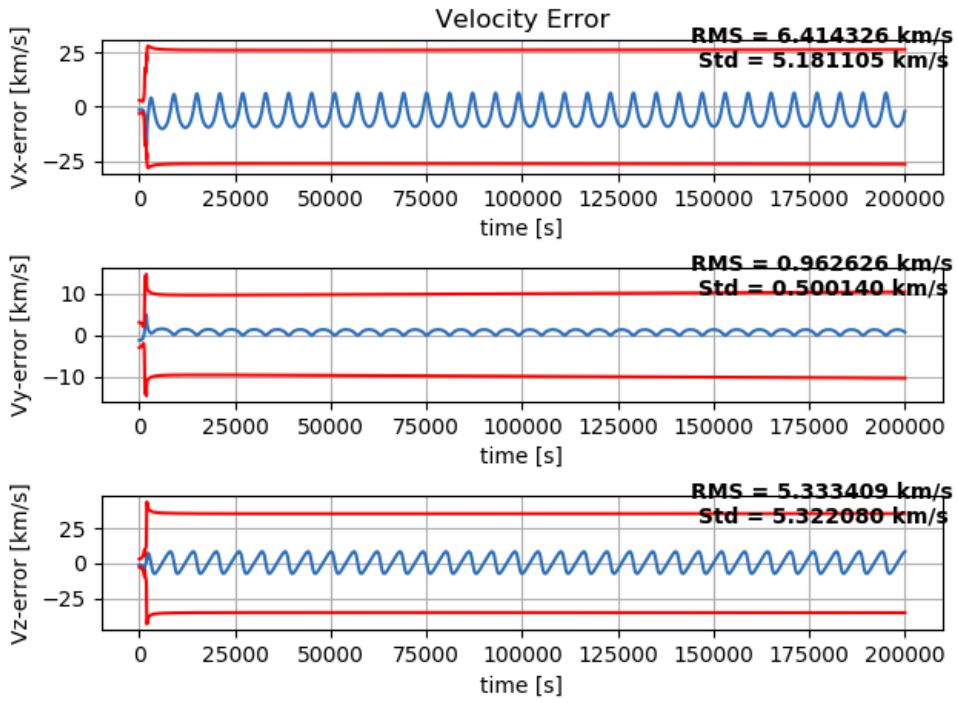


Figure D.4: Velocity errors using no measurements

D.2. RADIAL VELOCITY SENSOR ONLY

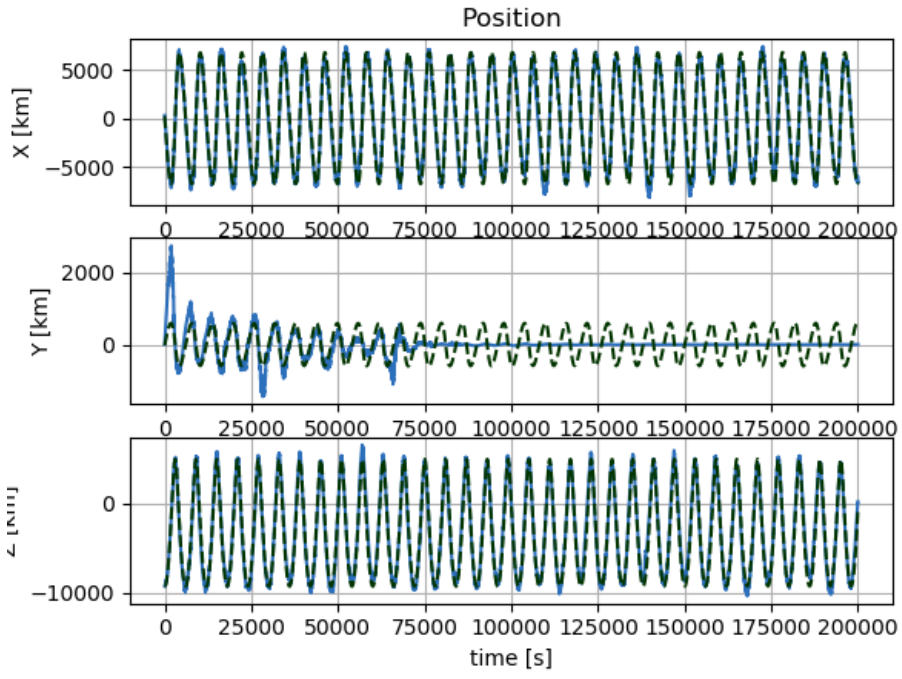


Figure D.5: Position using radial velocity sensor

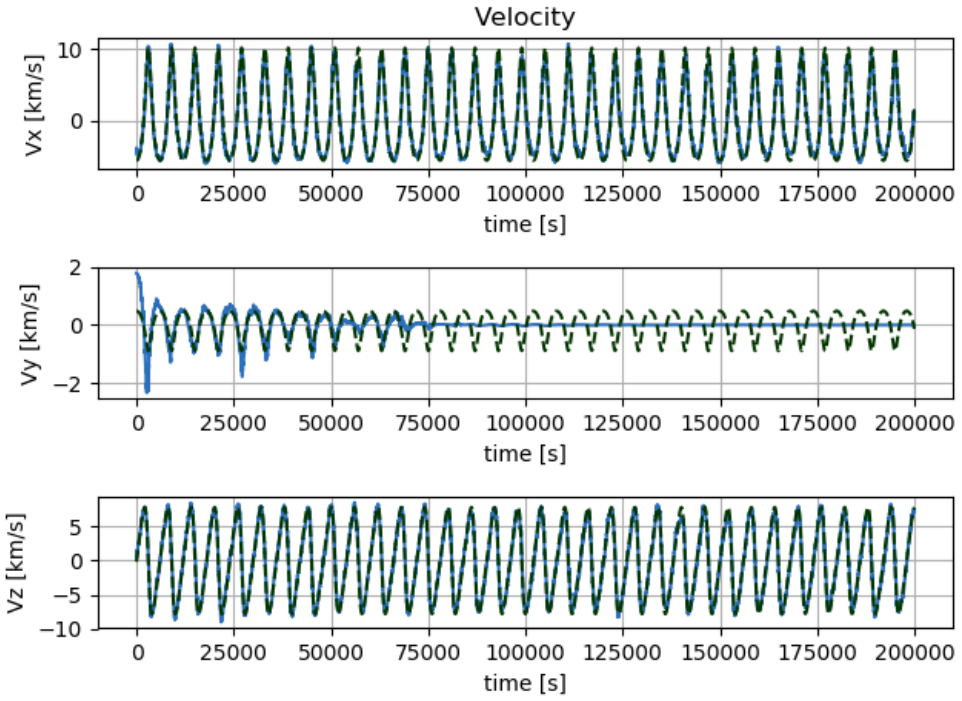


Figure D.6: Velocity using radial velocity sensor

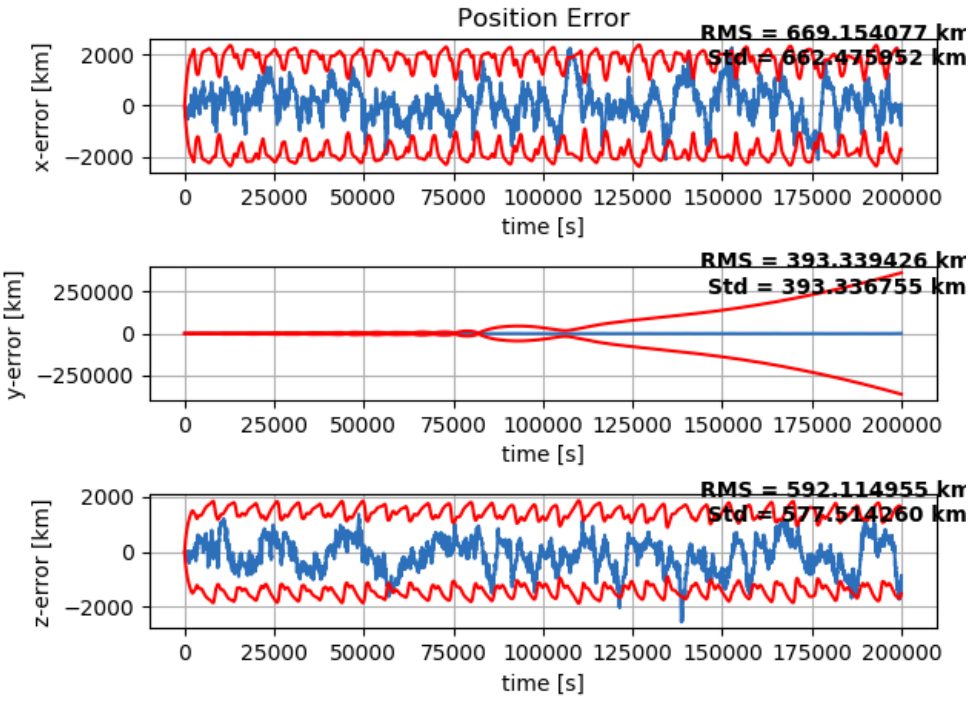


Figure D.7: Position errors using radial velocity sensor

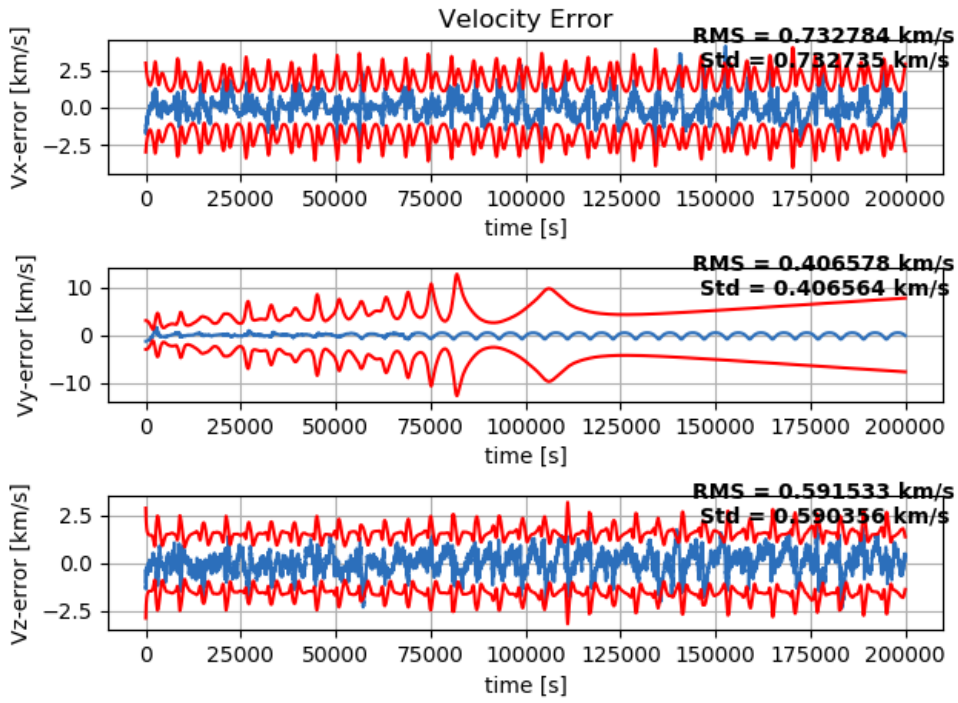


Figure D.8: Velocity errors using radial velocity sensor

D.3. ANGLE SENSOR ONLY

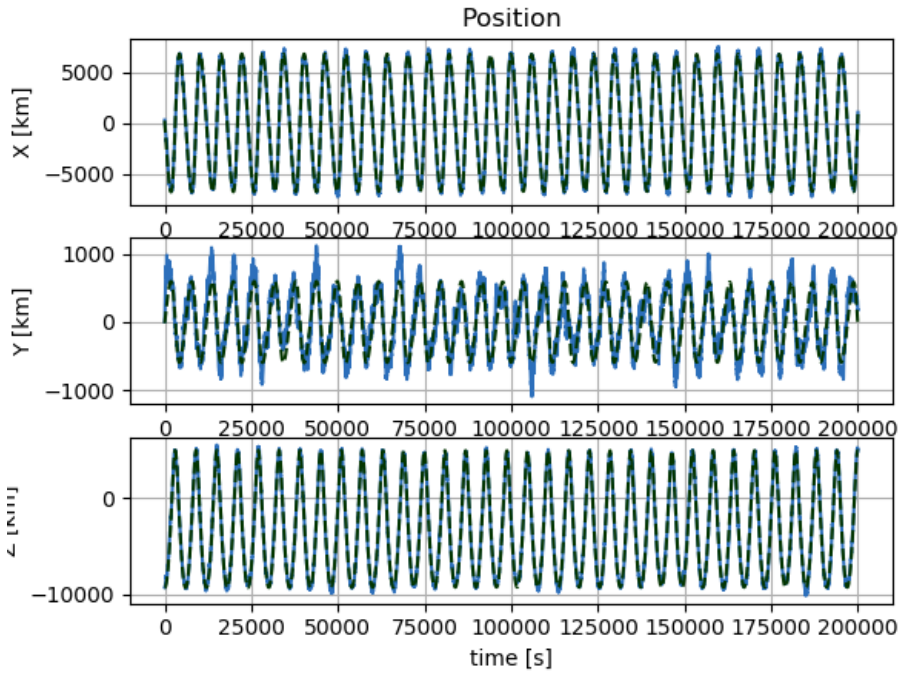


Figure D.9: Position using angle sensor

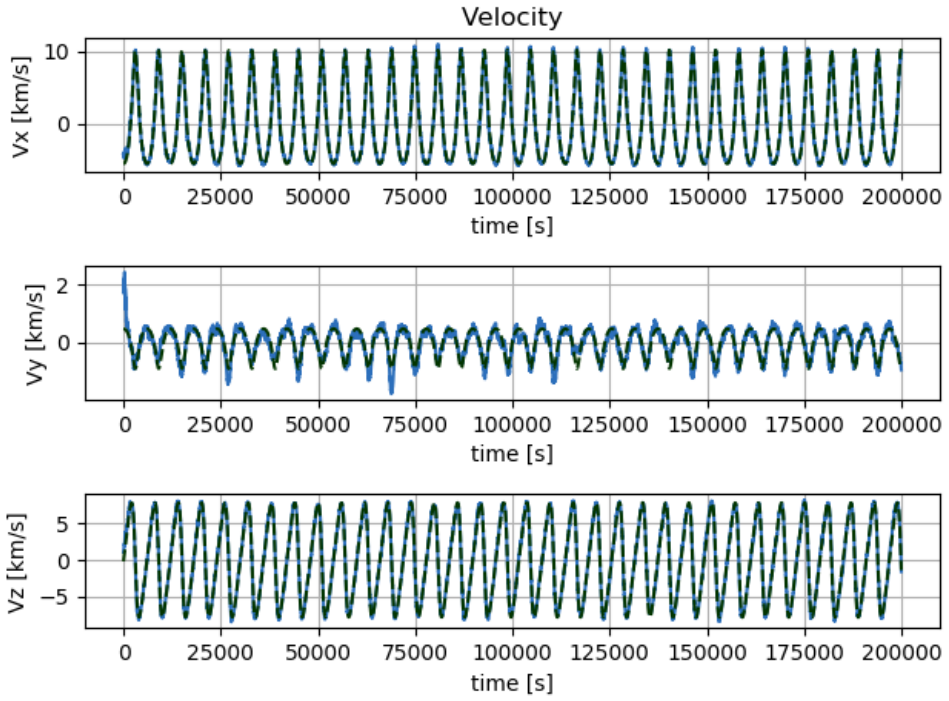


Figure D.10: Velocity using angle sensor

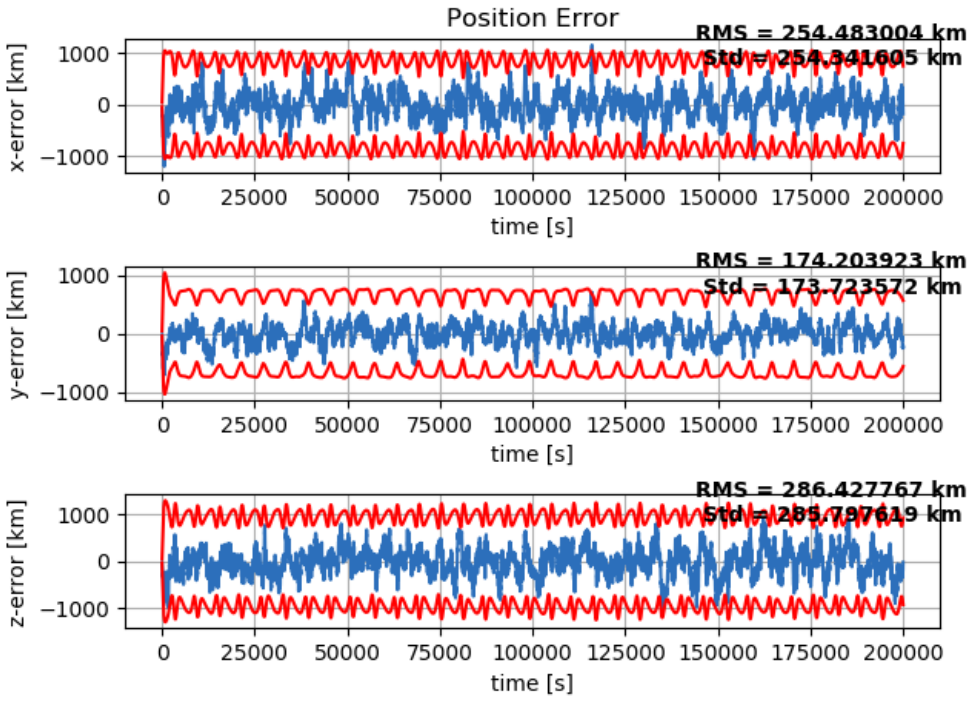


Figure D.11: Position errors using angle sensor

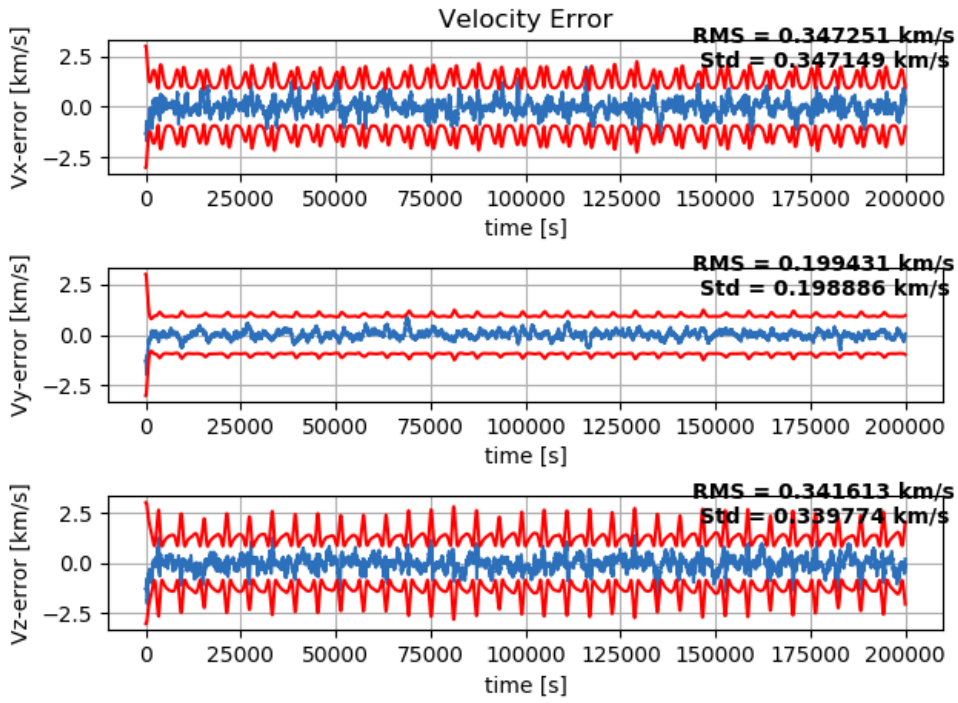


Figure D.12: Velocity errors using angle sensor

D.4. INTEGRATED

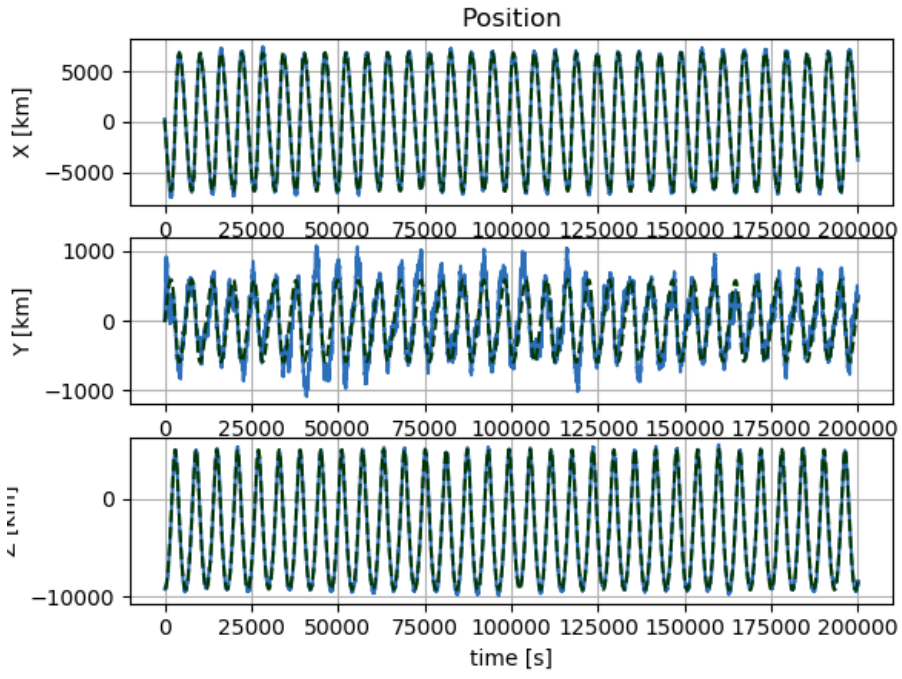


Figure D.13: Position using integrated angle and radial velocity sensors

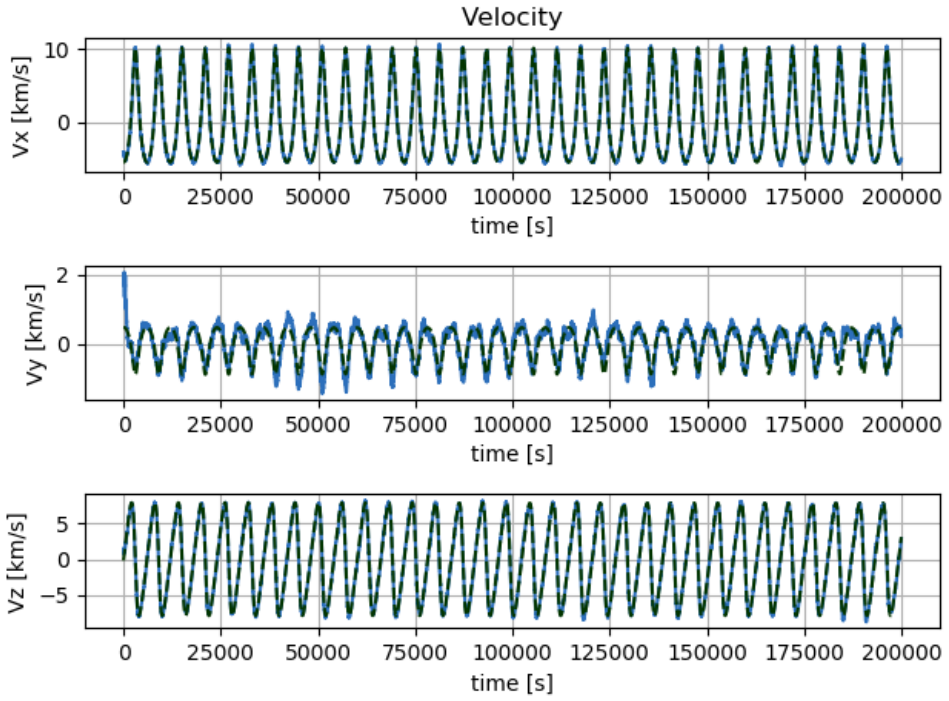


Figure D.14: Velocity using integrated angle and radial velocity sensors

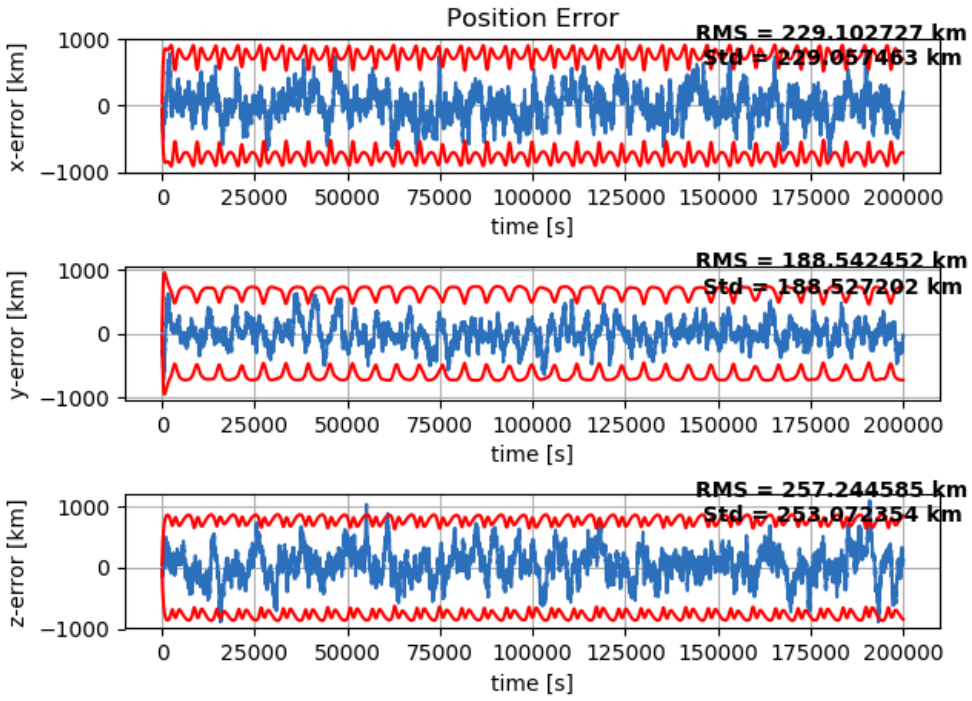


Figure D.15: Position errors using integrated angle and radial velocity sensors

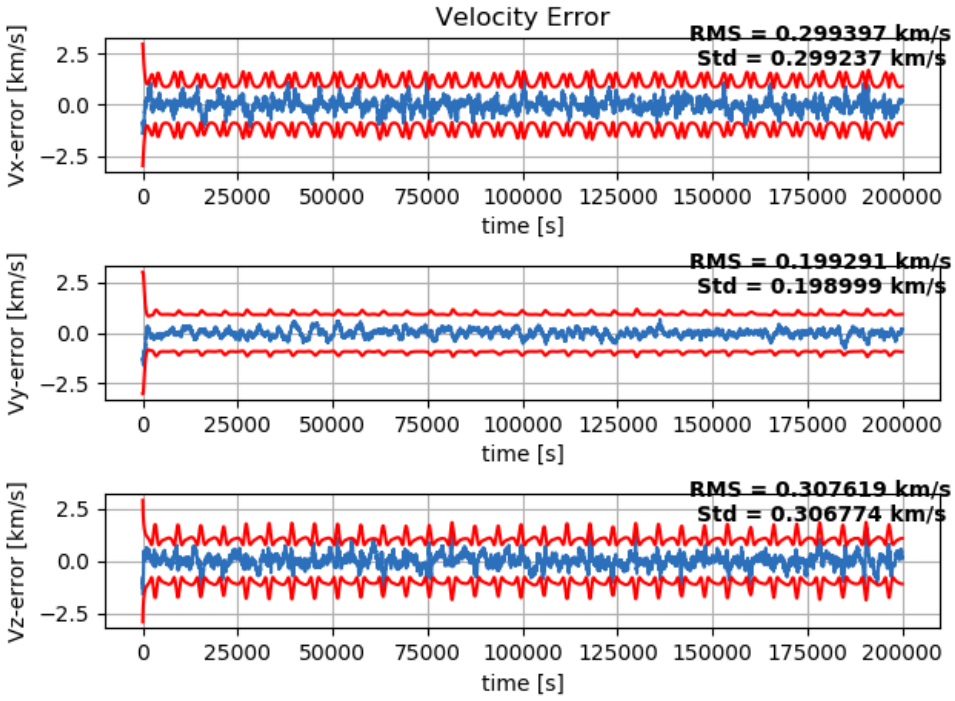


Figure D.16: Velocity errors using integrated angle and radial velocity sensors

D.4.1. DATA

Non PNAV LEO

Non PNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
No Meas	32705.32	5887.64	32301.66	0.3235	0.0520	0.2852	107989.00	21048.54	68959.74	5.45	0.64	5.56	56989.57	11639.77	34389.64	1.09	0.11	0.99	23631.54	2.20E-01
Radial vel	153.97	171.37	252.79	0.0683	0.0016	0.1037	866.26	1722.88	657.18	0.89	1.64	0.75	179.36	852.56	137.11	0.20	0.78	0.16	192.71	5.79E-02
Angle	75.12	36.65	87.99	0.1242	0.0099	0.0370	748.74	370.75	927.74	1.03	0.39	1.01	266.16	155.53	360.88	0.55	0.29	0.56	66.59	5.71E-02
INT	8.58	6.04	6.61	0.0150	0.0006	0.0050	505.30	358.48	539.70	0.59	0.36	0.56	64.31	79.91	64.95	0.17	0.19	0.14	7.07	6.87E-03

E

PULSAR TIMING ERROR

The following equation may be used to approximate the error in timing of pulsars:

$$c(\delta t_b - \delta t_{obs}) - \delta\hat{\mathbf{n}} \cdot \mathbf{r}' = \hat{n} \cdot \delta\mathbf{r}$$

the accuracy of navigation can be estimated using the timing errors calculated in the previous section and the following information about the positioning information about the three pulsars Crab, B1937+21 and B1821-24 as shown in E.1

Table E.1: Angular position errors for the three navigation pulsars. Data taken from [26].

Pulsar	B0531+21	B1821-24	B1937+21
RA error (mas)	5	0.9	0.002
RA error (nrad)	72.82	8.727	0.1164
DEC error (mas)	60	12.0	0.04
DEC error (nrad)	58.18	116.4	1.357

Figure E.1 shows only the RSS error due to pulsar position as a function of distance. This is clearly significant, to which the timing error plays a steadily decreasing role as a function of distance from the SSB.

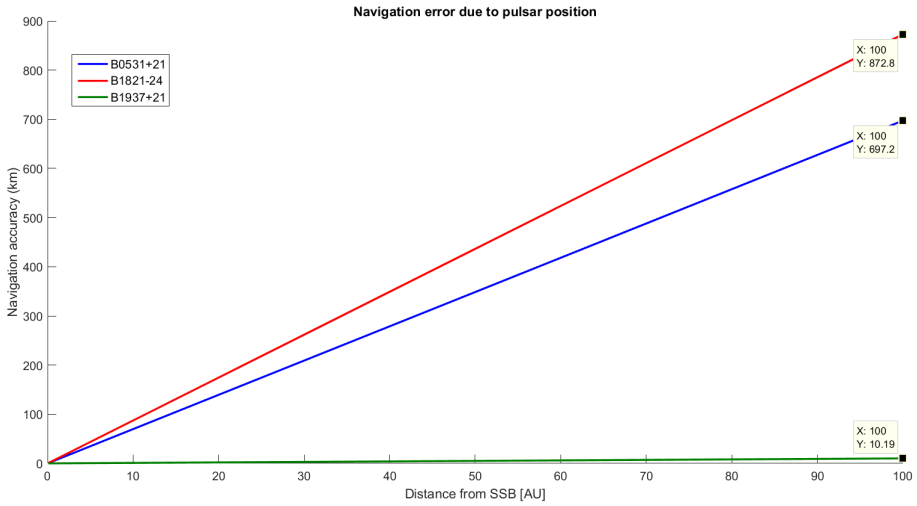


Figure E.1: Navigation error due to intrinsic uncertainty in the angular position of the pulsars

This shows that the distance the s/c is from the SSB clearly has a non-negligible effect on the positional accuracy.

X-RAY NAVIGATION ACCURACY

At a distance of 1 AU, the accuracy of pulsar timing on positing, including both timing and position error is shown by figure E.1

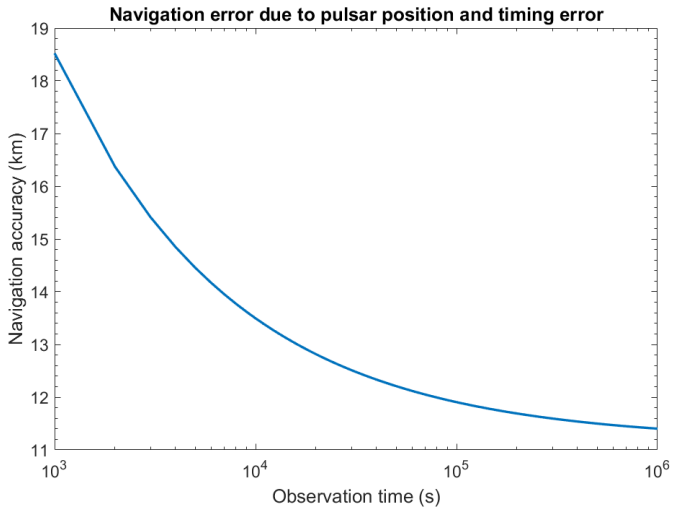


Figure E.2: Navigation error due to intrinsic uncertainty in the angular position and timing error of the pulsars at a distance of 1 AU.

F

Simulation Results

F.1. Deep Space Case

F.1.1. XNAV

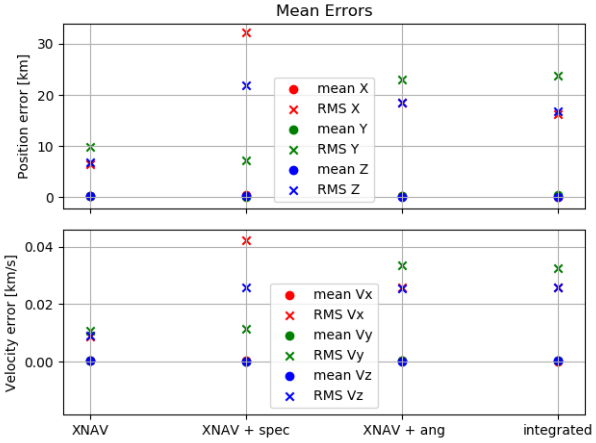


Figure F.1: XNAV for deep space case 1 m^2 area with 1500 s integration time

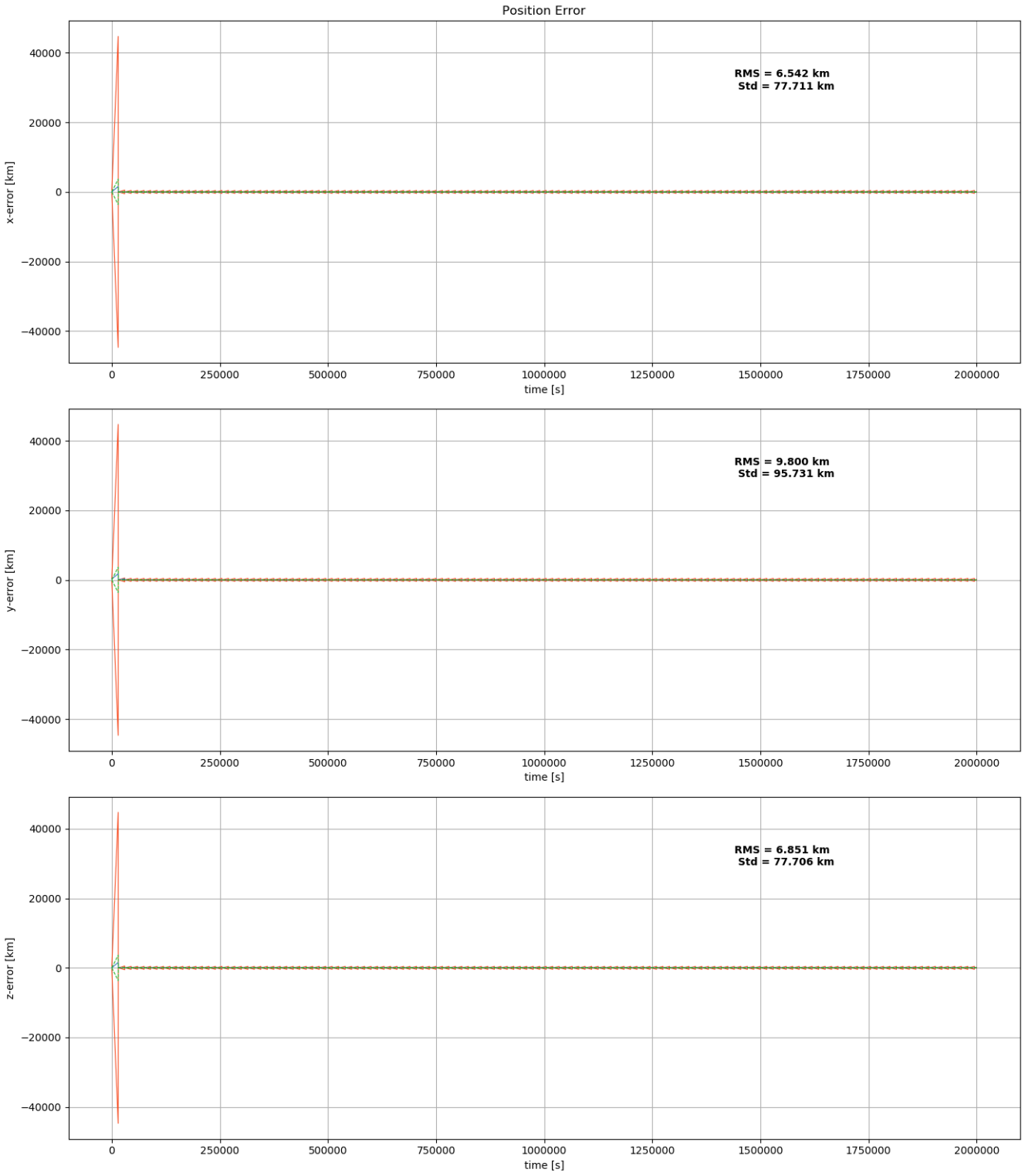


Figure F.2: XNAV-only deep space position error

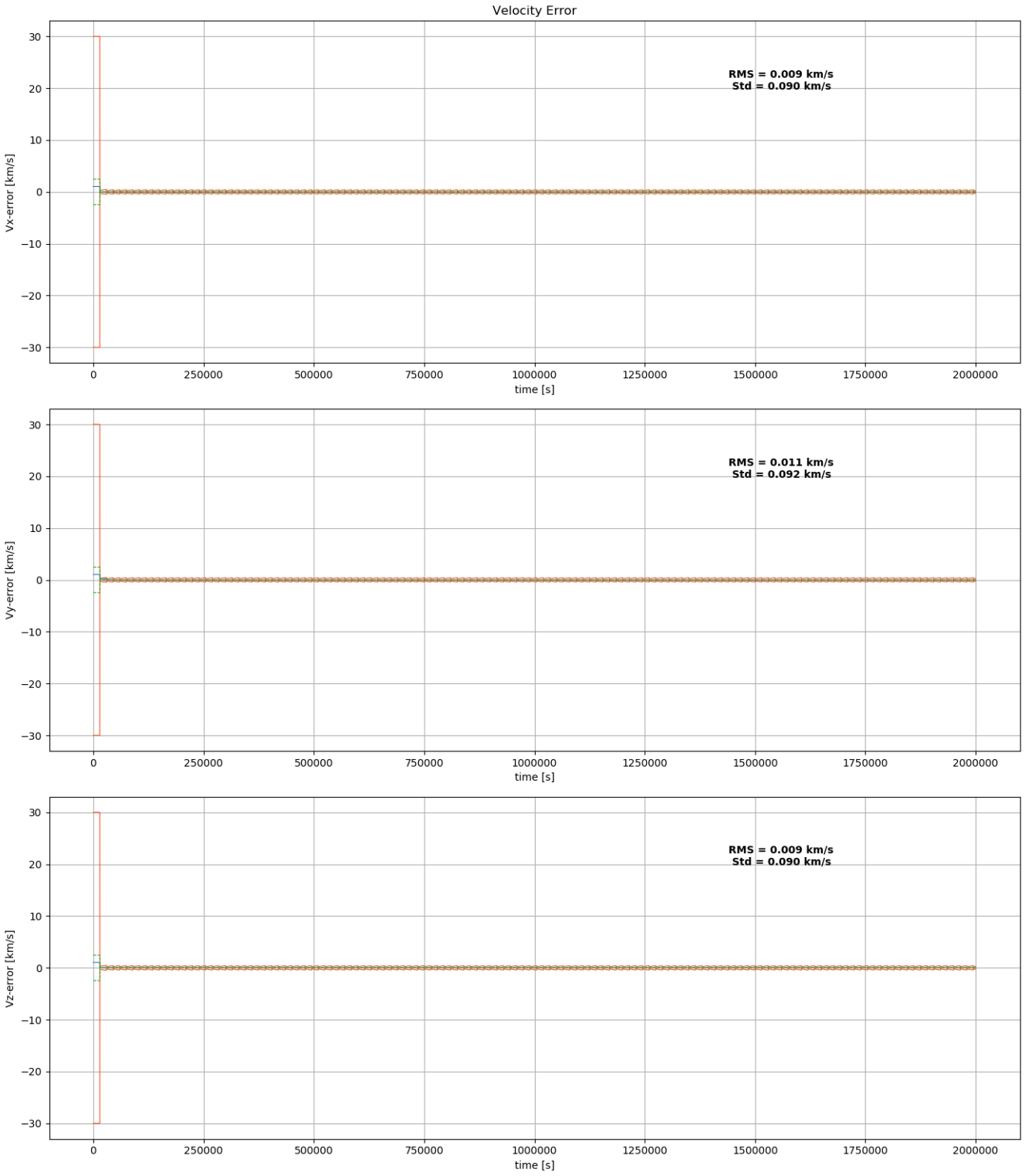


Figure F.3: XNAV-only deep space velocity error

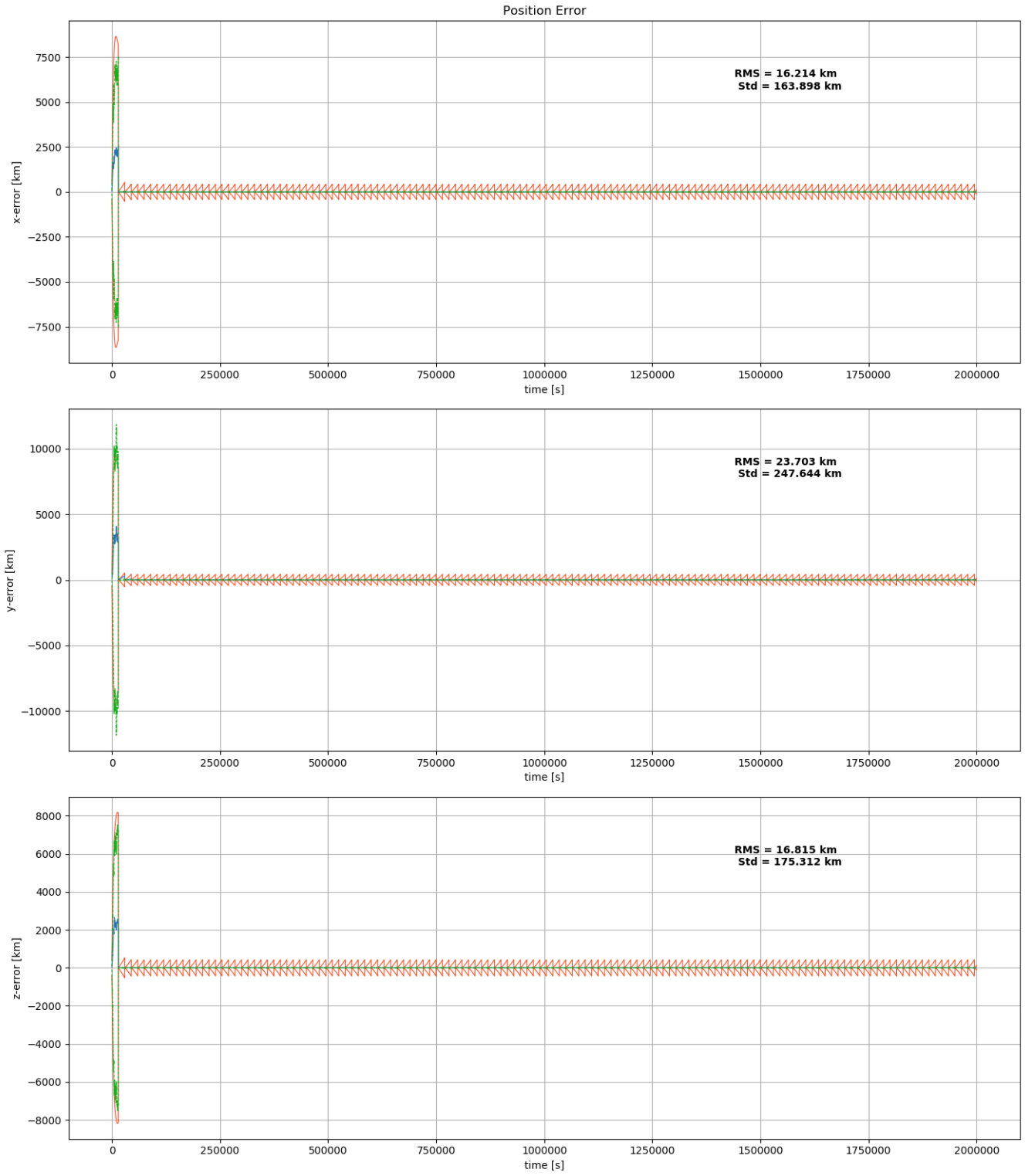


Figure F.4: Integrated deep space position error

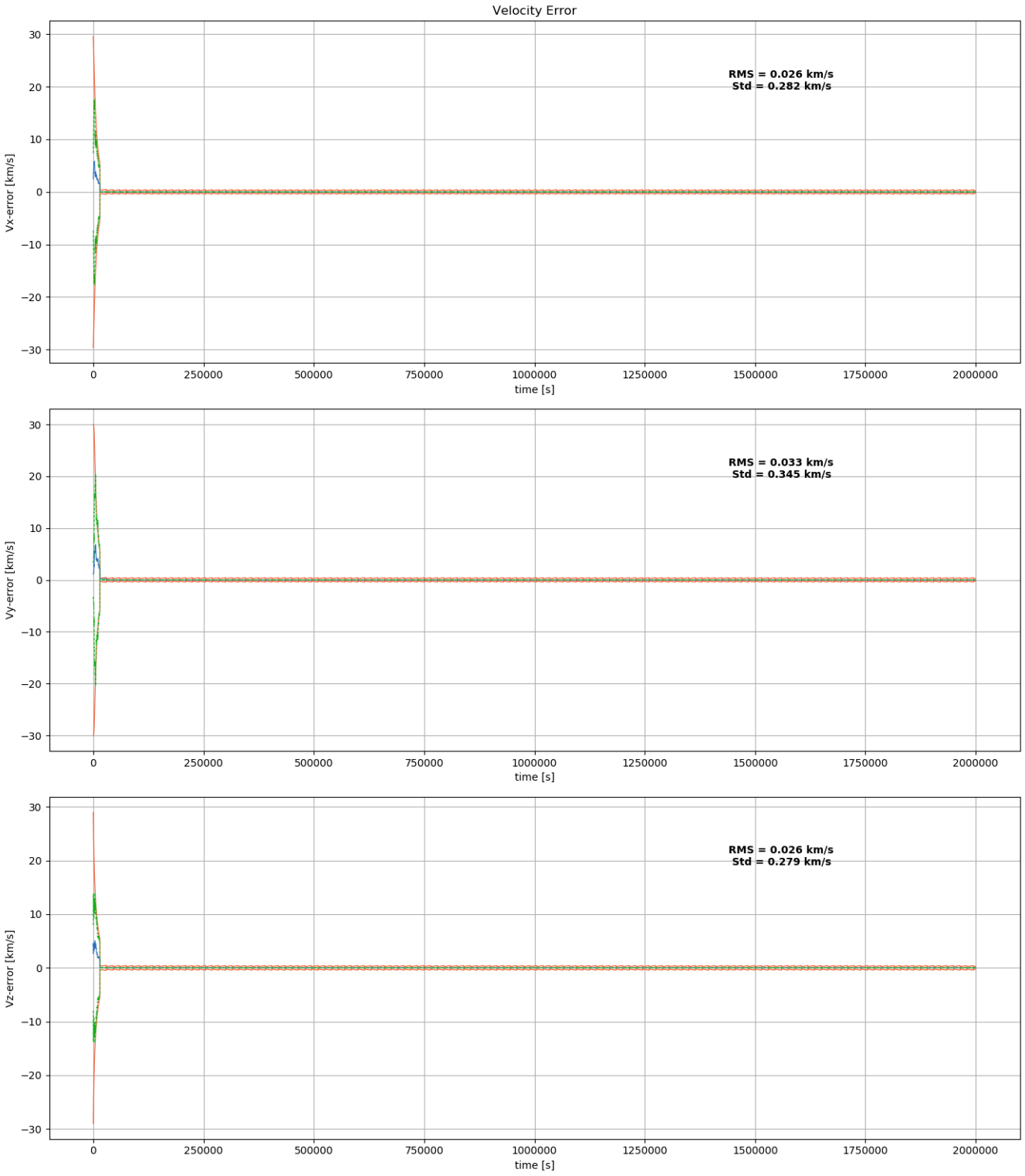


Figure F.5: Integrated deep space velocity error

F.1.2. RNAV

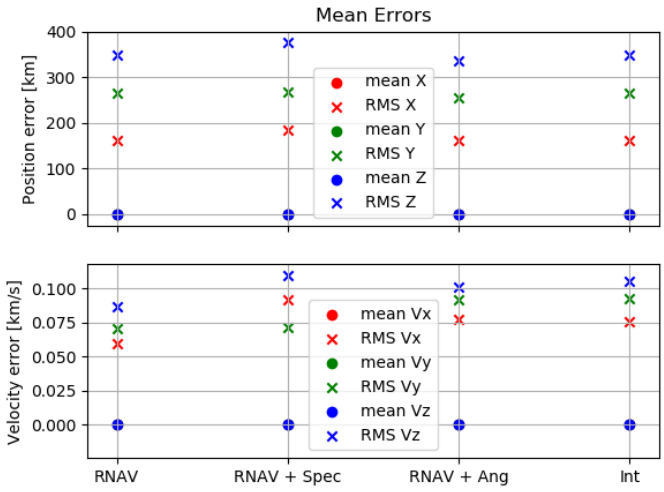


Figure F.6: RNAV for deep space case 100 m^2 area with 1500 s integration time

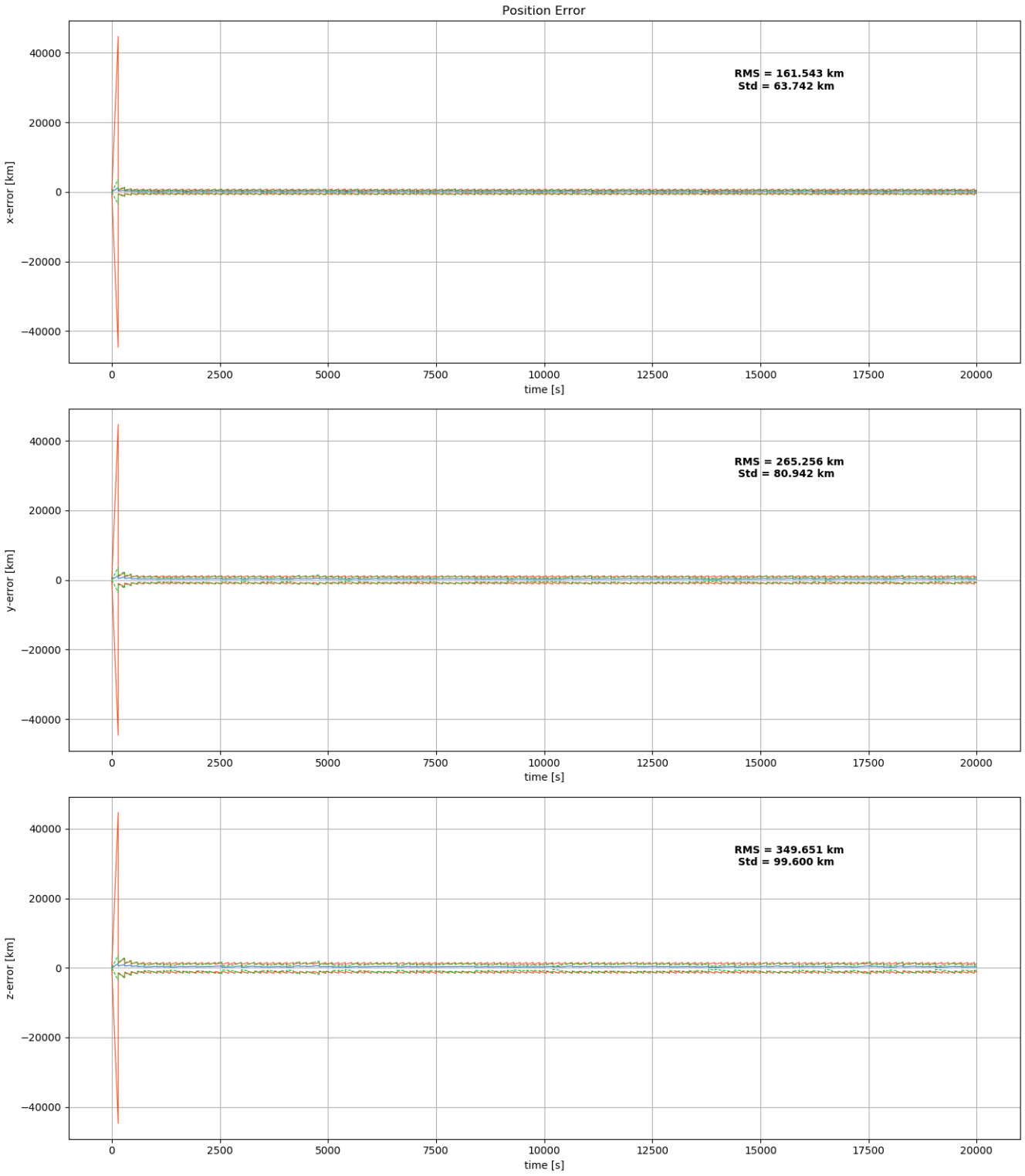


Figure F.7: RNAV-only deep space position error

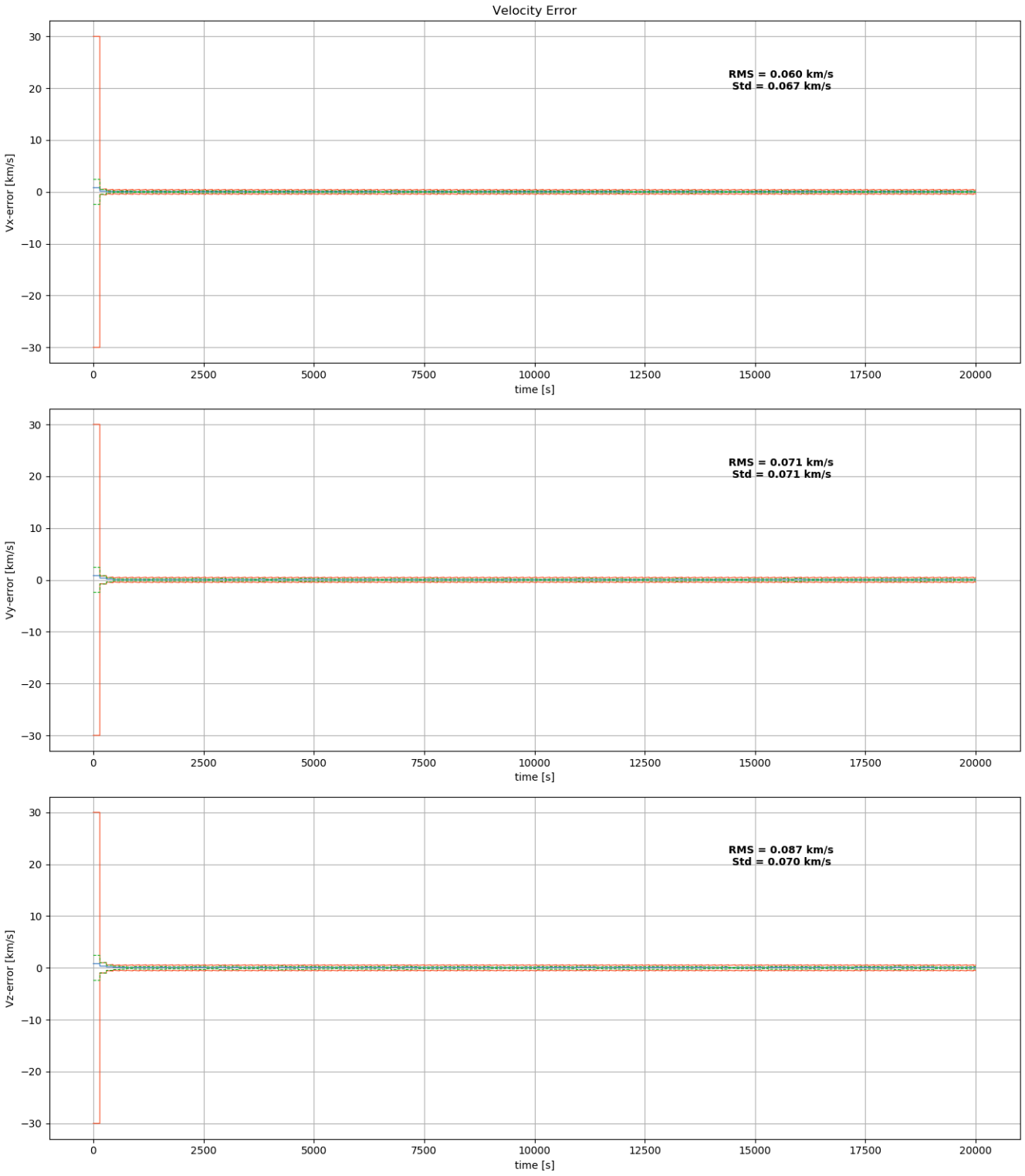


Figure F.8: RNAV-only deep space velocity error

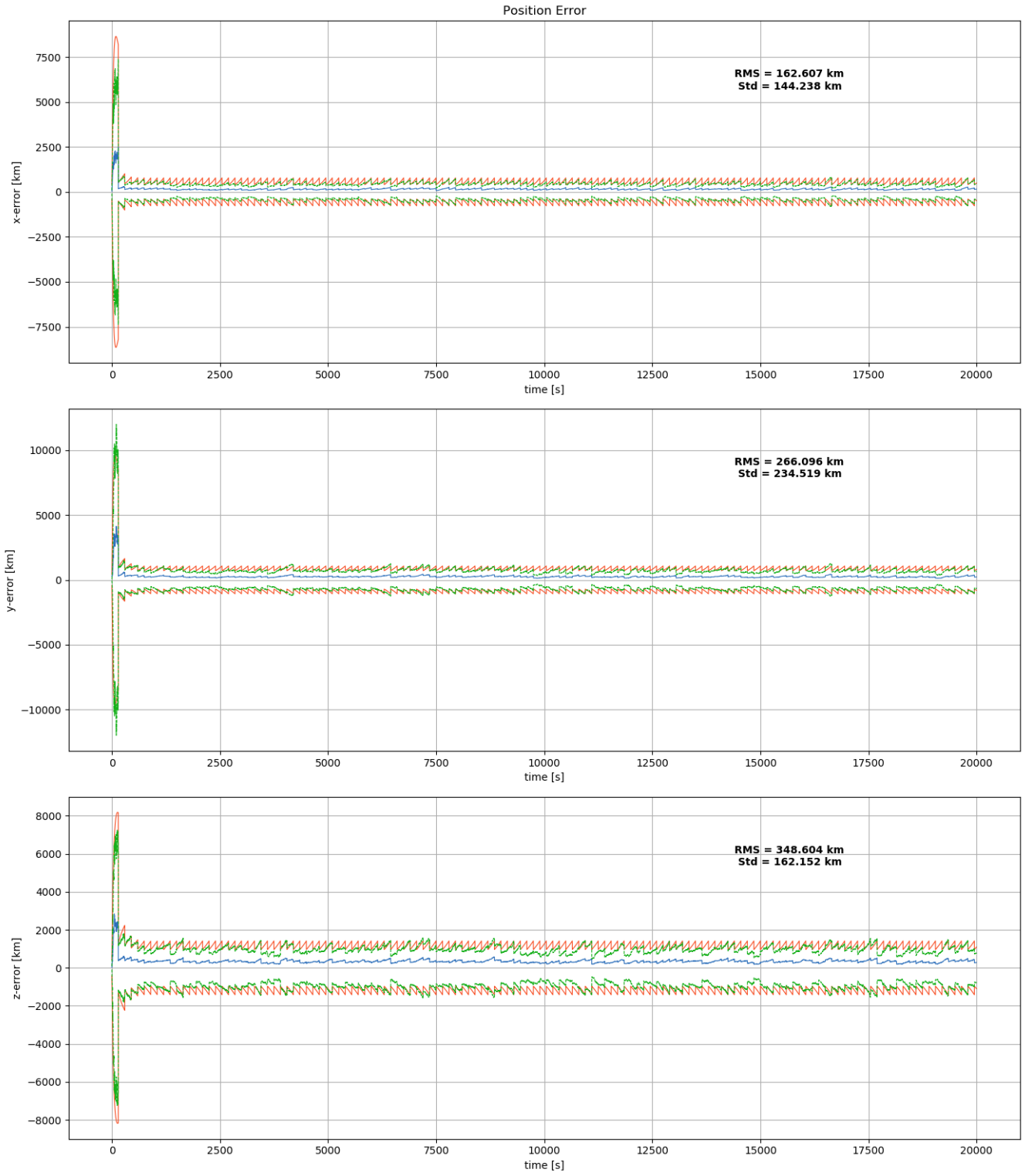


Figure F.9: Integrated deep space position error

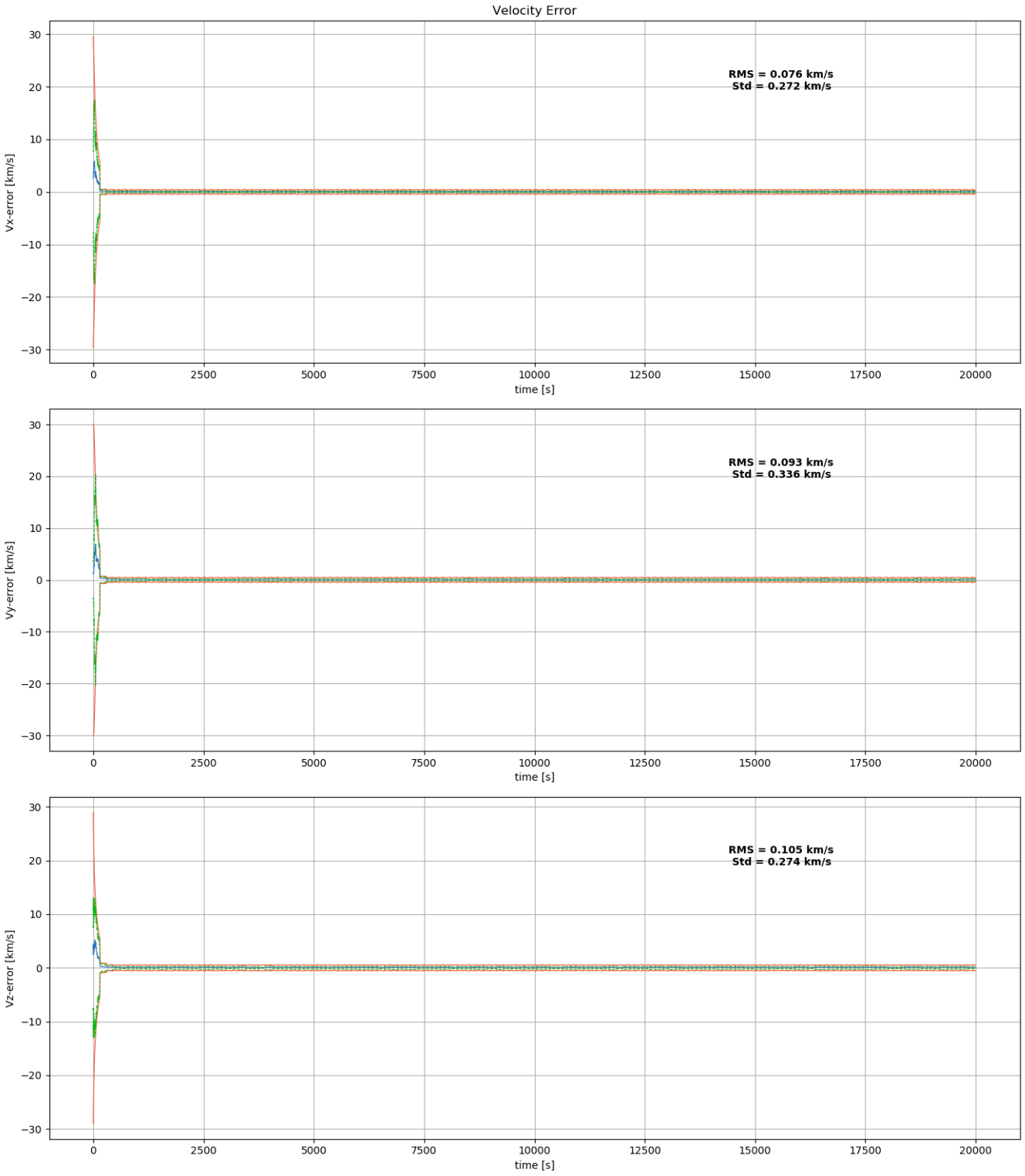
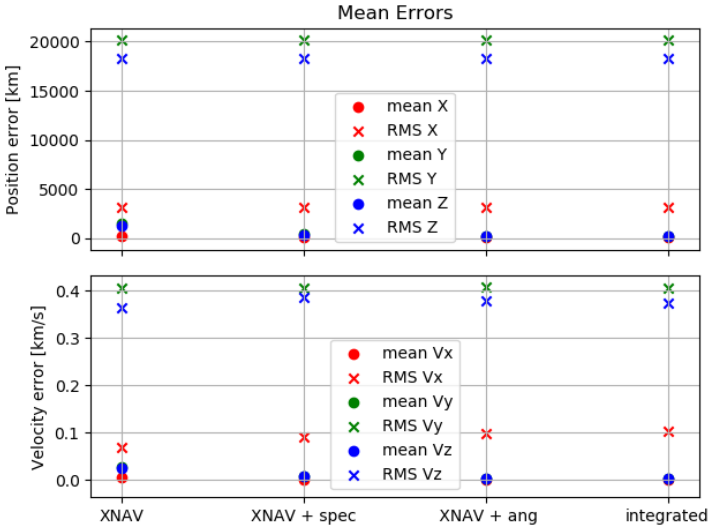
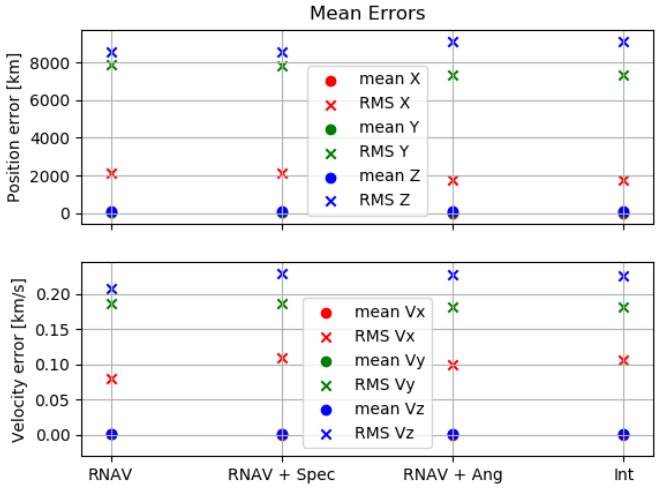


Figure F.10: Integrated deep space velocity error

F.2. CLOCK NOISE



(a) XNAV with clock error



(b) RNAV with clock error

Figure F.11: Example of addition of clock noise for Deep space PNAV

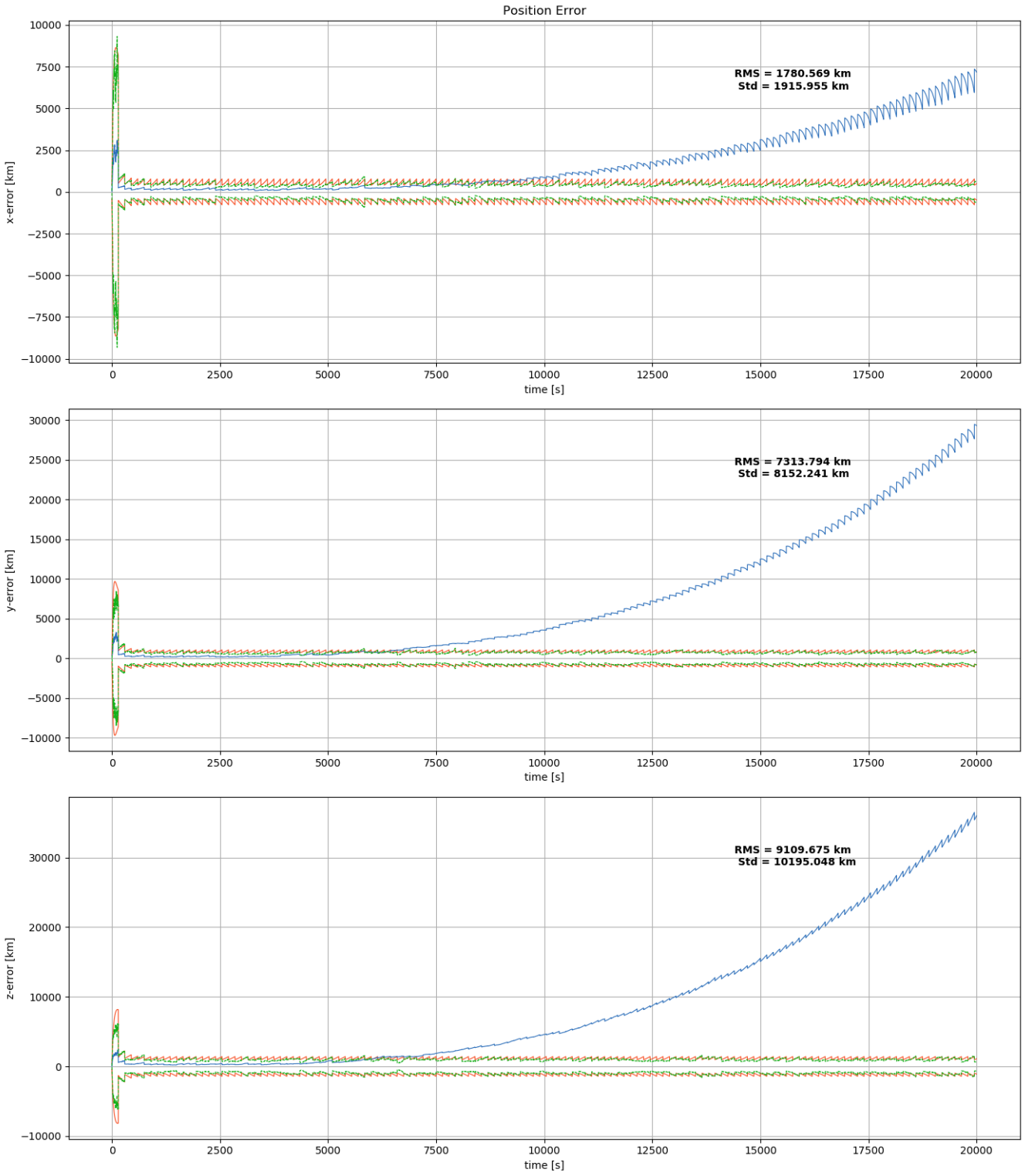
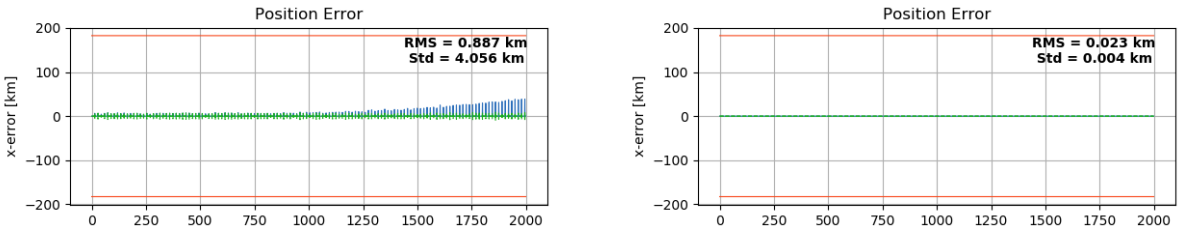


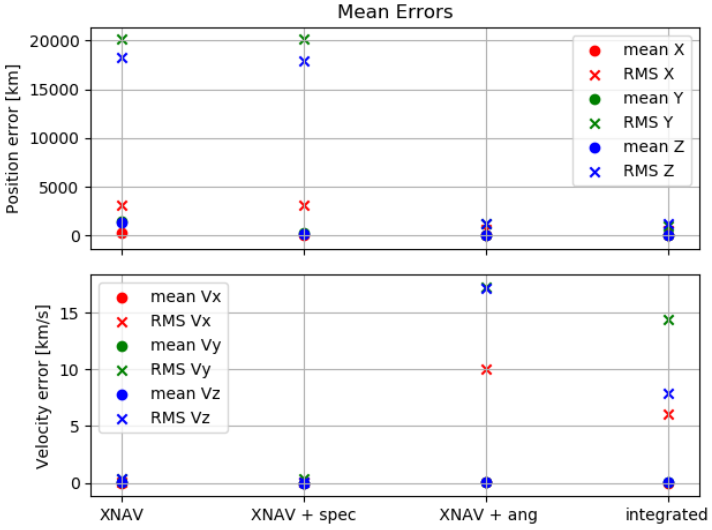
Figure F.12: Integrated RNAV deep space position error with clock noise



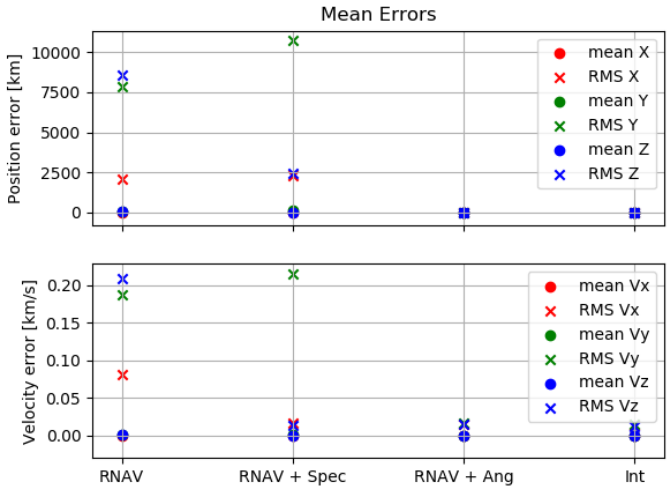
(a) XNAV with clock error

(b) RNAV with clock error

Figure F.13: Example of addition of clock noise with low noise additional sensors



(a) XNAV with clock error



(b) RNAV with clock error

Figure F.14: Addition of clock noise for Deep space PNAV with low noise sensors

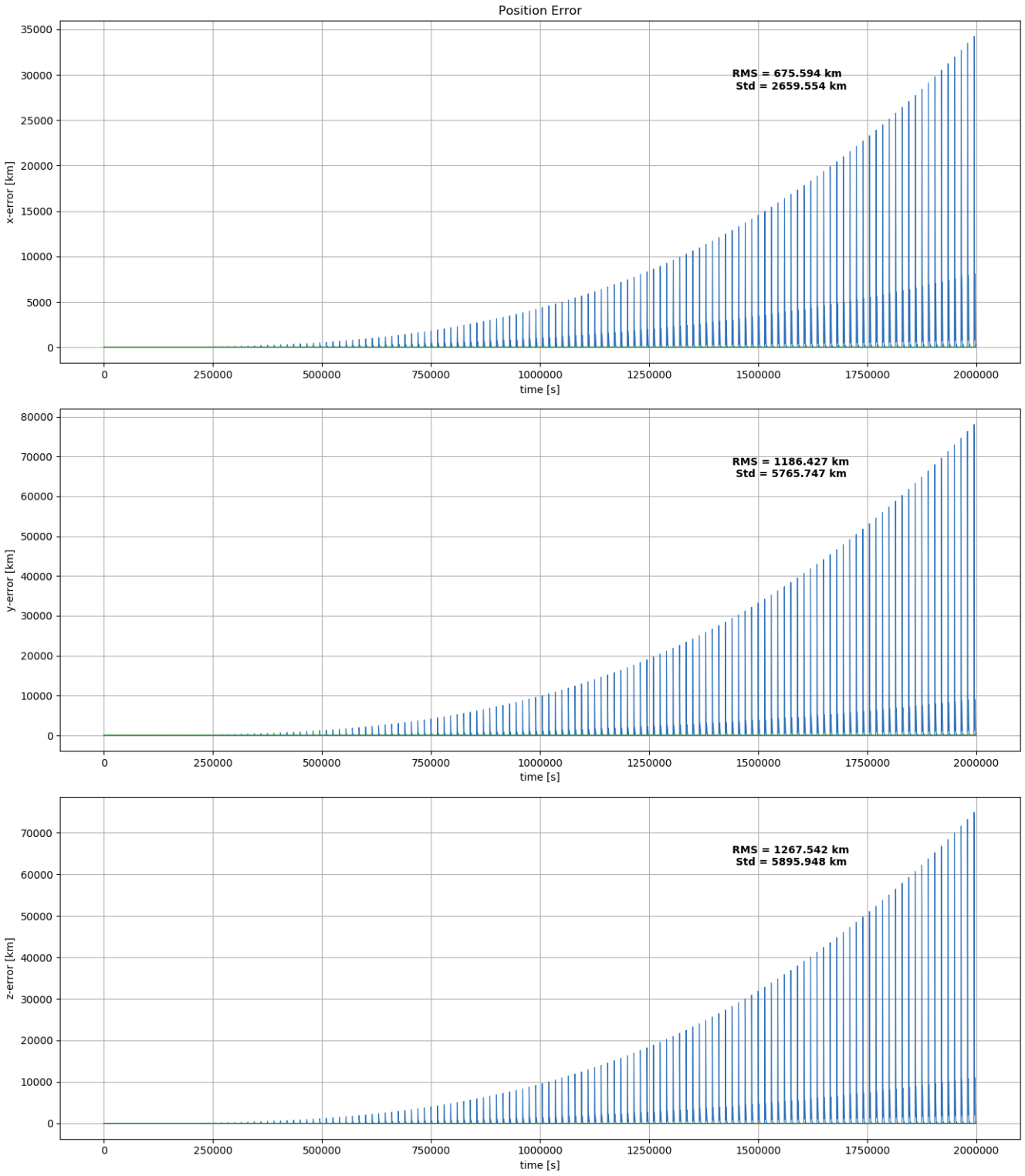


Figure F.15: Position error for integrated low noise sensors with XNAV for deep space with clock noise

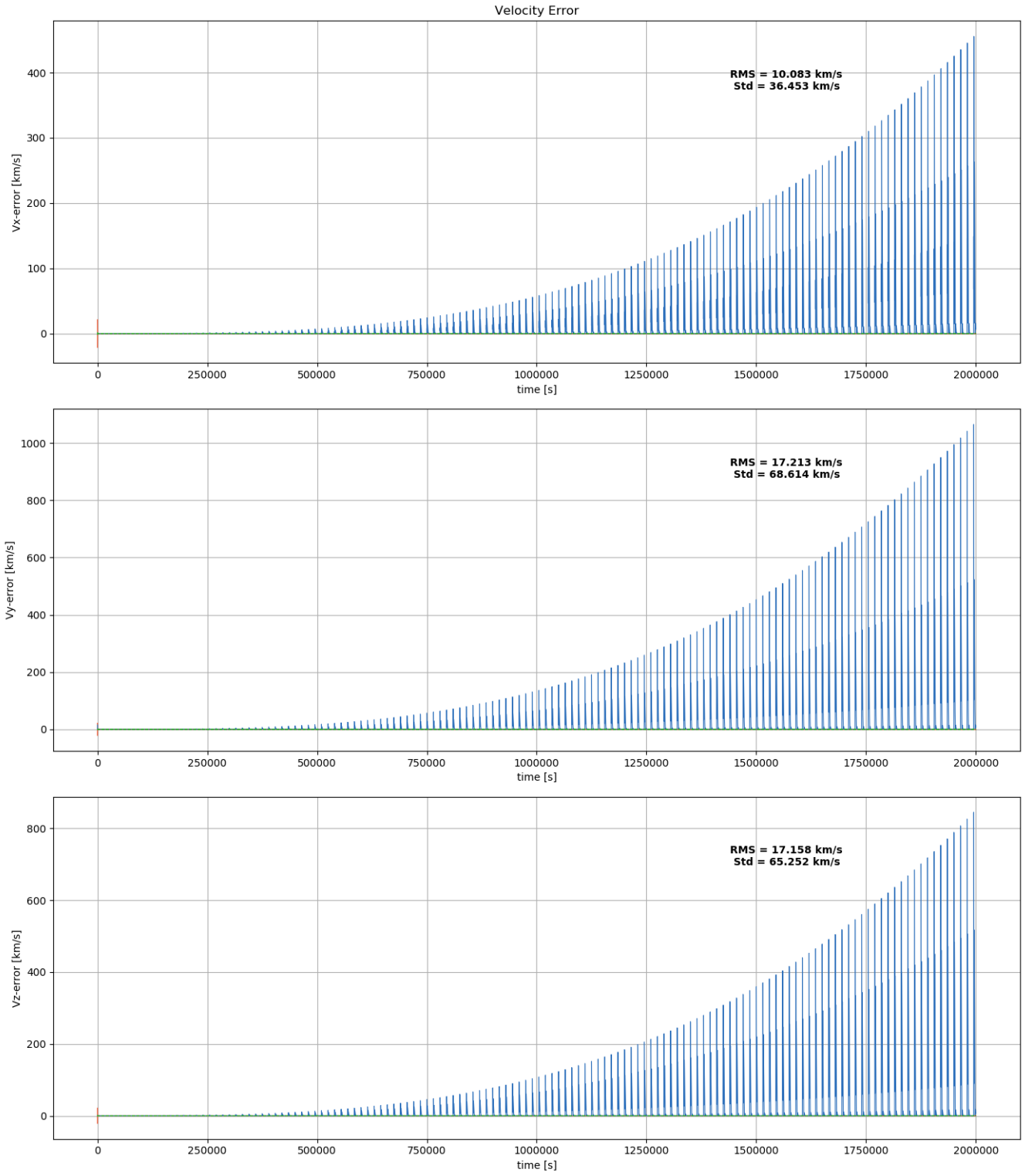


Figure F.16: Velocity error for integrated low noise sensors with XNAV for deep space with clock noise

F.3. PLANETARY ORBIT CASE

F.3.1. XNAV

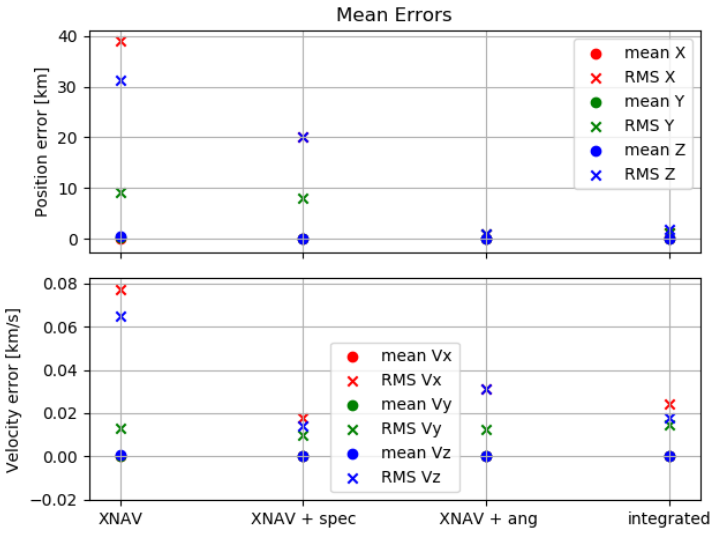


Figure F.17: XNAV for LEO case 100 m^2 area with 1500 s integration time

F.3.2. RNAV

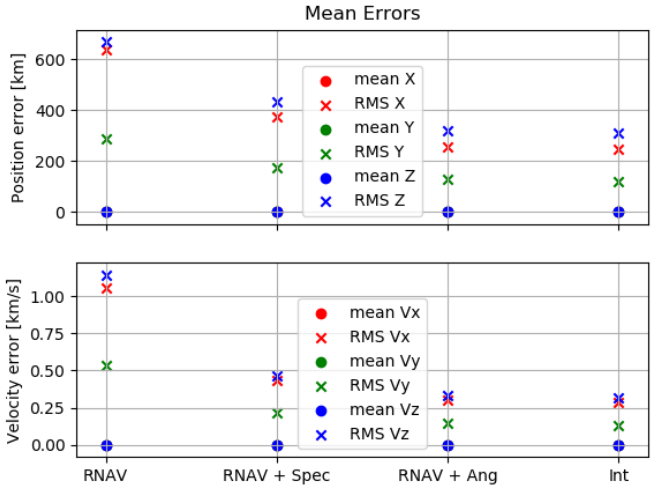


Figure F.18: RNAV for LEO case 100 m^2 area with 1500 s integration time

F.4. RNAV TABLES

RNAVdeepspace with Clock noise sensor nav q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	-2.05E+01	7.81E+01	8.44E+01	-3.97E-04	1.53E-03	1.64E-03	2.10E+03	7.86E+03	8.55E+03	8.06E-02	1.87E-01	2.08E-01	2.28E+03	8.78E+03	9.54E+03	8.07E-02	1.39E-01	1.44E-01	6.17E+03	1.59E-01
spec	-2.03E+01	7.79E+01	8.48E+01	-2.12E-04	1.44E-03	1.70E-03	2.11E+03	7.84E+03	8.58E+03	1.10E-01	1.86E-01	2.29E-01	2.29E+03	8.77E+03	9.54E+03	4.15E-01	1.35E-01	3.46E-01	6.18E+03	1.75E-01
ang	-1.71E+01	7.28E+01	9.04E+01	-2.06E-05	1.41E-03	1.87E-03	1.78E+03	7.31E+03	9.12E+03	1.00E-01	1.82E-01	2.27E-01	1.92E+03	8.15E+03	1.02E+04	2.83E-01	3.32E-01	3.31E-01	6.07E+03	1.70E-01
int	-1.73E+01	7.26E+01	9.01E+01	-1.43E-04	1.19E-03	1.80E-03	1.78E+03	7.31E+03	9.11E+03	1.05E-01	1.82E-01	2.26E-01	1.92E+03	8.15E+03	1.02E+04	3.34E-01	3.04E-01	2.95E-01	6.07E+03	1.71E-01

RNAVdeep space No Clock noisy sensor nav q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	8.17E-02	-3.72E-01	4.91E-01	-5.72E-05	-7.39E-05	-6.04E-05	1.62E+02	2.65E+02	3.50E+02	5.96E-02	7.06E-02	8.68E-02	6.37E+01	8.09E+01	9.96E+01	6.72E-02	7.08E-02	6.97E-02	2.59E+02	7.24E-02
spec	2.32E-01	-1.65E-01	2.96E-01	1.63E-04	-3.90E-05	1.48E-04	1.84E+02	2.68E+02	3.75E+02	9.16E-02	7.11E-02	1.09E-01	3.49E+02	6.24E+01	2.32E+02	4.27E-01	6.31E-02	3.13E-01	2.76E+02	9.07E-02
ang	-1.80E-01	4.95E-01	-4.73E-01	-9.42E-05	2.51E-04	-5.97E-05	1.62E+02	2.54E+02	3.36E+02	7.74E-02	9.22E-02	1.01E-01	1.93E+02	2.35E+02	2.08E+02	2.82E-01	3.84E-01	2.86E-01	2.51E+02	9.03E-02
int	2.01E-01	-3.53E-02	-9.22E-03	7.97E-06	1.21E-04	7.54E-05	1.63E+02	2.66E+02	3.49E+02	7.59E-02	9.27E-02	1.05E-01	1.44E+02	2.35E+02	1.62E+02	2.72E-01	3.36E-01	2.74E-01	2.59E+02	9.12E-02

RNAV Deep space With Clock errors Precise sensors q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	-2.05E+01	7.81E+01	8.44E+01	-3.97E-04	1.53E-03	1.64E-03	2.10E+03	7.86E+03	8.55E+03	8.06E-02	1.87E-01	2.08E-01	2.28E+03	8.78E+03	9.54E+03	8.07E-02	1.39E-01	1.44E-01	6.17E+03	1.59E-01
spec	-2.30E+01	1.07E+02	2.40E+01	-4.51E-06	1.87E-03	1.04E-06	2.31E+03	1.08E+04	2.42E+03	1.72E-02	2.15E-01	1.35E-02	2.77E+03	1.22E+04	2.96E+03	2.50E-03	1.77E-01	1.97E-03	5.16E+03	8.18E-02
ang	3.08E-03	3.35E-03	2.95E-03	4.12E-07	-6.97E-06	7.87E-07	1.28E+00	1.31E+00	1.35E+00	1.57E-02	1.67E-02	1.57E-02	2.33E+00	2.44E+00	2.02E+00	2.43E-02	1.07E-01	2.03E-02	1.31E+00	1.60E-02
int	3.46E-03	3.71E-03	3.17E-03	1.68E-07	-6.51E-06	2.52E-06	1.16E+00	1.23E+00	1.10E+00	1.41E-02	1.57E-02	1.26E-02	1.72E+00	1.94E+00	1.23E+00	1.32E-02	1.08E-01	6.05E-03	1.16E+00	1.41E-02

RNAVdeep space No Clock precise sensors q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	8.17E-02	-3.72E-01	4.91E-01	-5.72E-05	-7.39E-05	-6.04E-05	1.62E+02	2.65E+02	3.50E+02	5.96E-02	7.06E-02	8.68E-02	6.37E+01	8.09E+01	9.96E+01	6.72E-02	7.08E-02	6.97E-02	2.59E+02	7.24E-02
spec	-7.25E-02	1.53E-01	-1.90E-01	-2.41E-06	-2.25E-05	-1.62E-06	2.95E+01	8.03E+01	4.63E+01	1.70E-02	4.64E-02	1.34E-02	5.03E+00	5.03E+01	6.51E+00	3.16E-03	6.40E-02	2.50E-03	5.20E+01	2.56E-02
ang	2.18E-05	1.21E-04	1.41E-04	-1.35E-07	-7.60E-06	3.23E-07	9.48E-01	9.56E-01	1.05E+00	1.19E-02	1.27E-02	1.26E-02	2.72E-01	2.74E-01	2.97E-01	5.79E-03	1.07E-01	5.82E-03	9.85E-01	1.24E-02
int	5.34E-06	-2.12E-04	1.96E-04	1.55E-07	-7.27E-06	1.37E-06	8.75E-01	9.28E-01	8.76E-01	1.20E-02	1.28E-02	1.19E-02	1.65E-01	1.75E-01	1.68E-01	2.32E-03	1.08E-01	2.23E-03	8.93E-01	1.22E-02

RNAV LEO with Clock noise sensor nav q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	-1.08E+02	6.11E+02	6.16E+01	-8.76E-03	2.39E-03	4.02E-04	1.15E+04	6.18E+04	8.69E+03	4.37E+00	2.45E+00	6.76E+00	1.01E+04	7.05E+04	7.22E+03	3.59E+00	3.00E+00	4.37E+00	2.73E+04	4.53E+00
spec	-1.03E+02	6.09E+02	6.23E+01	-6.07E-03	5.25E-03	2.71E-03	1.15E+04	6.21E+04	8.54E+03	4.46E+00	3.03E+00	6.53E+00	1.05E+04	7.03E+04	7.12E+03	5.45E+00	6.95E+00	4.17E+00	2.74E+04	4.67E+00
ang	-9.76E+01	6.06E+02	5.71E+01	-7.30E-03	3.70E-03	5.51E-03	1.06E+04	6.14E+04	8.08E+03	4.48E+00	2.34E+00	6.82E+00	9.42E+03	7.03E+04	6.29E+03	5.71E+00	2.99E+00	5.18E+00	2.67E+04	4.54E+00
int	-1.08E+02	6.59E+02	6.18E+01	-1.02E-02	4.51E-03	6.13E-03	1.06E+04	6.13E+04	7.67E+03	4.04E+00	2.32E+00	6.23E+00	9.62E+03	7.04E+04	6.58E+03	3.91E+00	3.57E+00	4.48E+00	2.65E+04	4.20E+00

RNAV LEO No Clock noisy sensor nav q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	-2.73E-01	-1.88E-01	4.15E-01	1.39E-04	-6.02E-05	1.66E-04	6.35E+02	2.87E+02	6.69E+02	1.06E+00	5.31E-01	1.14E+00	3.47E+03	2.08E+03	3.25E+03	7.29E+00	4.44E+00	8.10E+00	5.30E+02	9.11E-01
spec	7.10E-01	1.08E-01	-2.53E-01	6.47E-04	-1.95E-05	-1.40E-04	3.72E+02	1.76E+02	4.31E+02	4.34E-01	2.19E-01	4.68E-01	5.94E+02	4.15E+02	5.84E+02	7.42E-01	5.82E-01	7.28E-01	3.26E+02	3.74E-01
ang	8.95E-01	3.14E-01	-9.55E-01	5.42E-04	1.75E-04	-1.11E-03	2.56E+02	1.26E+02	3.18E+02	3.03E-01	1.50E-01	3.32E-01	3.22E+02	2.59E+02	4.04E+02	4.83E-01	3.90E-01	5.17E-01	2.33E+02	2.62E-01
int	9.95E-01	2.04E-01	-4.74E-01	8.47E-04	4.94E-05	-2.95E-04	2.44E+02	1.17E+02	3.09E+02	2.85E-01	1.27E-01	3.17E-01	2.40E+02	2.44E+02	3.44E+02	3.70E-01	3.19E-01	4.23E-01	2.24E+02	2.43E-01

RNAV LEO With Clock errors Precise sensors q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	-1.08E+02	6.11E+02	6.16E+01	-8.76E-03	2.39E-03	4.02E-04	1.15E+04	6.18E+04	8.69E+03	4.37E+00	2.45E+00	6.76E+00	1.01E+04	7.05E+04	7.22E+03	3.59E+00	3.00E+00	4.37E+00	2.73E+04	4.53E+00
spec	-1.94E+01	6.50E+02	6.80E+00	-4.74E-07	4.89E-03	4.25E-05	1.93E+03	6.52E+04	1.05E+03	2.00E-01	1.96E+00	1.25E-01	2.06E+03	7.69E+04	1.18E+03	1.77E-01	1.81E+00	1.06E-01	2.27E+04	7.62E-01
ang	-6.71E-03	-3.18E-03	-8.49E-03	2.25E-06	-2.87E-07	1.55E-06	1.74E+00	1.36E+00	2.09E+00	3.90E-02	1.70E-02	4.24E-02	4.86E+00	1.94E+00	6.35E+00	5.81E-02	2.05E-02	6.99E-02	1.73E+00	3.28E-02
int	-6.33E-03	-2.98E-03	-8.53E-03	2.89E-06	-6.13E-07	-3.17E-06	1.85E+00	1.47E+00	2.53E+00	2.79E-02	1.80E-02	2.13E-02	3.31E+00	1.31E+00	3.85E+00	3.07E-02	2.13E-02	2.53E-02	1.95E+00	2.24E-02

RNAV LEO No Clock precise sensors q = 1e-5

RNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
RNAV	-2.73E-01	-1.88E-01	4.15E-01	1.39E-04	-6.02E-05	1.66E-04	6.35E+02	2.87E+02	6.69E+02	1.06E+00	5.31E-01	1.14E+00	3.47E+03	2.08E+03	3.25E+03	7.29E+00	4.44E+00	8.10E+00	5.30E+02	9.11E-01
spec	-2.66E-03	2.29E-01	1.31E-03	-1.41E-06	1.85E-04	-2.17E-07	2.79E+01	7.38E+01	2.76E+01	1.72E-02	7.50E-02	1.35E-02	1.21E+01	3.59E+01	1.35E+01	3.26E-03	4.86E-02	2.61E-03	4.31E+01	3.52E-02
ang	2.42E-05	1.23E-04	1.42E-04	5.25E-06	-4.16E-07	-4.61E-07	9.57E-01	9.55E-01	1.06E+00	3.05E-02	1.22E-02	3.07E-02	2.74E-01	2.73E-01	2.98E-01	2.47E-02	5.66E-03	1.69E-02	9.89E-01	2.45E-02
int	-1.07E-04	-3.02E-04	9.85E-05	2.64E-06	-9.99E-07	2.35E-06	1.29E+00	1.22E+00	1.83E+00	2.39E-02	1.45E-02	1.77E-02	4.29E-01	3.59E-01	8.65E-01	9.67E-03	5.87E-03	6.38E-03	1.45E+00	1.87E-02

F.5. XNAV TABLES

XNAV deep space With clock noise 1e-05

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	2.25E+02	1.44E+03	1.30E+03	4.63E-03	2.85E-02	2.58E-02	3.15E+03	2.02E+04	1.82E+04	6.87E-02	4.06E-01	3.65E-01	3.56E+03	2.29E+04	2.07E+04	9.23E-02	3.57E-01	3.24E-01	1.39E+04	2.80E-01
Radial vel	6.10E+01	3.89E+02	3.51E+02	1.94E-03	7.49E-03	6.96E-03	3.16E+03	2.02E+04	1.82E+04	9.22E-02	4.05E-01	3.85E-01	3.55E+03	2.29E+04	2.07E+04	3.55E-01	3.56E-01	4.28E-01	1.39E+04	2.94E-01
angle	3.15E+01	2.02E+02	1.82E+02	7.47E-04	3.75E-03	3.41E-03	3.18E+03	2.02E+04	1.82E+04	9.94E-02	4.09E-01	3.80E-01	3.57E+03	2.29E+04	2.06E+04	2.82E-01	4.40E-01	4.28E-01	1.39E+04	2.96E-01
integrated	3.15E+01	2.02E+02	1.82E+02	5.75E-04	3.87E-03	3.61E-03	3.17E+03	2.02E+04	1.82E+04	1.03E-01	4.07E-01	3.75E-01	3.57E+03	2.29E+04	2.06E+04	3.46E-01	4.32E-01	4.05E-01	1.39E+04	2.95E-01

XNAV deep space NO clock noise 1e-05

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	2.80E-01	3.16E-01	2.83E-01	3.73E-04	3.39E-04	3.75E-04	6.54E+00	9.80E+00	6.85E+00	8.89E-03	1.08E-02	9.02E-03	7.77E+01	9.57E+01	7.77E+01	9.00E-02	9.15E-02	9.00E-02	7.73E+00	9.56E-03
Radial vel	3.61E-01	-5.65E-02	1.83E-01	4.66E-04	-1.09E-04	1.19E-04	3.22E+01	7.16E+00	2.19E+01	4.24E-02	1.14E-02	2.58E-02	4.22E+02	6.90E+01	3.04E+02	4.82E-01	9.96E-02	2.92E-01	2.04E+01	2.65E-02
angle	-2.12E-01	-2.33E-01	-1.64E-02	-2.77E-04	-3.42E-04	2.47E-05	1.85E+01	2.30E+01	1.85E+01	2.59E-02	3.36E-02	2.54E-02	2.02E+02	2.40E+02	2.09E+02	2.85E-01	3.79E-01	2.88E-01	2.00E+01	2.83E-02
integrated	-1.15E-01	-3.49E-01	1.26E-01	-1.14E-04	-4.51E-04	2.54E-04	1.62E+01	2.37E+01	1.68E+01	2.59E-02	3.27E-02	2.59E-02	1.64E+02	2.48E+02	1.75E+02	2.82E-01	3.45E-01	2.79E-01	1.89E+01	2.82E-02

XNAV deep space NO clock noise precise sensors q=1e-5

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	2.80E-01	3.16E-01	2.83E-01	3.73E-04	3.39E-04	3.75E-04	6.54E+00	9.80E+00	6.85E+00	8.89E-03	1.08E-02	9.02E-03	7.77E+01	9.57E+01	7.77E+01	9.00E-02	9.15E-02	9.00E-02	7.73E+00	9.56E-03
Radial vel	-3.89E-04	1.65E-02	-4.30E-03	1.83E-07	-1.02E-05	-5.57E-06	4.66E+00	7.06E+00	3.05E+00	1.70E-02	8.82E-03	1.34E-02	2.08E+00	6.15E+01	1.41E+00	2.93E-03	6.93E-02	2.29E-03	4.92E+00	1.31E-02
angle	-2.41E-04	-1.61E-04	-2.05E-04	-8.73E-07	-8.03E-06	-6.35E-08	9.30E-01	9.42E-01	1.04E+00	1.19E-02	1.27E-02	1.26E-02	2.53E-01	2.43E-01	2.61E-01	5.65E-03	1.07E-01	5.70E-03	9.70E-01	1.24E-02
integrated	5.22E-05	6.71E-05	-1.75E-04	1.78E-06	-6.30E-06	-2.45E-07	8.51E-01	9.08E-01	8.57E-01	1.20E-02	1.28E-02	1.18E-02	1.84E-01	1.79E-01	1.65E-01	2.08E-03	1.07E-01	1.97E-03	8.72E-01	1.22E-02

XNAV deep space WITH clock noise precise sensors q=1e-5

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	2.25E+02	1.44E+03	1.30E+03	4.63E-03	2.85E-02	2.58E-02	3.15E+03	2.02E+04	1.82E+04	6.87E-02	4.06E-01	3.65E-01	3.56E+03	2.29E+04	2.07E+04	9.23E-02	3.57E-01	3.24E-01	1.39E+04	2.80E-01
Radial vel	3.26E+01	2.13E+02	1.89E+02	2.35E-05	4.27E-03	1.17E-04	3.10E+03	2.02E+04	1.80E+04	1.89E-02	4.07E-01	2.38E-02	3.52E+03	2.29E+04	2.04E+04	1.14E-02	3.59E-01	6.53E-02	1.38E+04	1.50E-01
angle	3.19E+00	7.46E+00	7.66E+00	-1.46E-02	-5.85E-02	-5.63E-02	6.76E+02	1.19E+03	1.27E+03	1.01E+01	1.72E+01	1.72E+01	2.66E+03	5.77E+03	5.90E+03	3.65E+01	6.86E+01	6.53E+01	1.04E+03	1.48E+01
integrated	5.48E+00	9.70E+00	1.27E+01	-3.34E-03	-4.83E-02	-2.09E-02	5.61E+02	9.91E+02	1.27E+03	6.01E+00	1.44E+01	7.86E+00	2.43E+03	5.54E+03	5.58E+03	2.29E+01	6.22E+01	3.28E+01	9.40E+02	9.44E+00

XNAV LEO With clock noise noisy sensors q=1e-5

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	-5.12E+01	2.35E+03	8.01E+02	-2.48E-03	2.17E-02	1.43E-02	3.06E+03	5.88E+04	2.06E+04	4.34E+00	1.52E+00	4.92E+00	3.41E+03	6.71E+04	2.27E+04	3.52E+00	1.26E+00	3.78E+00	2.75E+04	3.59E+00
Radial	-4.19E+01	2.02E+03	6.90E+02	-1.52E-04	1.79E-02	1.08E-02	3.05E+03	5.88E+04	2.06E+04	4.37E+00	1.58E+00	4.94E+00	3.44E+03	6.71E+04	2.26E+04	3.55E+00	1.27E+00	3.77E+00	2.75E+04	3.63E+00
Angle	-1.23E+01	5.86E+02	2.00E+02	5.66E-04	5.17E-03	3.03E-03	2.98E+03	5.87E+04	2.06E+04	4.34E+00	1.50E+00	4.97E+00	3.36E+03	6.70E+04	2.27E+04	3.48E+00	1.25E+00	3.82E+00	2.74E+04	3.60E+00
integrated	-1.26E+01	5.86E+02	2.01E+02	-3.79E-05	4.92E-03	3.47E-03	3.11E+03	5.87E+04	2.07E+04	4.48E+00	1.52E+00	5.05E+00	3.33E+03	6.70E+04	2.26E+04	3.42E+00	1.21E+00	3.63E+00	2.75E+04	3.68E+00

XNAV LEO NO clock noise noisy sensors q=1e-5

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	-1.37E-01	1.65E-01	5.22E-01	-1.92E-04	2.14E-04	6.60E-04	3.90E+01	9.24E+00	3.13E+01	7.72E-02	1.32E-02	6.51E-02	2.73E+02	6.33E+01	1.98E+02	3.98E-01	7.27E-02	2.83E-01	2.65E+01	5.18E-02
Radial vel	5.89E-01	1.47E-01	2.59E-01	8.35E-04	1.81E-04	6.27E-04	5.43E+01	1.05E+01	5.71E+01	1.06E-01	1.50E-02	1.04E-01	3.83E+02	6.91E+01	3.65E+02	5.73E-01	8.03E-02	5.20E-01	4.06E+01	7.49E-02
angle	-1.54E-01	-1.41E-01	-9.51E-02	-1.10E-04	-1.18E-04	-8.65E-05	1.30E+02	5.26E+01	1.20E+02	2.11E-01	7.03E-02	2.45E-01	6.79E+02	2.84E+02	5.05E+02	9.11E-01	4.17E-01	1.46E+00	1.01E+02	1.75E-01
integrated	-6.70E-02	-2.88E-01	1.19E-01	1.52E-05	-3.26E-04	2.78E-04	4.26E+01	2.83E+01	5.68E+01	8.36E-02	3.71E-02	1.01E-01	2.17E+02	2.43E+02	2.79E+02	3.46E-01	3.22E-01	4.11E-01	4.26E+01	7.39E-02

XNAV LEO NO clock noise precise sensors q=1e-5

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	-1.37E-01	1.65E-01	5.22E-01	-1.92E-04	2.14E-04	6.60E-04	3.90E+01	9.24E+00	3.13E+01	7.72E-02	1.32E-02	6.51E-02	2.73E+02	6.33E+01	1.98E+02	3.98E-01	7.27E-02	2.83E-01	2.65E+01	5.18E-02
Radial vel	-2.06E-02	4.28E-02	-4.17E-02	5.92E-06	7.69E-05	1.33E-05	2.01E+01	7.94E+00	2.02E+01	1.78E-02	9.64E-03	1.40E-02	1.62E+01	3.12E+01	1.60E+01	3.67E-03	4.19E-02	2.99E-03	1.61E+01	1.38E-02
angle	2.48E-04	1.85E-04	-5.73E-05	1.97E-06	-9.68E-07	1.38E-06	9.51E-01	9.46E-01	1.05E+00	3.10E-02	1.24E-02	3.14E-02	2.06E-01	2.20E-01	2.23E-01	2.31E-02	6.96E-03	1.56E-02	9.82E-01	2.49E-02
integrated	8.48E-05	1.05E-04	9.21E-05	1.16E-06	-9.47E-07	1.07E-07	1.26E+00	1.18E+00	1.77E+00	2.40E-02	1.45E-02	1.78E-02	4.41E-01	3.78E-01	8.42E-01	9.59E-03	6.04E-03	6.70E-03	1.40E+00	1.88E-02

XNAV LEO WITH clock noise precise sensors q=1e-5

XNAV	Means						RMS						STD							
	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	x [km]	y [km]	z [km]	vx [km/s]	vy [km/s]	vz [km/s]	POS MEAN	VEL MEAN
XNAV	-5.12E+01	2.35E+03	8.01E+02	-2.48E-03	2.17E-02	1.43E-02	3.06E+03	5.88E+04	2.06E+04	4.34E+00	1.52E+00	4.92E+00	3.41E+03	6.71E+04	2.27E+04	3.52E+00	1.26E+00	3.78E+00	2.75E+04	3.59E+00
Radial vel	-2.38E+01	1.13E+03	3.81E+02	-8.89E-05	1.08E-02	2.15E-04	1.25E+03	5.90E+04	1.98E+04	2.28E-01	2.12E+00	1.47E-01	1.77E+03	6.70E+04	2.24E+04	2.16E-01	2.84E+00	1.44E-01	2.67E+04	8.31E-01
angle	8.29E+00	2.49E+01	1.28E+01	-4.61E-02	-2.56E-01	-1.02E-01	1.92E+03	4.01E+03	2.34E+03	3.31E+01	6.50E+01	2.86E+01	7.83E+03	2.01E+04	9.73E+03	1.21E+02	2.67E+02	9.78E+01	2.75E+03	4.22E+01
integrated	1.29E+01	2.93E+01	2.15E+01	-1.95E-02	-2.35E-01	-3.84E-02	1.42E+03	3.38E+03	2.15E+03	2.10E+01	6.02E+01	1.23E+01	6.64E+03	1.96E+04	9.04E+03	8.22E+01	2.64E+02	4.44E+01	2.32E+03	3.12E+01