Signal Processing Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2024-05

M.Sc. Thesis

Multi-Sensor Fusion for Localization of a Lunar Micro Rover Using Non-Vision Sensors

Anitha Sarah Koshy

5528402

Multi-Sensor Fusion for Localization of a Lunar Micro Rover Using Non-Vision Sensors

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Anitha Sarah Koshy
born in Bengaluru, India

This work was performed in:

Signal Processing Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Multi-Sensor Fusion for Localization of a Lunar Micro Rover Using Non-Vision Sensors"** by **Anitha Sarah Koshy** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 8 May 2024

Chairman: _____

Dr. Raj T. Rajan

Advisor: _____

Dr. Raj T. Rajan

Committee Members: _____

Dr.ir. Chris J.M. Verhoeven

_____

Dr. Arash Noroozi

Abstract

Autonomous navigation is a critical aspect of robotic systems, particularly in hostile and uncertain environments, and robot localization is central to navigation. Robot localization establishes its position within its surroundings. This thesis addresses the challenge of robot localization, focusing on a lunar-like environment with constraints such as limited computational resources and the use of non-visual based sensors.

In this thesis, three sensors — wheel encoders (WE), Sun sensor (SS), and inertial measurement unit (IMU) — are employed for localization. Each sensor contributes distinct information regarding position and orientation. However, individual sensor measurements suffer from inherent inaccuracies and errors, especially the IMU's reliance on integration over time, leading to significant drift.

To mitigate these challenges, three fusion methods are explored: sensor selection based on predefined thresholds, Kalman filtering, and weighted fusion. Results indicate substantial improvements in localization accuracy compared to individual sensor measurements. The weighted fusion method, in particular, demonstrates superior performance by assigning appropriate importance, according to their accuracy, to each sensor's information, resulting in significantly reduced positioning errors. The maximum localization error using this method is 92m, which is smaller than reported in the literature. Further, the maximum localization percentage error over 65m is around 8%, which is comparable to the literature with visual sensors. The weighted fusion method introduces only a marginal increase in the computational complexity. Thus, this method stands out for its simplicity and delivers results superior to those documented in existing literature for non-visual sensors.

Despite promising results, the research is met with certain hurdles, notably the availability and consistency of datasets. The reliance on existing datasets, such as the Devon Island Rover dataset, highlights the need for standardized and comprehensive datasets for thorough testing and validation. Calibration inconsistencies and verification issues further underscore the complexity of real-world implementation.

Nevertheless, the findings of this thesis offer insights into the integration of multiple sensors for enhanced localization in lunar-like environments. By leveraging complementary sensor data and employing efficient fusion techniques, the proposed approach enables more accurate and reliable navigation of lunar micro rovers, thus advancing the capabilities of autonomous robotic systems for future lunar exploration missions.

Acknowledgments

I extend my heartfelt gratitude to Dr. Raj T. Rajan my advisor, for his invaluable assistance throughout the writing process of this thesis and for providing feedback that enhanced the presentation of my work.

To my parents, whose unwavering support and guidance have afforded me opportunities and encouraged me to strive for excellence. I am grateful to my brother and grandparents, for their boundless love and encouragement throughout my upbringing; and my extended family for their blessings.

Special appreciation goes to Nathan for his continuous love and care, as well as to his family for opening their home to me.

I express my sincere thanks to my friends, housemates, and colleagues in Delft, whose companionship has made the city feel like home. I am equally thankful to my friends from school and college, whose unwavering friendship and support have been instrumental in shaping my journey thus far.

Finally, I would like to express genuine gratitude towards my counselor and therapists, Paula, Jan and Simi for helping me navigate all aspects of my life.

Anitha Sarah Koshy
Delft, The Netherlands
8 May 2024

Contents

List of Figures

List of Tables

The following list shows some symbols and operators which will be used in the thesis.

Symbol	**Description**
x	Horizontal position
y	Vertical position
θ	Heading angle or orientation angle or yaw angle
$\mathbf{R}$	Rotation matrix
$\mathbf{C}$	Transformation matrix
Ψ	GAST
$\mathbf{S}$	State
$\mathbf{S}_t$	State at defined time
δ/Δ	change
δ_x/Δ_x	change in specific variable
t	time
ϕ	latitude
λ	longitude
π	Pi
σ	standard deviation
σ^2	variance
$\mathbf{A}$	State space system or transition matrix
$\mathbf{B}$	State space input matrix
$\mathbf{H}$	State space measurement matrix
z	measurement
k	Time step
u	control input
w	process noise
v	measurement noise
Q	Covariance matrix
$E[\cdot]$	mean/average

Symbol	Description
I	Reference Frames
s	position
v	velocity
a	acceleration
ω	angular velocity
Γ	Noise gain
l	IMU frame
G	Global frame
$\mathbf{b}_g$	Gyroscope measurement bias
$\mathbf{b}_a$	Accelerometer measurement bias
$\mathbf{n}_g$	Gyroscope white Gaussian noise
$\mathbf{n}_a$	Accelerometer white Gaussian noise
γ	Roll
β	pitch
α	yaw
$\mathbf{s}_F$	Predicted sun vector
$\mathbf{s}_S$	Measured sun vector
$\mathbf{u}_{Si}$	q-Method Observation vector
$\mathbf{u}_{Fi}$	q-Method Prediction vector
m	Number of measurements
$\mathbf{g}_F$	Gravity vector
k	heading weighting
P	Covariance
B_{ij}	Element of $\mathbf{B}$ at row i and column j
$\mathbb{R}^N$	Set of real vectors of length N
$\mathbb{R}^{MXN}$	Set of real matrices of size M X N

The following list tabulates some commonly used abbreviations which will be used in the thesis.

Abbreviation	**Description**
SLAM	Simultaneous Localization And Mapping
GPS	Global Positioning System
LiDAR	Light Detection And Ranging
IMU	Inertial Measurement Unit
UHF-FRID	Ultra High Frequency Radio frequency Identification
AI	Artificial Intelligence
UWB	Ultra-Wide Band
VLC	Visible Light Communication
IR	Infrared
ToA	Time of Arrival
ToR	Time of Flight
FOV	Field of View
GTIL	Ground in the Loop
ECI	Earth Centered Initial
ECF	Earth Centered Fixed
GAST	Greenwich Apparent Sidereal Time
JD	julian Day
LZ	Lunar Zebro
MEMS	Micro-Electromechanical Systems
DoF	Degrees of Freedom
AHRS	Attitude Heading Reference Systems
ICC	Instantaneous centre of curvature
KF	Kalman Filter
RMSE	Root Mean Square Error

1 Introduction

Robots have become integral to various sectors of humanity and are omnipresent in society. Their importance lies in their ability to perform tasks autonomously, often in challenging or hazardous environments. The Lunar Zebro is a micro-rover that students of the TU Delft are currently developing [1]. (A brief description of the Lunar Zebro is provided in § 2.4.1, page 17.) Pictured in Fig. 1.1, the space mission aims to take flight in 2025 to explore diverse scientific objectives. This thesis was conceived to resolve the localization aspect of the rover's mission. This chapter examines the localization problem and the need for robust localization technologies. The investigation includes the different types of information available to the robot, how they are used for localization, and finally, the difficulties and problems associated with localization.

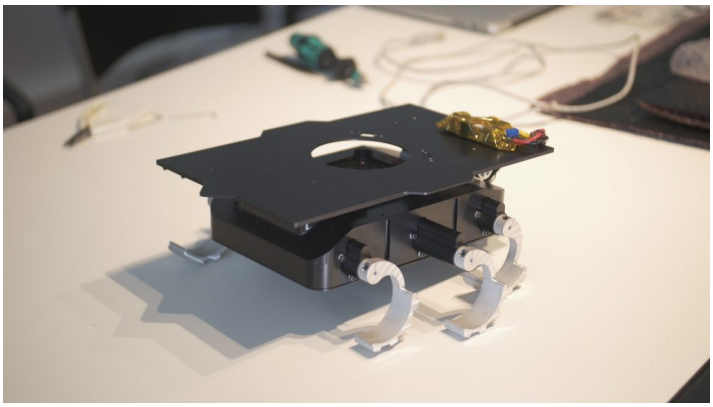


Figure 1.1: The Lunar Zebro [1]

1.1 Motivation

Robot navigation is the cornerstone of autonomous robotics, enabling robots to move through complex environments, make decisions about their paths, and reach their destinations without human intervention. This critical field of robotics encompasses a wide range of applications, from self-driving cars to space exploration, robots aiding in healthcare, and those enhancing the logistics industry. The ability of robots to navigate effectively and safely is at the heart of their utility and impact in today's world, promising increased efficiency, safety, and innovative solutions to many chal-

lenges. Some authors, like in [2], believe that localization is the "most fundamental problem to providing robots truly autonomous capabilities".

1.1.1 Robot Navigation and Localization

Robot navigation consists of finding the answers to the following three basic questions [3]:

1. Where is the robot?
2. Where is the robot going?
3. How should the robot get there?

While robot localization aims to answer the first question *Where is the robot now?*, setting a target goal and path planning are the concepts involved in answering the latter two questions.

Accurate localization lets robots know their precise position and orientation, ensuring they can navigate, interact, and perform tasks precisely. This capability is particularly critical in applications such as autonomous vehicles, where safety and precise positioning are paramount. Additionally, robot localization supports efficient path planning, resource optimization, and the ability to adapt to dynamic surroundings. In essence, it serves as the foundation for robots to function intelligently and safely, making it a fundamental element in the field of robotics and automation.

Localization implies that one needs to find the location and orientation of the robot relative to its environment. In other words, one must find the robot's (x, y, θ) configuration, with respect to the global or local frame, and not a topological frame [2]. Together, the parameters (x, y, θ) define a robot's pose, allowing one to pinpoint its location and the direction it is facing within a 2D environment.

As mentioned, a robot's pose is expressed using the three parameters. The x coordinate represents the robot's position along the horizontal axis in the 2D space. It specifies how far the robot is from a reference point (usually the origin) in the horizontal direction. A positive x-value indicates movement to the right, while a negative x-value represents movement to the left. Similarly, the y coordinate represents the robot's position along the vertical axis. Finally, θ represents the robot's orientation or angular position in the 2D space. It is typically measured in radians and indicates the angle the robot faces relative to a reference direction, often the horizontal axis (x axis). For example, a $\theta = 0$ value implies that the robot is facing the positive x axis, while other values of θ represent different orientations. The orientation angle is also called *heading angle*, and these terms are used interchangeably throughout this report.

1.2 Robot Localization

The localization of a robot plays a pivotal role in the effective operation of any robot, particularly a lunar rover. Accordingly, this topic is discussed in some detail here.

1.2.1 The Localization Problem

As mentioned earlier in Section 1.1.1, the aim is to determine the robot's pose, i.e., the robot's (x, y, θ) configuration. Based on information on initial position, self-localization can be divided into two sub-problems: *pose tracking* and *global localization* [4]. In the context of pose tracking, the starting robot pose is already established, and the primary objective of localization is to detect and rectify minor discrepancies in the robot's odometry as the robot navigates its path. Position tracking thus involves odometry and sensor data. In global localization, the robot must determine its pose using limited and incomplete observed information, even with uncertainty regarding its initial position. Of the two, global localization is the more challenging problem.

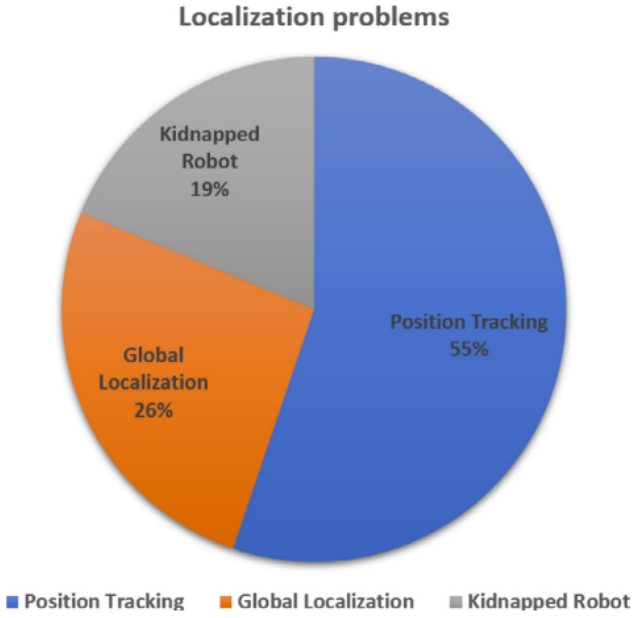


Figure 1.2: The percentage of each category of mobile robot localization problems surveyed in [5]

Apart from pose tracking and global localization, the robot may sometimes find itself in unfamiliar or arbitrary locations, either during pose tracking or through abduction to an undisclosed area. This scenario is commonly encountered in what is known as the *kidnapped robot problem*, where the robot is aware of its abduction [5]. Consequently,

the need for kidnap recovery becomes essential for any autonomous robot. Typically, the robot relies on its current sensor data to estimate its pose in most situations.

In [5], the authors surveyed around a hundred papers dealing with the robot localization problem. They found that the pose tracking problem is majorly studied compared to the other two. Their findings are depicted in Fig. 1.2.

The robot's environment also influences the localization problem. Environments may be static or dynamic. In a static environment, the localization process is relatively straightforward, as the robot is the sole mobile entity. However, in a dynamic environment, localization becomes considerably more complex because other moving objects can lead to confusion about the robot's position. Autonomous movement control presents a significant challenge for mobile robots operating in unfamiliar environments.

1.2.2 Available Information

When establishing its position, a robot has access to two distinct sources of information. Initially, it possesses a priori data, which can be collected by the robot or provided by an external source during the initialization phase. Furthermore, the robot continuously acquires navigation information about its environment through each observation and action it undertakes while navigating its environment.

A priori Information

Typically, the a priori information furnished to the robot defines the environment in which the robot operates, highlighting specific attributes that remain constant over time and can aid in localization. A priori information comes in various forms, including maps and cause-effect relationships [6], providing essential context for determining the robot's location. Maps could be topological maps like special features or geometric maps like pathways and grids. These could be initialized or learned as the robot explores its environment, a technique commonly called *Simultaneous Localization and Mapping (SLAM)*. An example of cause-effect knowledge could be high noise levels in a factory, which may be associated with regions with heavy machinery operating in them. If there are multiple locations with similar visual cues, this associative knowledge can help the robot distinguish between a working machine and a broken machine.

Navigational Information

In navigation, a robot typically engages in two distinct categories of actions: physical movement within the environment, or *driving*, and the sensory perception of the sur-

roundings, or *sensing*. These two action types yield two disparate forms of positional data.

A robotic vehicle possesses a locomotion system to move around in an environment. A guidance system can consist of wheels, tracks, or legs; these components are termed actuators. By employing sensors to monitor the real-time actions of the robot's driving system, it becomes possible to estimate the displacement of the robot vehicle. This estimation yields what are known as relative position measurements. These involve observing the robot exclusively, without relying on external data, thereby providing information relevant only to the reference point from which the measurements commenced. Reconciling such a self-contained system with the real outside world is a central challenge.

The robot utilizes its sensors to perceive or sense the environment, generating instantaneous situational data called observations or measurements. This information characterizes the state of the robot's surroundings at a specific instance. These observations, originating from the environment, offer insights into the robot's location that are not contingent on prior location estimations, thus constituting absolute position measurements.

Techniques to obtain the relative and absolute position measurements are investigated in Section 1.2.3.

1.2.3 Localization Techniques

There is no one-size-fits-all solution for the mobile robot positioning problem. Instead, one can use different partial solutions depending on the real-life application and constraints. There are several widely researched and implemented techniques for localization based on the type of sensors or hardware solutions. These can be divided into solutions that provide relative position measurements or those that yield absolute position measurements. Such techniques are briefly mentioned and explained in the following subsections.

Relative Position Measurements

The cumulative integration of a robot's motion information or *odometry* is widely used for pose tracking or *relative position measurements*. However, it leads to the unbounded accumulation of errors. In *inertial navigation*, accelerometers and gyroscopes provide linear acceleration and angular velocity measurements, respectively. These measurements are suitably integrated to obtain the position and orientation. However, similar to odometry, the small errors accumulate, causing a boundless increase in error.

Absolute Position Measurements

Magnetic compasses are sensors that provide accurate heading information. Vehicle heading is the most significant of the pose parameters in the manner it influences the accumulated dead reckoning errors [7]. Robot localization can also be achieved using satellite navigation systems such as the *Global Positioning System (GPS)*, Galileo, and GLONASS. An appropriate combination of usually two of the above-mentioned techniques is used for every mobile robot system [7]. *Active beacons* are commonly used by ships and airplanes. Beacons require accurate mounting for accurate positioning estimation. Stars are also considered to be active beacons for maritime applications. Robots can recognize distinct features or landmarks using sensory inputs with a technique called *landmark matching*. These include geometric shapes and may additionally include supplementary information, like bar codes or QR codes. The positions of the landmarks are fixed and known. Hence, the robot can localize itself relative to the known landmark positions. These landmarks could be naturally occurring or artificially placed. Finally, in *map matching*, the robot uses onboard sensors to create a local map of its environment. Also present in memory is a global map of the previously stored environment. The robot compares the local and global maps to find a match. The robot will have its current position and orientation if there is a positive match. All of these methods provide *absolute position measurements* for the robot.

1.2.4 Difficulties, Errors and Uncertainties

The navigation challenge presents various intricate hurdles. Several factors contribute to the complexity of navigation, including

- constraints on computational resources,
- challenges in object detection and recognition,
- obstacle avoidance,
- and the intricacies of effectively utilizing environmental information.

For a robot to comprehend its surroundings, it relies on sensors, much like living organisms. However, the robot's perception of information may not always align with reality. Faults in object identification, the dynamic nature of environments, and sensor data introduce complexities that hinder a precise understanding of the environment.

For example, a robot often uses odometry to estimate its position. However, due to factors such as wheel slippage or other minor noise sources, inaccuracies are introduced

into the odometer readings [8]. Additionally, since odometer readings rely solely on the count of wheel revolutions, the robot cannot assess their accuracy based on these readings alone. Consequently, the margin of error in the robot's perceived location is prone to escalate. The robot may employ visual sensors to maintain the location error within manageable limits. These sensors perceive the environment, and the data is used to scrutinize it. Resources such as vision, compasses, active beacons, landmarks, or potentially GPS systems reinforce its confidence in its current location [7].

1.2.5 Multi-Sensor Fusion

Algorithms designed to address the localization problem combine initial data with both relative and absolute position measurements, forming estimates of the robot's location at a specific point in time. When these measurements are read from diverse sensors, the challenge shifts to amalgamating these readings to create an optimized, unified representation of the environment. This dilemma is explored in research focused on multi-sensor fusion [9]. This technique enables the combination of diverse relative and absolute sensory input measurements to create a more comprehensive understanding of the robot's environment.

By combining the strengths of different sensors, multi-sensor fusion compensates for the limitations or deficiencies of individual sensors, providing a more detailed and accurate representation of the surroundings. This is particularly crucial when each sensor does not possess identical sensing capabilities. Fusion not only improves the perception of the environment but also aids in diminishing errors and uncertainties, fostering robust decision-making and localization capabilities for robots operating in complex and dynamic settings. Additionally, multi-sensor fusion is imperative as it can mitigate the impact of measurement errors.

1.3 Thesis Organization

In this chapter, localization is introduced. Its requirements and how it is achieved are discussed. The topic of multi-sensor fusion is briefly examined. Establishing this concept is integral in the venture to optimize lunar rovers with only the most basic hardware.

Chapter 2 presents the main research problem and associated research questions to addressed in this thesis. A literature review of the localization technologies that are in use today is conducted. Different studies into terrestrial and planetary localization technology are also presented. This provides a good foundation to understand what

can be considered as solutions to localize the lunar rover and also serves as a basis to identify literature gaps and novel contributions in this thesis. The assumptions and constraints of minimally equipped lunar rovers are described.

In Chapter 3, background knowledge about coordinate frames and rotations is introduced. This is essential for multi-sensor systems, probabilistic localization, and robot pose estimation. In addition, solar position calculations are detailed, which are necessary to use the data from the sun sensor.

In Chapter 4, the system model is developed. Methods used to obtain information about the position and orientation of the robot from the onboard sensors are presented. In addition, some methods to combine information from different sensors is discussed with reference to lunar rovers, specifically the Lunar Zebro.

Finally, Chapter 5 concludes the thesis with discussions and summaries of the results and some suggestions for future work.

Robot Localization Methods and Algorithms: A Review

2

In the previous chapter, an overview of general mobile robot localization technologies was presented (see, § 1.2.3, page 5). This chapter explores and analyzes a few solutions available in the literature. State-of-the-art solutions are included for generality. However, this thesis focuses on solutions viable for extraterrestrial (lunar and planetary) environments where GPS is not available.

2.1 State-of-the-art Localization Technology

Remarkable advancements in precision, reliability, and versatility mark the state of the art in robot localization technology. Modern robot localization systems integrate various sensors, including high-definition Light Detection and Ranging (LIDAR) sensors, cameras, and Inertial Measurement Unit (IMU), enabling robots to navigate complex and dynamic environments with exceptional accuracy.

Simultaneous Localization and Mapping (SLAM) algorithms have also seen significant improvements [10, 11]. These algorithms enable robots to build detailed maps of their surroundings and simultaneously localize themselves within these maps, even in previously unexplored terrain. Machine learning and deep neural networks have been instrumental in enhancing the performance of these algorithms, allowing robots to recognize and understand a wide range of environmental features and obstacles [12]–[14].

In addition, integrating cloud-based data and real-time communication further enhances the capabilities of robot localization [15, 16]. Robots can now access and share environmental data with other connected devices, making them more adaptive and efficient in collaborative tasks and contributing to developing autonomous and intelligent systems.

An extensive review of SLAM based on the omni-directional camera is presented in [17]. The authors in [18] present a state-of-the-art analysis of the current robot localization methods based on the passive UHF-RFID technology. Finally, the surge in artificial intelligence within the technological landscape has prompted the adoption of AI methodologies in robot localization. In [19], a review of relevant works in mobile robotics using AI techniques and visual information is presented. The review concentrates mostly on deep-learning tools for mobile robotics.

The authors in [5] present a comprehensive review of the analysis of existing localization technologies. Based on the sensors used, cameras, sonars, and laser sensors are the most prominent approaches. Other methods make use of IMU, odometry, and compass, supplemented with a GPS. The prominent probabilistic techniques include the extended Kalman filter-based localization and vision-based tracking. The authors also explore evolutionary approaches as well as SLAM-based localization.

2.2 Terrestrial GPS-Denied Localization Technologies

Robot localization in terrestrial environments is a dynamic and evolving field of study. While GPS technology is widely used for outdoor navigation, it may face limitations in urban canyons, dense forests, or areas with signal obstructions. Robots often rely on sensor fusion techniques to overcome challenges in such environments wherein data from multiple sensors like LIDAR, cameras, and IMUs are integrated to improve accuracy.

In addition to traditional approaches, machine learning and artificial intelligence-based techniques are increasingly being employed to enhance localization precision. Deep learning models and computer vision techniques help robots recognize and interpret environmental features, making them more adept at understanding their surroundings.

The importance of accurate terrestrial robot localization extends beyond traditional sectors like transportation and agriculture. It has applications in search and rescue missions, environmental monitoring, infrastructure inspection, and more.

2.2.1 Indoor Environment Localization

It is widely observed that despite very high accuracy and wide coverage, GNSS signals are not suitable for localization in indoor environments. With intricate geometries and dynamic scenarios, indoor environments present more complex constraints. The suggested solutions for indoor applications are highly coupled to the environment and target application [20]. Currently, indoor positioning widely makes use of Wi-Fi, Bluetooth, ZigBee, Radio Frequency Identification (RFID), Ultra-Wide Band (UWB), Inertial Measurement Unit (IMU), Visible Light Communication (VLC), Infrared (IR), Ultrasonic, Geomagnetic, Light Detection and Ranging (LiDAR), and Computer Vision technology. The authors in [20] summarize different modern-day positioning technologies based on different classifications. These different classifications are shown in Table 2.1.

Six categories of technologies based on the type of signal used to measure the position

Gu et al.[21] *Type of sensor signal*	Mautz [22] *Similar Performance Groups*	Basiri et al.[23] *Most Common*	Mendoza-Silva et al.[24] *Type of Sensor*
•Infrared •Vision-based •Magnetic •Audible Sound •Ultrasound •Radio Frequency	•Infrared •Camera •Magnetic Localization •Sound •INS •UWB •WLAN/WiFi •RFID •Tactile and Combined Polar Systems •High sensitive GNSS/ Assisted GNSS •Pseudolite Infrastructure systems •Other RF (Cellular Networks, Zigbee, Radar, DECTPhones, Digital TV, FM radio)	•Infrared Market or reflective element •Infrared Light image feature matching •Light image feature matching •Light image market •Magnetometer •Sound •UWB ToF •WiFi RSS •WiFi ToF/AoA •RFID active •Bluetooth RSS •Tactile on user device •Tactile Environment •Tactile Odometer •Pseudolite •GNSS •Electromagnetic Systems •Barometer •Mobile Network	•Light •Computer Vision •Magnetic Field •Sound •Dead Reckoning •UWB •WiFi •RFID and NFC •Tactile Odometer •BLE •Other Technologies (Cellular Network, ZigBee, 5G)

Table 2.1: Different types of groupings of state of the art positioning techniques, reproduced from [20].

in wireless personal networks are presented in [21], the authors present six categories of technologies based on the type of signal used to measure the position in wireless personal networks. In [22], the author classified them into 13 groups with similar performance for comparison. The authors in [23] survey the 20 most suitable positioning technologies for location-based service applications. The most common sensor types are categorized and compared in [24].

Six categories of technologies based on the type of signal used to measure the position in wireless personal networks are presented in [21], the authors present six categories of technologies based on the type of signal used to measure the position in wireless personal networks. In [22], the author classified them into 13 groups with similar performance for comparison. The authors in [23] survey the 20 most suitable positioning technologies for location-based service applications. The most common sensor types are categorized and compared in [24].

The technique, algorithm, or strategy used heavily depends on the hardware or technology employed. The most common techniques used collaboratively include using the Received Signal Strength Indicator (RSSI) and Time of Arrival/Flight (ToA/ToF) of signals. The predominantly used algorithms included Particle Filters, Belief Propagation, Extended Kalman Filters, and Geometric Algorithms. WiFi is the most significantly used technology, especially in combination with different techniques and methods/algorithms. The next most popular technologies include using UWB and Bluetooth.

In [25], the authors describe that the twelve positioning methods divided into Non-Radio Frequency (IMU, VLC, IR, Ultrasonic, Geomagnetic, LiDAR, and Computer Vision) and Radio Frequency (WiFi, Bluetooth, ZigBee, RFID, and UWB) methods. An analysis of 147 papers surveyed in this review illustrates that the most used technology in mobile robot technology is SLAM, with 68 papers using the technology. LiDAR and cameras are used to implement this. The authors also found that researchers are most interested in computer vision. Finally, they conclude that data fusion is the key direction for researchers to innovate, combining IMU and other technologies as the best solution. The predominant fusion algorithms that are used include the EKF, particle filter, and neural networks.

2.2.2 Outdoor Localization

Localization technology and algorithms that are in place today, as state-of-the-art technology, and those that are used in indoor environments, can also be used in outdoor environments with some modifications and with more effort towards robustness due to the dynamic nature of the environment and other environmental factors. The authors in [26] propose a method to provide accurate localization estimates of robots in greenhouses. These robots cannot rely on GPS measurements due to unstructured, dynamic, and GPS-denied environmental conditions. Instead, they employ multi-sensor fusion, including measurements from wheel odometry, an inertial measurement unit (IMU), and a tightly coupled visual-inertial odometry (VIO) that are integrated using an Extended Kalman Filter (EKF) in order to provide a more accurate pose estimate.

In [27], the authors propose a method for accurate dead-reckoning of wheeled vehicles only using an IMU. The key components of this method include using a Kalman filter and deep neural networks to dynamically adapt the noise parameters of the filter. The results achieved provide, on average, a 1.10 % translational error and thus, the algorithm competes with top-ranked methods which, by contrast, use LiDAR or stereo vision. The authors in [28] present an inertial-aided localization approach by fusing information from multiple IMUs. While a single-IMU-based localization yields

acceptable accuracy and robustness for different use cases, the overall performance can be further improved by using multiple IMUs. This research highlights the importance of environment adaptive fusion techniques and the use of multiple sensors to aid in robustness.

The research conducted in [29] argues that using a visual sun compass as a navigational aid to continuously determine absolute bearing considerably improves the localization and navigation ability of planetary robots. This work further supports this claim by presenting experimental results using a high rate 187° field of view (FOV) visual sun compass.

2.3 Extraterrestrial Localization Technologies

Localization in extraterrestrial environments poses unique hurdles due to the absence of familiar landmarks and the unavailability of traditional positioning systems like GPS. Robotic systems deployed on celestial bodies such as the Moon or Mars must rely on alternative methods, including sensor fusion and terrain mapping, to determine their precise locations.

2.3.1 The Lunar Environment

Localization of a rover in a lunar environment presents more constraints. The number of technologies that can be used is highly limited, with increasingly many constraints. For many missions, micro-rovers produced by companies, and especially those produced by student teams, have limited resources and infrastructure that can be deployed to aid localization. Additionally, the challenges posed by the lunar environment itself include:

- Lack of magnetic field
- Craters
- Volcanic features/lava tubes
- Areas of high and low (shadow) illumination
- No localization infrastructure

2.3.2 Localization in Extraterrestrial Environments

Many rover localization systems operate predominantly with the help of visual odometry and cameras. These are typically used for more accurate localization with more

contemporary methods. In [30], the authors present a method whereby they use craters as landmarks for localization. They achieve this using 3D point cloud data from LiDAR and stereo images for crater detection and shading cues from onboard monocular imagery.

Onboard localization systems usually require an absolute position update from human operators on the ground, "ground-in-the-loop" (GITL) systems. The authors in [31] build upon their crater detection algorithm to present LunarNav. Here, they focus on crater matching and state estimation aspects of this localization using the craters problem. Their proposed method permits higher autonomy, thereby eliminating the need for GITL cycles, contributing to faster, more efficient systems. The authors duly note that the sensors employed are not fully developed at present. Furthermore, they assumed that the computational load of the object detection system of rovers would be higher, and thus, their algorithm could be used efficiently for Lunar missions. However, this may not always be the case.

Stereophotoclinometry and the use of panoramic images for robot localization are investigated in [32]. The proposed technique can be used for both initial position determination and continuous movement tracking. While high-quality resolution imagery is needed for panoramic navigation, the resulting location estimates were within 6 m of the truth.

Visual odometry can rationally be used to determine the amount of slippage in high-slip environments [33]. The technique could consistently estimate rover motion within 2.5% of the distance traveled. The estimate formed by merging data from visual odometry and the inertial measurement unit (IMU) is compared to the kinematic data using a Kalman filter to determine the amount of slippage suffered. Necessary steering angles and velocities to compensate for the slip are then calculated and accounted for, allowing the rover to traverse through high-slip environments.

Celestial trackers like star and sun sensors are widely researched and frequented navigational tools. They can provide absolute heading information, which profitably reduces the error growth in vision-based navigation [34]. In this work, the authors explain in detail the formulation and working of star trackers. Using a star tracker and an inclinometer, the authors in [35] present an accurate celestial navigation method for localization. The work can manipulate multiple observations but focuses on using a single-star observation. Nevertheless, the accuracy of the estimates is shown to be relatively high. The accuracy of the star tracker limits the veracity of the estimates. The authors suggest exploring alternative methods of measuring tilt (roll and pitch angles) with higher precision and resolution.

The work in [36] provides an algorithm that employs star trackers to correct odometric estimates for drift and/or bias. Corrections are significant even during full-dark rover operations. The algorithm was tested on a real rover with a star tracker and an inclinometer. The results proved highly accurate, with an error rate of only 0.85 % of the total distance traveled when all the attitude measurements were used. The demonstrated results emphasize the possibility of simpler sensors rivaling visual odometry. However, simpler sensors like wheel encoders have disadvantages, such as wheel slip and 2D Pose estimation.

An adaptive celestial positioning algorithm is introduced in [37]. The method uses a self-calibration model for the star sensor. The positioning results indicated an obvious improvement in accuracy using the proposed algorithm compared to no calibration. Notably, the authors conclude that self-calibration should also improve the accuracy of azimuth determination when using the sun.

2.3.3 Localization in Past Rover Missions

In this section, the localization techniques are briefly discussed for a few previous rover missions.

2.3.3.1 Spirit and Opportunity

The rover carried a BAE Systems 20-MHz 32-bit RAD6000 CPU (a radiation-hardened version of the PowerPC), and on-board memory includes 128 MB RAM, 256 MB flash, and smaller amounts of other non-volatile memory [38]. The position estimation system mainly leverages a stereo navigational camera pair (Navcam). Spirit (and Opportunity) localization has been conducted through two distinct methods: one utilizes an incremental bundle adjustment (IBA) with rover imagery, while the other compares shared image features between a rover orthoimage and an orbital orthoimage [39]. The Spirit and Opportunity rovers utilized IMU (Inertial Measurement Unit) and wheel encoders for localization. The rovers utilize accelerometers and sun imagery to obtain their orientation, autonomously tracking the sun by scanning the sky with a movable camera. For updating their attitude and position, the rovers rely on either accelerometer and gyroscope data or gyroscope data and wheel odometry, adapting based on the commanded movements from operators on Earth [40].

2.3.3.2 Curiosity

The computational power available onboard included two redundant main computers (one operational and one spare), each powered by a BAE Systems RAD 750. Furthermore, each computer possesses 2 GB of flash memory, 256 MB of DRAM, and 256 KB of EEPROM [41]. The rover has two pairs of black and white navigation cameras mounted on the mast to support ground navigation. The rover also had the same IMU as the MERs (Spirit and Opportunity) rovers and its localization algorithms are very similar to those on the MERs mission. Additionally, a sequence of HiRISE images are taken when the Mars Reconnaissance Orbiter (MRO) flies over the landing site region, that can be used to identify rover tracks and determine the rover locations at the times of HiRISE imaging. Both MER and MSL rovers are equipped with an IMU that provides attitude information of the rovers, while wheel encoders allow for calculation of travelled distance. Finally, the cameras may also observe the Sun to obtain improved azimuth information [42].

2.3.3.3 Yutu-1

The Yutu rover utilized a stereo camera pair positioned approximately 1.5 meters above the lunar surface. This camera assembly could be adjusted in both yaw and pitch directions, enabling the rover to capture various regions from a single location. Similar to the Spirit and Opportunity rovers, Yutu primarily relied on an inertial measurement unit (IMU) and wheel encoder data for its baseline localization system. However, this system was further enhanced by incorporating a visual odometry algorithm, akin to the methodology employed by Spirit and Opportunity. The visual odometry algorithm utilized sparse feature correspondences extracted from multiple images. Initially, the algorithm searched for feature correspondences across successive stereo image pairs, with search regions initialized by the baseline position estimator. Detection and matching of these feature points were performed using the ASIFT method. In ground test scenarios, the baseline localization system achieved an error of about 14%. The visual odometry system reduced the errors to about 5% for a 43 m path [43] . Information on the onboard computational resources is currently unavailable.

Comment: In the literature and available information on past rovers, such as the Rocky 7 [44], the use of the sun sensor for pose estimation is often mentioned. However, the algorithms are rarely detailed. IMUs and wheel encoders are also present onboard every rover surveyed. Thus, it is well-assumed that most rovers make use of the IMU, the wheel encoder and the sun sensor. Furthermore, these rovers surely

have the capability to process information from these sensors, given that other rovers also make use of more computationally intensive processes such as SLAM and visual odometry.

2.4 Problem Formulation

As stated in the previous section, the moon's conditions must be considered to localize a lunar rover. Individual methodologies may not be sufficiently effective for application on the moon or other celestial bodies. For example, GPS systems are impractical due to the requirement of a minimum of four satellites, a condition unmet in such environments. Moreover, detailed maps are often unavailable beforehand, rendering map-based localization unfeasible. Landmark-based localization necessitates the deployment of numerous landmarks on the lunar or planetary surfaces. Magnetic compasses are ineffectual due to the lack of a magnetic field. Additionally, dead reckoning lacks accuracy, especially over extended robot movement ranges, primarily due to the accumulation of integration errors. Such restrictions are to be taken into account while formulating a localization solution.

2.4.1 The Lunar Zebro

In addition to the environment, one must also consider the capabilities of the specific rover and its constraints. The Lunar Zebro is being developed as the world's smallest and lightest rover yet. With these specialities, however, the computing power available onboard is limited. Further, as a choice, the localization must not use the single onboard camera as it is exclusively used to take pictures of the lunar surface alongside executing object detection. Supplementary sensors such as LiDAR increases the weight and costs of the operation. Therefore, an additional constraint is to avoid such sensors. Additionally, computational constraints also limit the use of machine learning, deep learning and computer vision technologies, and by extension, the most popular localization technique, SLAM, is also not considered.

The Lunar Zebro is designed to have an onboard Inertial Measurement Unit and Hall-Effect sensors (i.e., wheel encoders) on the wheels. Additionally, a sun sensor aids in orienting the solar panels towards the Sun for more efficient battery charging. Inertial navigation and odometry from wheel information are relative positioning techniques. Navigation using the Sun is a type of localization using active beacons, a global positioning technique. Thus, by combining information from these sensors, one can estimate the pose of the lunar rover.

An additional property to note is the rover's special locomotion system. It has specialised C-type wheels that can climb rocks similar in size to the rover. The Lunar Zebro has a differential drive wheeled (see § 4.4, page 38) configuration and has very slow speeds in the order of a few centimetres per second. Furthermore, the presupposition that the rover will alternate between mobility, and recharging for extended periods of time, is assumed for the rest of this thesis. Therefore, in its lifetime of a single lunar day, or a fortnight in Earth timings, the rover may only travel a few hundred metres.

2.4.2 Intended Contributions

Multiple works in the literature have discussed using Sun sensors or star trackers to aid in heading angle determination. For example, a lunar global/local localization method using Sun and earth vectors is described in [45]. Their simulations show that the rover's position is localized to within 0.5 km with the Sun and Earth vectors. The authors develop a method of linking consecutive global localization readings using dead-reckoning. Using this method increases the accuracy of the estimation to within 150 m. However, the authors do not report any results of hardware experiments. Field testing of the relative localization system on the Rocky 7 field rover is dealt with in [44]. The Sun sensor heading determination scheme limits the cross-track error to linear growth as the rover travels.

The authors in [46] present accurate results of using a Sun sensor to determine heading. They also use position and gravity information to determine precise heading angles. However, they do not use odometry and inertial measurement units. To estimate position they make use of the sun sensor and inclinometer to estimate position coordinate. However, the resulting positioning errors obtained are in the range of 200km even though their heading angles are quite accurate. As a future work point, they intend to use visual odometry to localize the rover better. The authors in [47] present an algorithm to determine the attitude of a rover using a Sun sensor. This information is fused with an IMU in an Extended Kalman Filter framework to obtain a more accurate heading estimate. While the heading angle result was highly accurate, the position estimate that resulted from their algorithm was incorrect in the order of kilometres. As a future research suggestion, the authors also suggest using wheel encoders to determine accurate position estimates. Finally, [29] used a 187° field-of-view camera-based sun sensor to correct IMU and wheel odometry-based pose predictions. This study is the first of its kind. Their results are highly accurate for a 302m travelled distance with an error of less than 5%.

Thus, finding ways to properly combine the information from three commonly

present onboard sensors (i.e., IMU, wheel-encoder and Sun sensor) for lightweight micro-rovers is a critical problem. There appears to be a gap in the existing literature that treats this problem. Accordingly, in this work, a study of some fusion methods that can combine information from multiple sensors is carried out and compared. Specifically, the fused information is used to localize the rover. The efficacies of the considered fusion methods are compared with respect to localization accuracy.

2.4.3 Research Questions

The crucial problem of lunar rover localization problem is investigated in this thesis. The aim is to determine the most accurate position using information from the sensors in a typical lightweight, low-complexity micro-rover, such as the Lunar Zebro. The IMU, wheel encoder, and Sun sensor are considered in this thesis, and the use of other specialized imaging and localization sensors is avoided. The following research questions are addressed here:

- How can one combine information from the IMU, wheel encoder and sun sensor to estimate the rover's pose?
- How do different combinations of sensor fusion and estimation techniques compare in accurately estimating the position and orientation of a Lunar rover using the aforementioned three sensors?
- What is the achievable localization accuracy for such rovers?

3 Preliminaries

This chapter covers different coordinate frames of reference and the transformations between these. These concepts are crucial for multi-sensor systems to combine information from various sensors. In addition, basic concepts in robot pose and motion are presented in this chapter. Finally, the required calculations for determining solar positions at any given time from solar ephemeris data are also discussed. As will be seen later in § 4.5.1 (page 47), this is made use of to derive information about the rover's orientation from the sun sensor.

3.1 Coordinate Systems and Frames of Reference

Multi-sensor systems are ever-prevalent today. Any random assortment of sensors could sense different quantities to meet various objectives. An essential aspect of multi-sensor systems is how one associates these assorted sensing platforms. Different sensors record data in their respective coordinate frames. For example, a rover's heading and motion are best understood relative to the local topocentric frame. This frame is defined with respect to the local horizontal. The reference frames relevant to this thesis are illustrated in Fig. 3.1, and the parameters are tabulated in Table 3.1.

Frame	Notation	x-axis	y-axis	z-axis
Earth Centred Initial (ECI)	I	Vernal Equinox		North Pole
Earth Centred Fixed (ECF)	F	Prime Meridian		North Pole
Topocentric	T	Local East	(Local North)	Opposite Gravity
Sun Sensor	S		Alignment Pins	Outward Normal

Table 3.1: The relevant frames of reference [48].

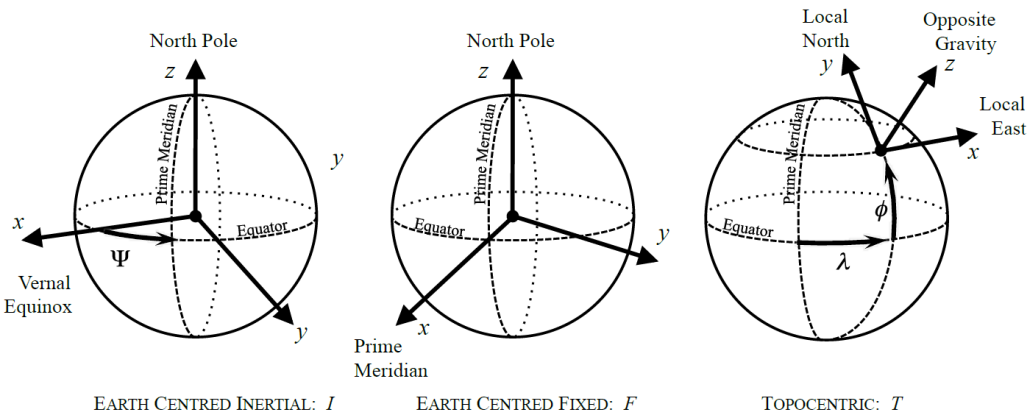


Figure 3.1: Illustration of the different coordinate frames [48].

3.2 Coordinate Rotations and Transformations

In this thesis, all vector transformations are described as rotations, captured in transformation matrices or rotation matrices. The rotation from b to frame a is denoted $\mathbf{C}_{ab} \in \mathbb{R}^{3\times3}$. Hence, the transformation of a 3D vector in frame b, $x_b \in \mathbb{R}^3$, to its representation in frame a, $x_a \in \mathbb{R}^3$, is given by

$$x_a = \mathbf{C}_{ab} x_b \tag{3.1}$$

It is most convenient to express frame transformations as a series of principal axis rotations. To this end, one defines three standard rotations as follows:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\theta & -s_\theta \\ 0 & s_\theta & c_\theta \end{bmatrix}, \quad \mathbf{R}_y(\theta) = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix}, \quad \mathbf{R}_z(\theta) = \begin{bmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.2}$$

Here, $c_\theta \equiv \cos(\theta)$ and $s_\theta \equiv \sin(\theta)$ [46]. These matrices represent right-handed rotations by an angle of θ about the x-, y-, and$z$-direction, respectively. Any rotation can be described by a series of rotations around these principal axes.

Some of the relevant coordinate reference frames are illustrated in Fig. 3.1. These come into play when sun-sensor data is used to estimate the rover's heading. From the figure, the transformation from ECF to ECI is a simple rotation:

$$\mathbf{C}_{IF} = \mathbf{R}_z(\Psi), \tag{3.3}$$

where Ψ is termed the *Greenwich Apparent Sidereal Time (GAST)*. This is used and described in § 4.5.1 (page 47).

3.3 Robot Pose and Motion

Kinematics is the calculus describing the effect of control actions on the configuration of a robot [49]. Six variables describe the configuration of a rigid mobile robot: its three-dimensional Cartesian coordinates and its three Euler Angles (roll, pitch, yaw) relative to the external coordinate frame. As mentioned in Chapter 1, the goal of this project is to estimate the robot's pose, the two-dimensional planar coordinates relative to an external coordinate frame, along with its angular orientation, at each time step. The vector $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$ represents this for each instance. The three aspects of robot pose are illustrated in Fig. 3.2.

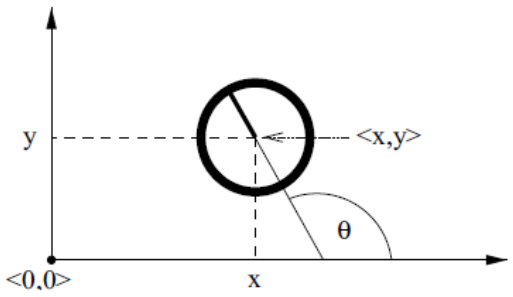


Figure 3.2: Robot pose, illustrated in a global coordinate system [49]

Given a moving robot, one can describe its motion as going from a state S_{t-1} to S_t due to the execution of a control action u_t at time step t. In this thesis, the state $S = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$ is the pose. Motion models, or the probabilistic kinematic model, comprise the state transition probability $p(S_t \mid u_t, S_{t-1})$.

3.3.1 Odometry Motion Model

The motion of a robot is modelled to calculate or estimate its motion over time. As introduced earlier in § 1.2.3 (page 5), odometry is commonly obtained by integrating wheel rotation information. This leads to the odometry motion model. This model uses odometry measurements instead of controls. It is important to note that odometry information is only available retrospectively or after the robot moves. This poses no problem for filtering algorithms; however, they are not useful for motion planning.

The standard odometry model assumes that the motion of a robot can be described by three deltas of motion [49]. The motion is approximated by a rotation δ_{rot1}, followed by a translation δ_{trans}, and finally a second rotation δ_{rot2}. This is illustrated in Fig. 3.3. The odometry information can be considered to be as analogous to the control

$u = (\delta_{rot1}, \delta_{trans}, \delta_{rot2})$. The pose computed at every step can also be considered as a measurement from the wheel encoder sensors.

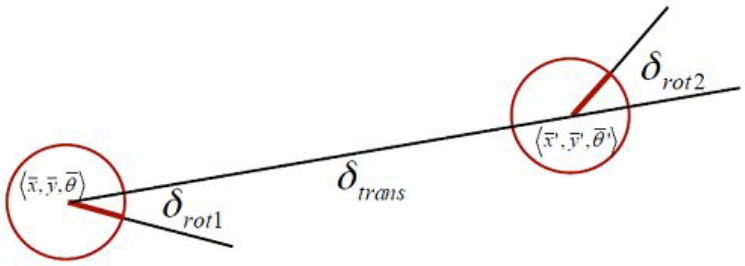


Figure 3.3: Illustration of the Standard Odometry Model [50]

In the time interval $(t-1, t]$, the robot advances from a pose S_{t-1} to pose S_t. The odometry reports a relative advance from $\bar{S}_{t-1} = \begin{pmatrix} \bar{x} & \bar{y} & \bar{\theta} \end{pmatrix}^T$ to $\bar{S}_t = \begin{pmatrix} \bar{x}' & \bar{y}' & \bar{\theta}' \end{pmatrix}^T$. The relative advances and change in motion can be calculated, via geometry, as follows. The robot's motion can be fully described with the pose information at different time steps:

$$\begin{aligned} \delta_{trans} &= \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2} \\ \delta_{rot1} &= \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} \\ \delta_{rot2} &= \bar{\theta}' - \bar{\theta} - \delta_{rot1} \end{aligned} \tag{3.4}$$

Here, atan2 represents the 2-argument *arctangent* function and is equivalent to finding the phase angle of the complex number $a + \sqrt{-1}b$ which accounts for the quadrant to which the complex number belongs.

3.3.2 Distributions for Probabilistic Motion Models

Typically, motion models are described in a probabilistic manner, usually either by a *normal* distribution model, which is described by (3.5) and illustrated in Fig. 3.4, or a *triangular* distribution model [49]. The standard normal distribution curve is defined as follows:

$$\varepsilon_{\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}} \tag{3.5}$$

In the normal distribution, there is a tiny probability that the robot's position is at an infinite distance since this curve never reaches zero. In contrast, the triangular distribution has well-defined bounds. It is defined by (3.6) and illustrated in Fig. 3.5.

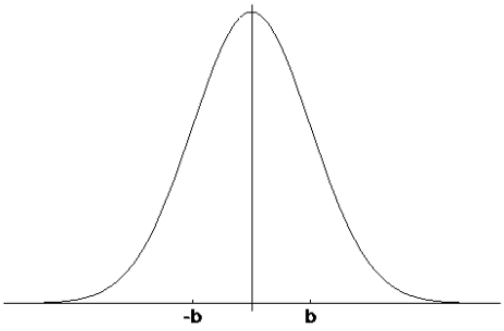


Figure 3.4: Normal Distribution Curve

There are clear restrictions on the probable robot's position.

$$\varepsilon_{\sigma^2}(x) \begin{cases} 0 & |x| > \sqrt{6\sigma^2} \\ \frac{\sqrt{6\sigma^2}-|x|}{\sqrt{6\sigma^2}} & \text{otherwise} \end{cases} \tag{3.6}$$

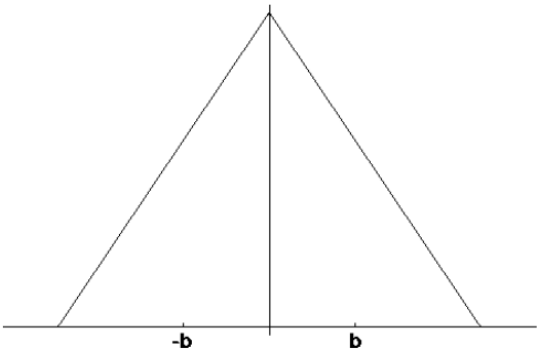


Figure 3.5: Triangular Distribution Curve

3.4 Solar Ephemeris Predictions

As stated earlier in § 2.4 (page 17), sun sensor measurements are used to estimate the rover's position on the moon.

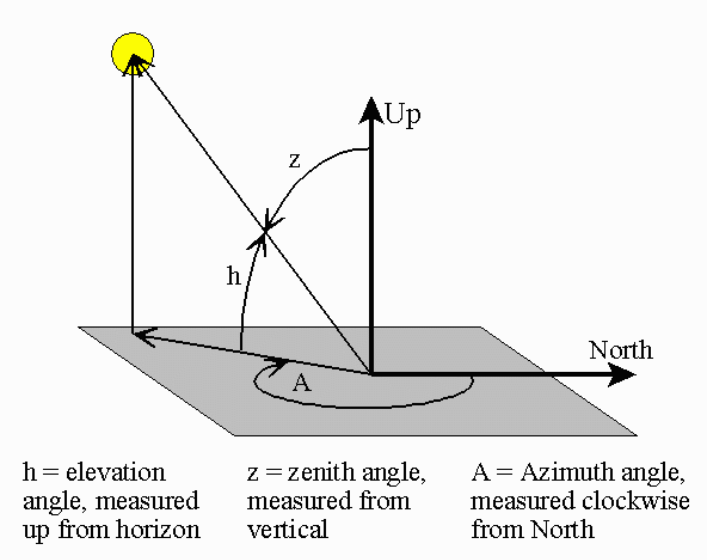


Figure 3.6: Components of Solar Position [51]

However, to accurately do so, the sun's position with reference to the planet is to be predicted. Specifically, the azimuth and elevation of the sun with respect to the observer. These are referred to as the solar ephemeris calculations and are illustrated in Fig. 3.6. The following calculations make use of Meeus' solutions for approximations of the solar position [52]. This section presents various astronomical concepts, the explanations for which are beyond the scope of this thesis. Only those quantities relevant to making the computations of the solar position are described. The computational path is shown in Fig. 3.7. In what follows, each block depicted here is detailed.

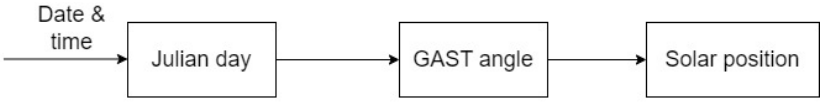


Figure 3.7: Procedural Flow to Calculate Solar Position Predictions or Ephemeris

3.4.1 Julian Day

Given the date and time in the Gregorian calendar, one can calculate the corresponding Julian Day (JD) as follows: Let Y be the year, M the month number and D the day of the month of the given calendar date.

- If $M > 2$, leave Y and M unchanged.
- If $M = 1$ or 2, replace Y by $Y - 1$, and M by $M + 12$

- In the Gregorian calendar

$$\begin{aligned} A &= \text{INT}\left(\frac{Y}{100}\right) \\ B &= 2 - A + \text{INT}\left(\frac{A}{4}\right) \end{aligned} \tag{3.7}$$

- In the Julian calendar $B = 0$
- The required Julian Day is given by

$$JD = \text{INT}(365.25(Y + 4716)) + \text{INT}(30.6001(M + 1)) + D + B - 1524.5 \tag{3.8}$$

In the steps above, INT(x) represents the greatest integer less than or equal to x.

3.4.2 The Greenwich Apparent Sidereal Time (GAST) Angle

Given JD at any instant, there are a series of computations to determine the GAST angle. These are as follows:

$$T = \frac{JD - 2451545.0}{36525} \tag{3.9}$$

The mean sidereal time θ_0 is given by:

$$\begin{aligned} \theta_0 &= 280.46061837 + 360.98564736629(JD - 2451545.0) + \\ &\quad 0.000387933T^2 - T^3/38710000 \end{aligned} \tag{3.10}$$

A periodic variation in the inclination of the axis of a rotating object is called *nutation*. The nutation in longitude, denoted $\Delta\psi$, and the nutation in obliquity, denoted $\Delta\varepsilon$, of the ecliptic are calculated as follows:

$$\begin{aligned} \Omega &= 125.04452 - 1934.136261T + 0.0020708T^2 + T^3/450000 \\ L &= 280.4665^\circ + 36\,000.7698^\circ T \\ L' &= 218.3165^\circ + 481\,267.8813^\circ T \\ \Delta\psi &= -17.20''\sin\Omega - -1.32''\sin 2L - 0.23''\sin 2L' + 0.21''\sin 2\Omega \\ \Delta\varepsilon &= 9.20''\cos\Omega + 0.57''\cos 2L - 0.10''\cos 2L' - 0.09''\cos 2\Omega \end{aligned} \tag{3.11}$$

The *obliquity* of the ecliptic is given by

$$\varepsilon_0 = 23^\circ 26' 21.448'' - 46.8150''T - 0.000\,59''T^2 + 0.001\,813''T^3 \tag{3.12}$$

The true obliquity of the ecliptic is $\varepsilon = \varepsilon_0 + \Delta\varepsilon$. Finally, the apparent sidereal time or the Greenwich hour angle Ψ of the true vernal equinox is given by:

$$\Psi = \theta_0 + \Delta\psi\cos(\varepsilon) \tag{3.13}$$

3.4.3 Solar Position

The Sun's position in the Earth Centred Inertial (ECI) frame requires the computation of the quantities mean longitude of the Sun L_0, the mean anomaly of the Sun M, the eccentricity of the Earth's orbit e, and the Sun's equation of the center C:

$$\begin{aligned} L_0 &= 280.466\,46^\circ + 36\,000.769\,83^\circ T + 0.000\,303\,2^\circ T^2 \\ M &= 357.529\,11^\circ + 35\,999.050\,29^\circ T - 0.000\,153\,7^\circ T^2 \\ e &= 0.016708634 - 0.000042037T + 0.0000001267T^2 \end{aligned} \tag{3.14}$$

$$\begin{aligned} C = {} & (1.914\,602^\circ - 0.004\,817^\circ T + 0.000\,014^\circ T^2)\sin M \\ & + (0.019\,993^\circ - 0.000\,101^\circ T)\sin 2M \\ & + (0.000\,289^\circ)\sin 3M \end{aligned}$$

The Sun's true longitude $\odot$, and the true anomaly, ν are calculated using the relations,

$$\begin{aligned} \odot &= L_0 + C \\ \nu &= M + C \end{aligned} \tag{3.15}$$

The apparent longitude λ of the Sun is found as follows:

$$\begin{aligned} \Omega &= 125.04^\circ - 1934.136^\circ T \\ \lambda &= \odot - 0.005\,69^\circ - 0.004\,78^\circ \sin\Omega \end{aligned} \tag{3.16}$$

The Sun's right ascension α and declination δ are then calculated as,

$$\begin{aligned} \tan\alpha &= \frac{\cos\varepsilon \sin\lambda}{\cos\lambda} \\ \sin\delta &= \sin\varepsilon \sin\lambda \end{aligned} \tag{3.17}$$

Finally, the horizontal coordinates are calculated using the observer's latitude φ and the longitude L, using the Hour Angle H. These are the azimuth A and the altitude or elevation angle h.

$$\begin{aligned} H &= \theta_0 - L - \alpha \\ \tan A &= \frac{\sin H}{\cos H\ \sin\varphi - \tan\delta\ \cos\varphi} \\ \sin h &= \sin\varphi\ \sin\delta + \cos\varphi\ \cos\delta\ \cos H \end{aligned} \tag{3.18}$$

With these steps, the solar ephemeris is obtained in the ECI frame. This is useful for pose computations using the sun sensor, further detailed in Section 4.5.1 (page 47).

3.4.4 Testing and Results

In this section, the aforementioned algorithm, i.e. (3.7)–(3.18), to compute the solar position is tested and evaluated for accuracy. Specifically, the solar position for the day `12 July 2008`, from times `13:00:00` to `24:00:00` are computed. The positions are calculated from a point on the Earth with latitude 75.43100678° and longitude −89.8733219°. This particular day and geographical coordinates are especially relevant in this thesis. The sun sensor dataset used later in the sequel corresponds to this date and location (see, § 4.1, page 30). The calculated ephemeris or solar position predictions using (3.7)–(3.18) are compared with the true solar angle data of the azimuth and elevation obtained from an online solar position calculator [51]. The results are illustrated in Fig. 3.8. Evidently, the ephemeris calculations are quite accurate as the two have negligible visual differences. To further emphasize, the errors in the angles are shown in Fig. 3.9. The maximum error in the altitude angle over a 10-hour period is 0.115° and the maximum error in the azimuth angle is of the order of 0.0035°. To conclude, the sun's predicted (estimated) position is highly accurate compared to the true value obtained from the online solar calculator. These errors are orders of magnitudes smaller than those that arise when using sensor data for localization purposes. In the next chapter, the ephemeris predictions are juxtaposed with the sun sensor angles to ascertain the rover's orientation angle.

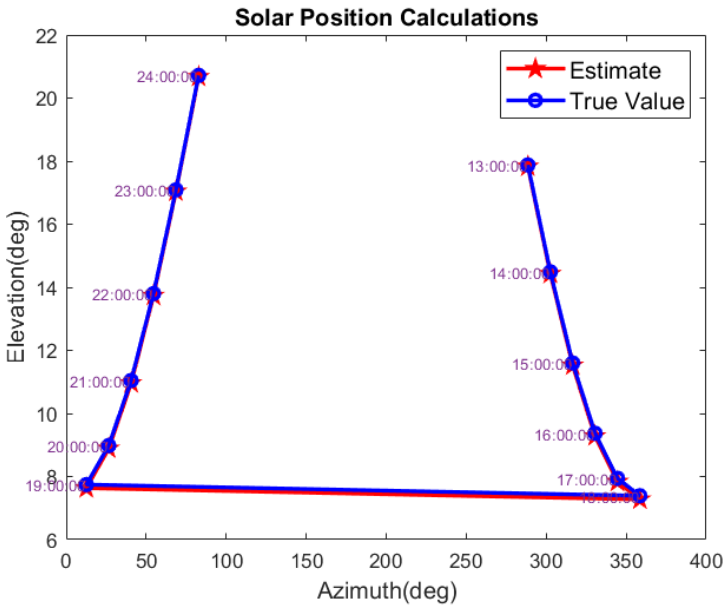
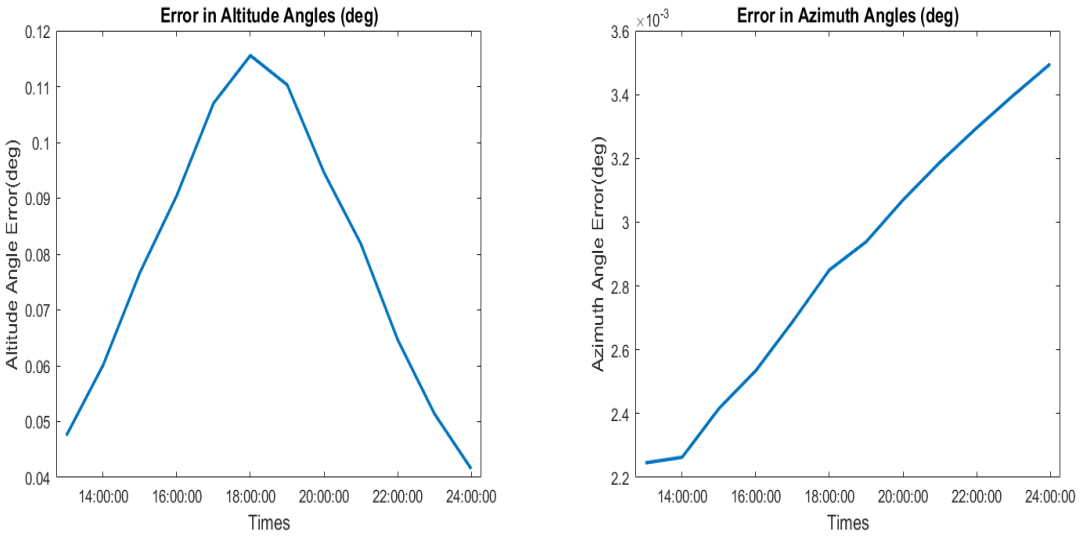


Figure 3.8: Solar Position Computation ("Estimates") Using the Algorithm from [52] in Comparison to the ("True") Positions from Online Calculators [51].



(A) (B)

Figure 3.9: Errors in Calculated Angles. (A): Altitude. (B): Azimuth.

Pose Estimation Using Sensors and Sensor Fusion

4

This chapter evaluates the methods to extract the pose information from the data obtained with an IMU, wheel encoder, and a sun sensor. MATLAB tools are used to simulate the path taken by the Devon Island Rover. Subsequently, sensor fusion techniques are assessed to procure more precise localization results from these sensors.

The odometry motion model was introduced earlier in § 3.3.1 (page 22). This model describes the motion of a robot over time. However, the computation of the robot's new position requires a control input u. Currently, the development of the Lunar Zebro takes effect in a decentralized manner, impeding direct integration of the locomotion and control module's output with the localization system. Accordingly, the exploration in this chapter delves into the realm of passive localization, which operates independently of the control input. In contrast, in active localization, the control input is pivotal in estimating the robot's position and orientation.

4.1 Simulation Setup

This section discusses the simulation setup to evaluate the methods of pose information extraction. This dataset is used to extract pose information using different sensors. Subsequently, the effect of fusing information from multiple sensors is also analyzed using this dataset.

4.1.1 Dataset

The Devon Island Rover Navigation Dataset [53] encompasses a ten-kilometre rover journey. Researchers collected many different types of data during the travel. They recorded the rover's journey on Devon Island between 10:00 and 18:00 hours. This experimental setup is the only one available that includes a sun sensor. Accordingly, this dataset is used in this thesis rather than other more commonly available simulated data. The dataset comprises GPS measurements used here as the ground truth. These measurements are used as waypoints of the rover. The GPS measurements, which used as waypoints, captured during the same interval are illustrated in Fig. 4.1.

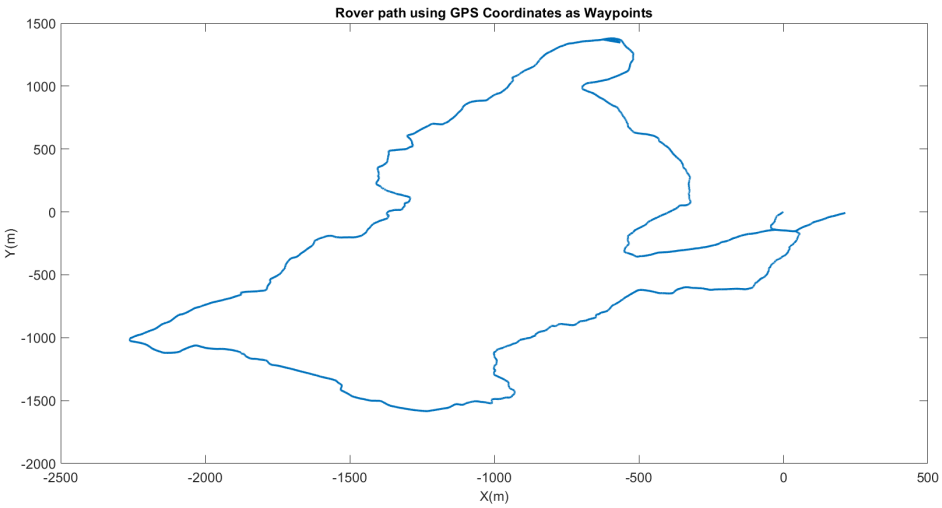


Figure 4.1: Rover path using GPS coordinates from [53] as waypoints.

It is important to note that the sun sensor dataset available in [53] consists of fewer data points than the GPS measurement data. The reason for this is that the sun sensor data is not recorded whenever the rover is stationary. Therefore, as a pre-processing step, the GPS measurements at timestamps when the sun sensor data was not recorded are ignored. Only the remaining information is used to appropriately combine with the sun sensor data. The format of the dataset used here is shown in Table 4.1. Using the GPS measurements as the ground truth for waypoints that the rover must follow, built-in libraries are used to simulate IMU and wheel encoder data. Accordingly, data from three sensors are available for use in this thesis. The trajectory was modeled using the `waypointTrajectory` function in the Sensor Fusion Toolbox of MATLAB.

The planar path of the `waypointTrajectory` trajectory (the x-y plane projection) consists of piecewise, clothoid curves. The curvature between two consecutive waypoints varies linearly with the curve length between them. The tangent direction of the path at each waypoint is chosen to minimize discontinuities in the curvature. Once the path is established, the object uses cubic Hermite interpolation to compute the location of the vehicle throughout the path as a function of time and the planar distance traveled [54].

Timestamp	Azimuth angle	Zenith angle	Latitude	Longitude

Table 4.1: Illustration of the format of the data from [53]. All angles are measured in degrees.

The Devon Island rover travels a distance in the range of kilometres in the directions

of x and y. In contrast, the Lunar Zebro travels only a distance of a few hundred meters in its entire lifetime. Consequently, three subsets from the Devon Island rover dataset are used in this thesis. These datasets, of much shorter duration, are listed below:

- *Dataset 1*: 14:18:25 to 14:36:03
- *Dataset 2*: 15:33:57 to 15:45:17
- *Dataset 3*: 14:24:58 to 14:46:47

These datasets are illustrated in Fig. 4.2. The number of datapoints with missing sun sensor data is minimal during these durations. Additionally, these subsets are in the middle of the overall data. These measurements are assumed stable in contrast to the start and end measurements.

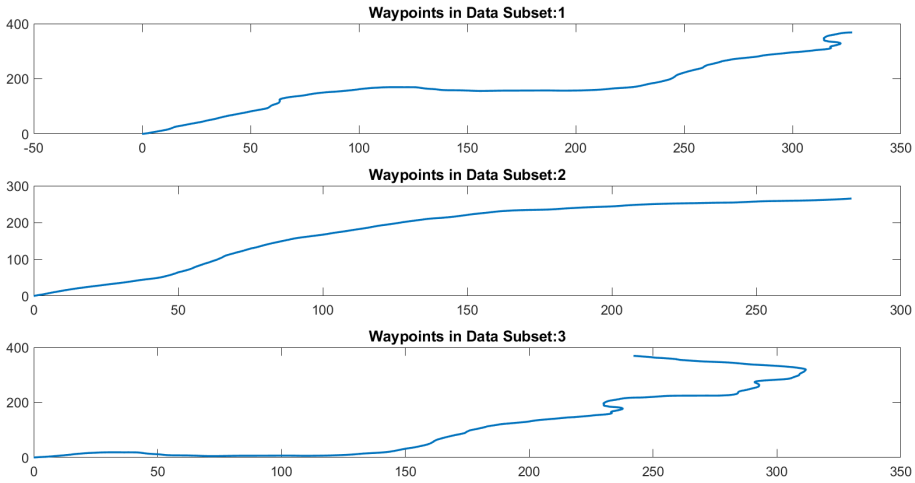


Figure 4.2: Subsets of Waypoints of Rover Path Used in This Thesis

4.2 State Space Model

The state space representation allows one to model the rover motion system comprehensively. The system's state is the smallest dimension vector that completely summarizes the system's past [55]. The general discrete-time linear state space model is represented by:

$$\begin{aligned} S(k+1) &= AS(k) + Bu(k) + w(k) \\ z(k) &= HS(k) + v(k) \end{aligned} \tag{4.1}$$

Here, A, B and H are the system, input and measurement matrices, respectively. Also, $S(k)$ represents the state at time step k, $u(k)$ is the control input at instant k, and $z(k)$

is the measurement at instant k. Finally, $w(k)$ represents the process noise and $v(k)$ is the measurement noise. It is assumed that $w(k)$ and $v(k)$ are independent of each other at every instant k, and independent of the initial condition $S(0)$.

4.2.1 Kinematic Model for Position Estimation

The kinematic model is derived from the first principles. Let the robot's position be denoted $s(t)$. The measurements from IMU provide acceleration data. An assumption of constant acceleration results in the time-derivative of the acceleration being zero, i.e., $\dddot{s} = 0$, leading to a third-order model. In practice, the acceleration is not exactly constant, and a continuous-time zero-mean white noise process can model its changes. Thus, one obtains a Wiener process model $\dddot{s}(t) = \tilde{v}(t)$ [55].

Suppose that the state vector is defined as follows:

$$S = \begin{bmatrix} s & \dot{s} & \ddot{s} \end{bmatrix}^T \tag{4.2}$$

where s, $\dot{s}$ and $\ddot{s}$ represent the distance, velocity and acceleration, respectively. The corresponding continuous-time state-space model is

$$\dot{S}(t) = AS(t) + D\tilde{v}(t), \tag{4.3}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{4.4}$$

In contrast, the standard one-dimensional equations of motion for constant acceleration are

$$\begin{aligned} s &= s_0 + v_0 t + \frac{1}{2} a t^2 \\ v &= v_0 + at \end{aligned} \tag{4.5}$$

4.2.2 Discretized State-Space Model

The discrete-time state-space model is obtained by discretizing the continuous-time state-space model. The resulting discrete-time state space model is given by

$$S(k+1) = AS(k) + v(k) \tag{4.6}$$

with the transition matrix defined as:

$$A = \begin{bmatrix} 1 & T & \frac{1}{2}T^2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.7}$$

where T is the sampling interval. In addition, the covariance matrix of $v(k)$ is given by:

$$Q = E[v(k)v(k)^T] = \begin{bmatrix} \frac{1}{20}T^5 & \frac{1}{8}T^4 & \frac{1}{6}T^3 \\ \frac{1}{8}T^4 & \frac{1}{6}T^3 & \frac{1}{2}T^2 \\ \frac{1}{6}T^3 & \frac{1}{2}T^2 & T \end{bmatrix} \tilde{q} \tag{4.8}$$

where $E[\cdot]$ represents the expectation operator.

Comment: Although the above-mentioned algorithm was recorded in 2002, it is still relevant today. For example, the authors in [56] use this model to estimate the position and velocity from acceleration measurements. Similarly, this constant acceleration model is also used in [57] in target tracking applications using a Kalman Filter. Finally, in [58] the constant acceleration model is used to fit the movement of UAVs in an EKF framework, in order to reduce localization errors.

4.3 Inertial Measurement Unit

The inertial measurement unit (IMU) provides linear acceleration along the three axes and angular velocity values about the three axes. The accelerometer and the gyroscope, respectively, provide this. Most IMUs also have a magnetometer module that can help detect and measure magnetic fields. However, this module is irrelevant for the application of lunar navigation due to the lack of a magnetic field on the moon.

Today's most common technology in producing inertial sensors is micro-electromechanical systems (MEMS). MEMS IMUs are used in different applications with various complexity. For example, Analog Devices' surface-mount ADIS16470 is a compact IMU that targets a range of applications, including the Internet of Moving Things. Aceninna manufactures 9-DOF high-performance navigation systems and attitude heading reference systems (AHRS) that incorporate Kalman filters with GPS-aided attitude/heading algorithms to provide accurate roll/pitch/heading in dynamic environments. They also offer 6-DOF vertical gyroscopes with Kalman filter and attitude algorithm support.

IMU measurements of angular velocity, $\omega_m \in \mathbb{R}^3$ and linear acceleration, $a_m \in \mathbb{R}^3$ can be described by the following equations [59], with the IMU frame represented by $\{I\}$, and the global frame, $\{G\}$:

$$\begin{aligned} \omega_k &= {}^I\omega + \mathbf{b}_g + \mathbf{n}_g, \ \mathbf{n}_g \sim \mathcal{N}(\mathbf{0}, \sigma_g^2 \mathbf{I}_3) \\ \mathbf{a}_k &= {}^I\mathbf{R}_G({}^G\mathbf{a} - {}^G\mathbf{g}) + \mathbf{b}_a + \mathbf{n}_a, , \ \mathbf{n}_a \sim \mathcal{N}(\mathbf{0}, \sigma_a^2 \mathbf{I}_3) \end{aligned} \tag{4.9}$$

Here, ${}^I\omega$ and ${}^G\mathbf{a}$ represent the the angular velocity and linear acceleration of the IMU expressed in frames $\{I\}$ and $\{G\}$, respectively. ${}^I\mathbf{R}_G$ represents the rotation from $\{G\}$ to

$\{I\}$ and ${}^{G}\mathbf{g}$ is the known gravity vector in the global frame. Finally, $\mathbf{b}_g$ and $\mathbf{b}_a$ represent the measurement biases of the gyroscope and accelerometer, which are modelled as random walk processes, and $\mathbf{n}_g$ and $\mathbf{n}_a$ are the white gaussian noises of the gyroscope and the accelerometer.

4.3.1 Pose Using Inertial Measurement Unit

The IMU provides information on linear and angular accelerations. From these measurements one has to obtain the velocity and position of the robot at each time step. This is obtained by integration, a process called dead reckoning. The pose model from the IMU measurements are as follows [60]:

$$\begin{bmatrix} v_{x,k} \\ v_{y,k} \end{bmatrix} = \begin{bmatrix} v_{x,k-1} \\ v_{y,k-1} \end{bmatrix} + \begin{bmatrix} \Delta T & 0 \\ 0 & \Delta T \end{bmatrix} \begin{bmatrix} a_{x,k} \\ a_{y,k} \end{bmatrix} \tag{4.10}$$

These represent the velocity update along the x- and y-directions. The velocity v at time step k is calculated using the accelerometer acceleration measurements a_k at the same time step, for each axes. The quantity ΔT is the time-step between the previous measurement and the current measurement. The position update is then calculated by integrating the estimated velocities:

$$s_k = s_{k-1} + \int_{t_{k-1}}^{t_k} v_k dt \tag{4.11}$$

Here, s_k is the computed position at time-step k, while v_k is the velocity at the same time-step, also calculated previously. The time interval $[t_k, t_{k-1}]$ is the time-step length. This complete computational process is illustrated in Fig. 4.3

.

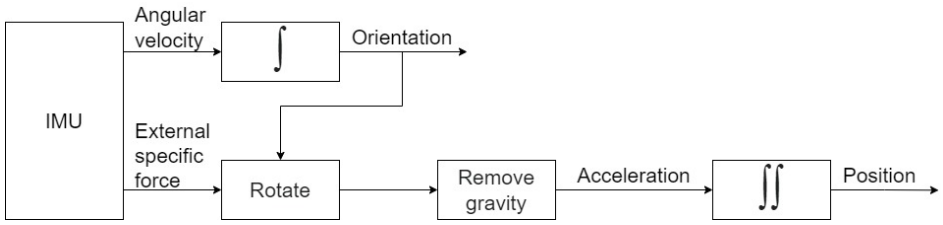


Figure 4.3: Process of estimating position and orientation from the IMU

The pose estimation from a single IMU in a Kalman filter framework is analyzed using the simulated IMU data from the rover simulation of *Dataset 1* described earlier in § 4.1. The resulting estimated pose is shown in Fig. 4.4. Clearly, the estimate drifts considerably from the true position values. The deviation is of the order of kilometres

even though the initial difference is zero. The corresponding position estimates along the x- and y-directions are provided in Fig. 4.5 and Fig. 4.6, respectively. While there is major drift in the y-direction, the drift along the x-direction is exponential. The metric root-mean-square-error (RMSE) is used throughout the rest of the thesis to quantify the errors. This RMSE metric is given by the following equation:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=0}^{N}|e^2(k)|},$$

where e_k is the error between the true value and its estimate, and N is the total number of samples in a dataset.

The RMSE values for the x- and y-position estimates respectively are 79639.4611 and 897.0596. The drift in the estimates is due to the dead reckoning process. There is a double integration at every time-step resulting in accumulation of errors. Such a phenomenon has been reported in the literature; see, for example, [61]. To conclude, pose estimated using only IMU measurements is highly inaccurate and unreliable. It may be emphasized that in applications utilizing the IMU, this typically involves integrating its data with that of other sensors. For example, the IMU is used to correct and smooth-out possible errors in GPS measurements in localization of vehicles in [62]. Here, the IMU measurements are used sparingly only when there is a perceived error in GPS measurements. In [63], the IMU is used in combination with other sensors such as the LiDAR, for robot localization. In aerospace applications, particularly during extensive traversals exceeding 100 kilometers, LiDAR can serve to refine the position estimated via the IMU, as demonstrated in [64]. Employing this method resulted in a final position error of 27 meters over a 218-kilometer flight, which was only 0.012% of the total distance covered. Later in this thesis, combining IMU sensor measurements with other sensor measurements is explored.

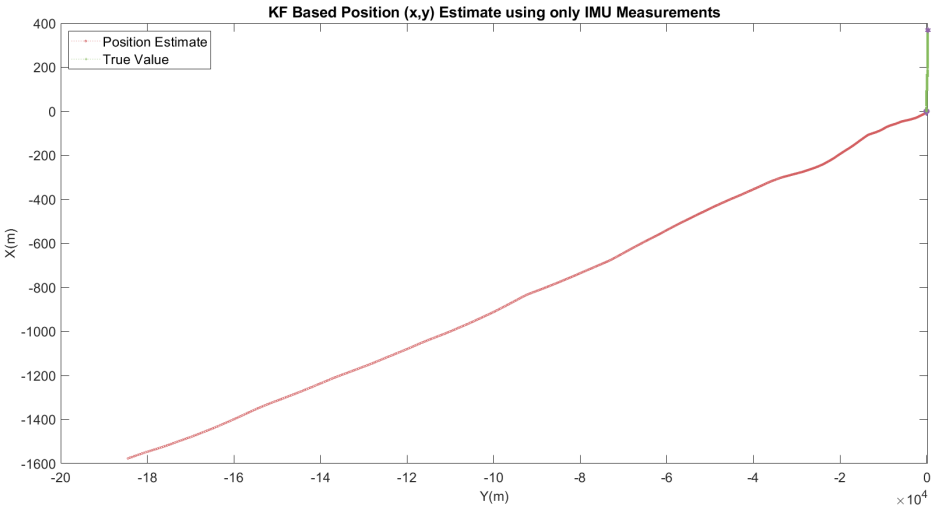


Figure 4.4: Pose Estimation Using the IMU (Dataset 1)

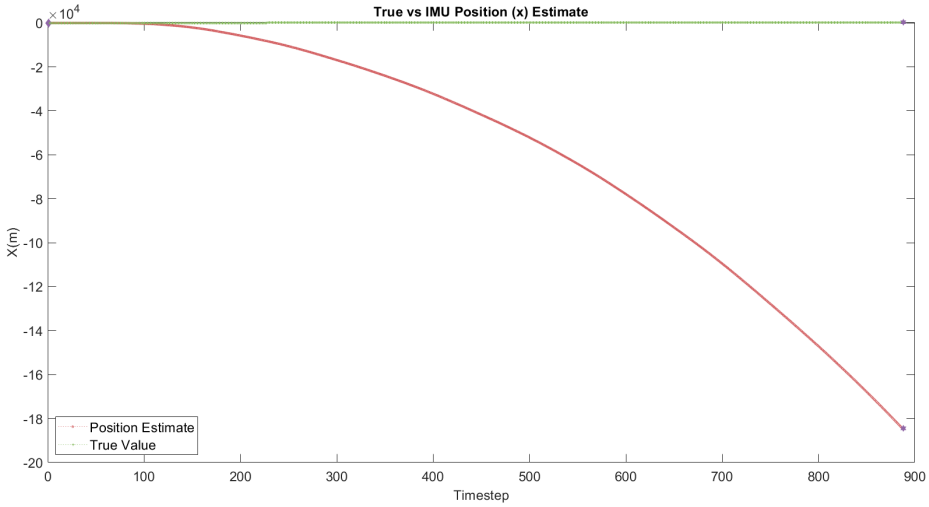


Figure 4.5: Comparison of true and IMU-estimated position in the x direction (Dataset 1)

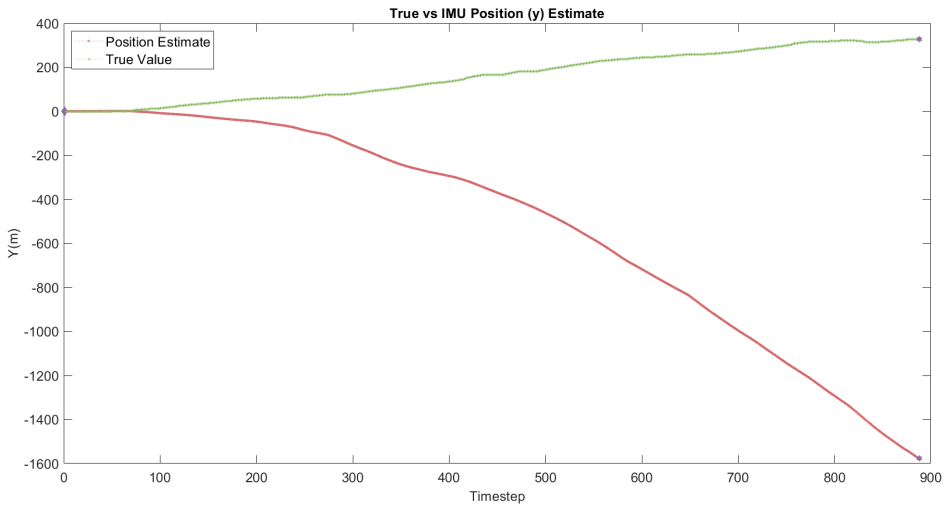


Figure 4.6: Comparison of true and IMU-estimated position in the y direction (Dataset 1)

Similarly, the results of pose computation using an IMU with Dataset 2 are illustrated in Fig. 4.7. These estimates are relatively better when compared with those obtained with Dataset 1, and the RMSE values reflect these observations. For the x- and y-position estimates, these values respectively are 52504.5042 and 264.4270. From these values pertaining to both datasets, it can be concluded that the IMU alone results in rather poor estimates. As explained earlier, this is due to the drift phenomenon. In addition, the quality of the dataset affects the outcomes. It may be recalled that the IMU measurements in both Dataset 1 and Dataset 2 are derived from the same Devon Island rover dataset with the help of simulation tools in MATLAB. While the resulting heading angle is quite reasonable, the acceleration sometimes does not appear to be satisfactory due to overcompensation.

4.4 Wheel Encoders

Wheel encoder sensors are used to count the number of times the wheel motor has rotated, often called *wheel ticks*. They are usually attached to each wheel. The wheel encoder consists of:

- a Hall Effect Sensor which measures the strength of the magnetic field
- a ring magnet which is attached to the motor shaft

When the motor rotates the wheel, it additionally rotates the ring magnet, causing a change in the magnetic field. The Hall effect sensor positioned near the ring detects

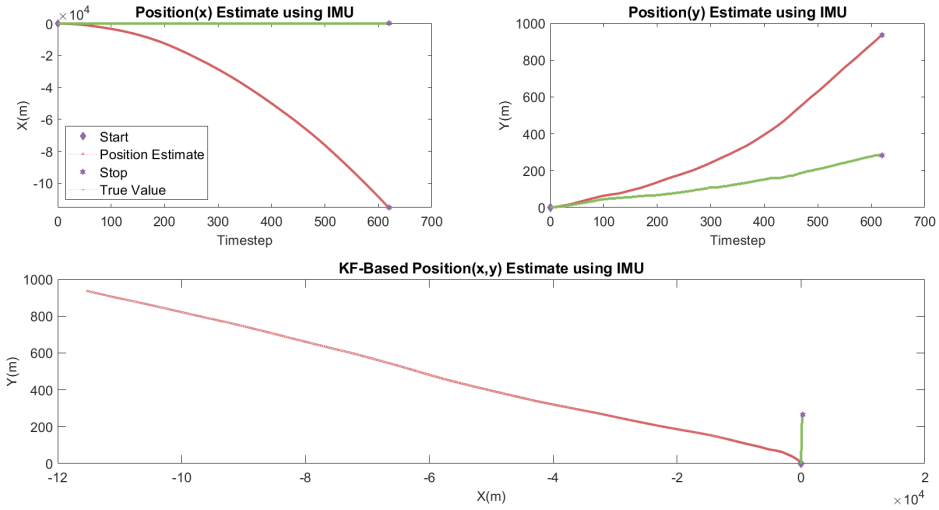


Figure 4.7: Position Estimation Using IMU for Dataset 2

Table 4.2: Computational flow for Estimating Position Using IMU in KF framework.

Matrix Initialisation: State Matrix $A = \begin{bmatrix} 1 & T & \frac{1}{2}T^2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}$, Measurement Matrix $H = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$, Process and Noise Covariance Matrices Q and R

Initialisation : Sampling time T, State Estimate $\hat{S}_0$, State Variance P_0, $z_1, \ldots, z_n$ are the acceleration measurements from IMU

for $k = 1, 2, \ldots$ n **do**

Compute $\hat{S}_k$ with A and S_{k-1}.

Compute P_k with A, P_{k-1} and Q {(4.43)}

Determine $\tilde{y}_k$, M_k and K_k with z_k, H, S_k, P_k and R {(4.44)}

Update $\hat{S}_k$ with $\tilde{y}_k$, K_k and S_k

Update $P_{k|k}$ with K_k, H and P_k {(4.45)}

end for

---

changes in the magnetic field as the ring rotates. This is how the sensor counts the number of times the wheel has rotated.

Popular configurations for a wheeled robot include [65]:

- *Differential drive:* Differential drive has two motorized wheels that are steerable and can be driven independently either forward or backward, and one passive wheel (in general).

- *Synchro drive:* The wheels are mechanically coupled. All synchro drive robot has two motors: a drive motor and a steering motor. The drive motor sets the speed for all three wheels and the steering motor spins all three wheels together. All wheels rotate at the same speed and orientation. The three wheels point at the same direction and turn at the same rate.

- *Ackermann drive:* Here, there two steerable wheels in the front plus two non-steerable wheels in the back. At least two wheels connected by an axle are motorized.

The Lunar Zebro has a differential drive wheeled configuration, as designed by the locomotion and mechanical engineers of the development team. For the robot to perform rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. The point that the robot rotates about is known as the Instantaneous Center of Curvature (ICC). This is illustrated in Fig. 4.8.

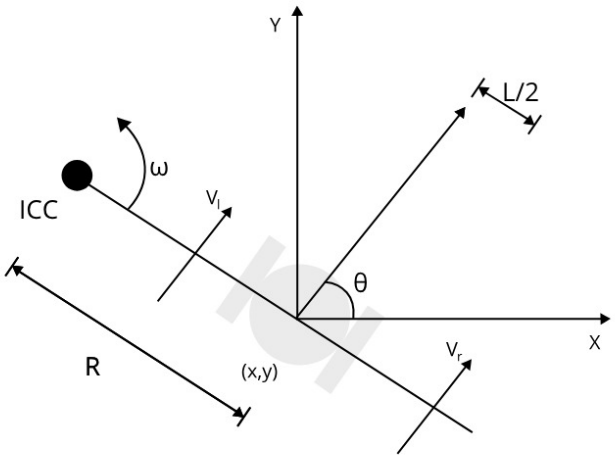


Figure 4.8: Illustration of the differential drive wheeled robot

4.4.1 Pose Using Wheel Encoders

Given the information on wheel rotations, one can derive pose information using the process illustrated below:

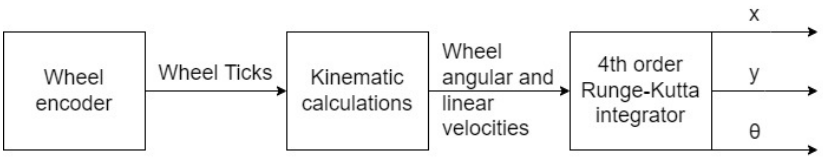


Figure 4.9: Process of estimating position and orientation from the wheel encoder

The equations of pose can be derived using the kinematic and trigonometric relations applied to robot motion, as pictured in Fig. 4.10. The left and right wheel angular velocities in radians per second (ν_l, ν_r) are given by,

$$\begin{aligned} \nu_l &= \frac{\text{Encoder ticks since previous loop (left motor)}}{\text{Time elapsed since last poll from encoders}} \frac{\pi}{180} \\ \nu_r &= \frac{\text{Encoder ticks since previous loop (right motor)}}{\text{Time elapsed since last poll from encoders}} \frac{\pi}{180} \end{aligned} \tag{4.12}$$

The linear velocities (in m/s) thus travelled by the wheels, whose radius is R, is given by

$$\begin{aligned} V_l &= \nu_l R \\ V_r &= \nu_r R \end{aligned} \tag{4.13}$$

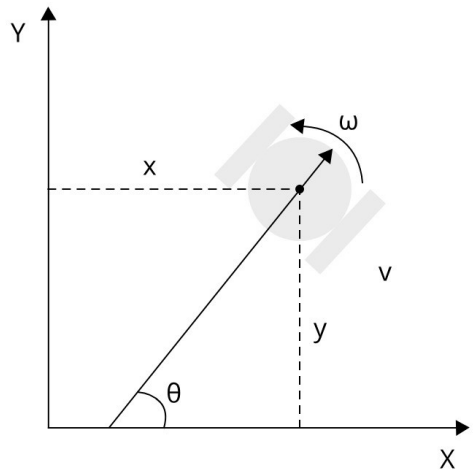


Figure 4.10: Dead Reckoning of a Robot in Motion using Differential Drive Rover Model

The linear (in m/s) and angular velocity (in rad/sec) for the entire robot can now be obtained. Let L be the wheel base, which is calculated as the distance between the

point of contact of the two wheels to the ground (in meters). Then,

$$\begin{aligned} \nu &= \frac{V_r + V_l}{2} \\ \omega &= \frac{V_r - V_l}{L} \end{aligned} \tag{4.14}$$

From the computed velocities of the robot, its position (in meters) and its orientation (in radians) are determined as follows:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \nu\cos(\theta) \\ \nu\sin(\theta) \\ \omega \end{pmatrix} \tag{4.15}$$

To obtain $\begin{pmatrix} x & y & \theta \end{pmatrix}^T$, this non-linear system of differential equations must be solved. The fourth-order Runge-Kutta method is a good technique to numerically solve differential equations:

$$\begin{aligned} k_{00} &= \nu\cos(\theta_{k-1}) \\ k_{01} &= \nu\sin(\theta_{k-1}) \\ k_{02} &= \omega \\ \\ k_{10} &= \nu\cos(\theta_{k-1} + \frac{t}{2}k_{02}) \\ k_{11} &= \nu\sin(\theta_{k-1} + \frac{t}{2}k_{02})) \\ k_{12} &= \omega \\ \\ k_{20} &= \nu\cos(\theta_{k-1} + \frac{t}{2}k_{12}) \\ k_{21} &= \nu\sin(\theta_{k-1} + \frac{t}{2}k_{12})) \\ k_{22} &= \omega \\ \\ k_{30} &= \nu\cos(\theta_{k-1} + \frac{t}{2}k_{22}) \\ k_{31} &= \nu\sin(\theta_{k-1} + \frac{t}{2}k_{22})) \\ k_{32} &= \omega \end{aligned} \tag{4.16}$$

Finally, the pose can be computed as:

$$\begin{pmatrix} x_k \\ y_k \\ \theta_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{pmatrix} + \frac{t}{6} \begin{pmatrix} k_{00} + 2(k_{10} + k_{20}) + k_{30} \\ k_{01} + 2(k_{11} + k_{21}) + k_{31} \\ k_{02} + 2(k_{12} + k_{22}) + k_{32} \end{pmatrix} \tag{4.17}$$

For clarity, t is the time elapsed since the last integration loop and k represents the current timestamp or sample number of the loop.

Table 4.3: Computational flow for Pose Estimation By Odometry using Wheel Encoders.

Initialisation: Vehicle track width L, wheel radius R, vehicle x position x_1,vehicle y position y_1, vehicle orientation θ_1

for $k = 2, 3, \ldots$ n **do**

Compute ν_l and ν_r with encoder ticks and time. {(4.12)}

Compute V_l and V_r with ν_l, ν_r and R. {(4.13)}

Compute ν and ω with L, V_l and V_r. {(4.14)}

Determine constants k_{00}, k_{01}, k_{02}, k_{10}, k_{11}, k_{12}, k_{20}, k_{21}, k_{22} with ν, ω and θ_{k-1}. {(4.16)}

Update x_k,y_k and θ_k with x_{k-1}, y_{k-1}, θ_{k-1} and k_{00}, k_{01}, k_{02}, k_{10}, k_{11}, k_{12}, k_{20}, k_{21} and k_{22}. {(4.17)}

end for

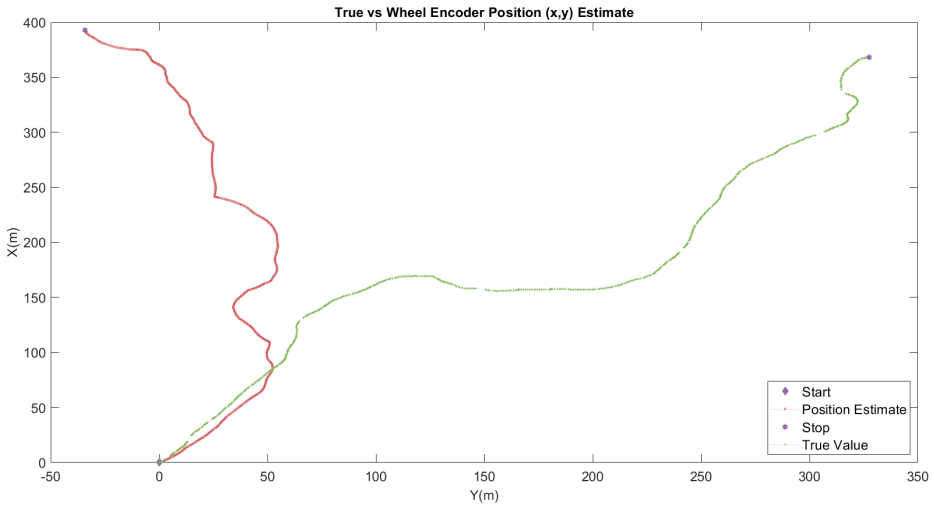


Figure 4.11: Position Estimation of the robot Using Wheel Encoder

The wheel encoder ticks information simulated using MATLAB is used to evaluate

this method. Note that this information is derived and simulated using GPS measurements as waypoints with their arrival times (see § 4.1). The computational flow is provided in Table 4.3. Dataset 1 is used for the simulations. The sampling rate is set at 1 Hz, the same as that for the sun sensor used in the Devon Island rover dataset. That is, the sampling interval is 1 s. The results are shown in Fig. 4.11. It can be observed that initially, the position estimates of the rover closely match the true values. However, as time passes, the error between the two continuously increases. Such an increase is expected in the case of a dead reckoning method wherein there is a process of integration resulting in the accumulation of errors.

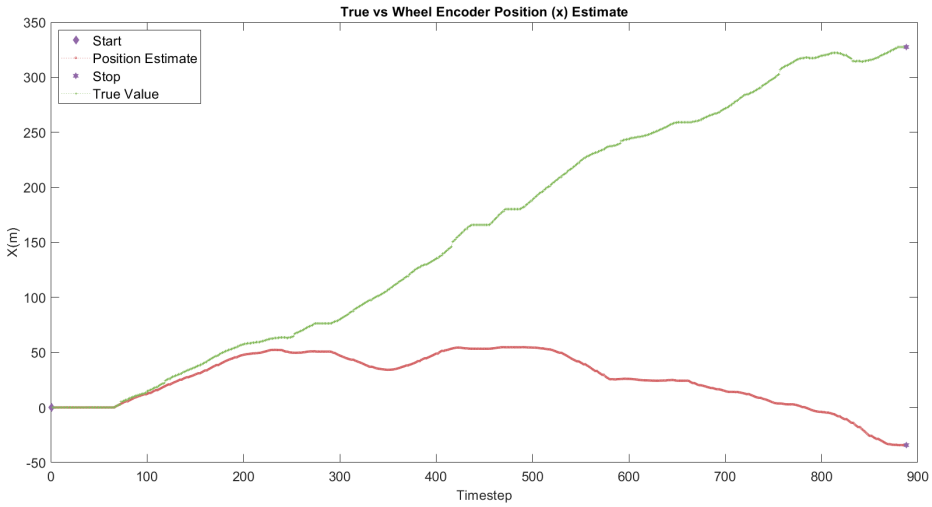


Figure 4.12: Comparison of true and wheel encoder odometry-based estimated position in the x direction

The individual errors along the x- and y-directions are shown in Fig. 4.12 and Fig. 4.13, respectively. The accumulated drifts in the position estimates are clearly exhibited and are more pronounced along the x direction. This may be attributed to the specific path of the rover, wherein there is an apparent change in the direction along this direction. This change in the direction is not captured very well by the wheel encoder. In contrast, the rover's movement along the y-direction is in the same direction, resulting in fewer errors. A similar phenomenon was observed with the IMU sensor. However, the drifts with the wheel encoder are quite small relative to those observed with the IMU sensor.

The estimates of the orientation angle are depicted in Fig. 4.14. Although the changes in the estimation angles reflect well with those in the heading angle, a bias is observed, which can be as large as 2 rad. The position estimates are computed at

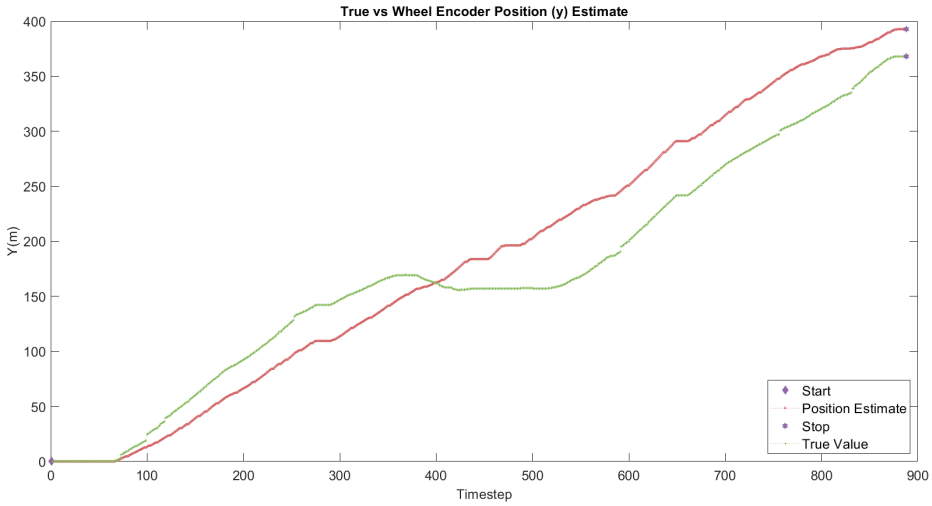


Figure 4.13: Comparison of true and wheel encoder odometry-based estimated position in the y direction

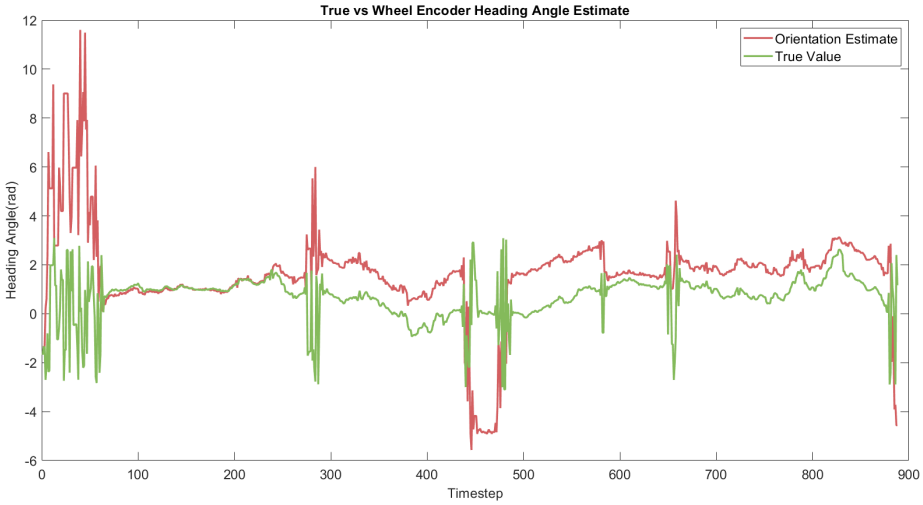


Figure 4.14: Orientation Angle Estimation Using Wheel Encoder

any instant using the heading estimates from the previous instant. Thus, errors in these estimates automatically reflect on the position estimates. Integrating the errors and the error in the orientation angle are the main barriers to using wheel odometry for pose computations alone. The RMSE values for the x-position, y-position and the orientation estimates are 182.5101, 36.5429, and 65.0466, respectively. These values concur with the observations made from the Figures 4.12–4.14, and they are much lower than those obtained using an IMU. The estimates obtained using the wheel encoder

are clearly better than those obtained with an IMU.

The results of pose computation using a wheel encoder with Dataset 2 are shown in Fig. 4.15. The trend observed here is opposite that observed with Dataset 1. Here, the x-position estimates are better than the y-position estimates even though the orientation estimates are of the same order. The corresponding RMSE values of 38.6495, 202.6770, and 51.7827 agree with these observations. Overall, one can conclude that errors are present when the wheel encoder is used.

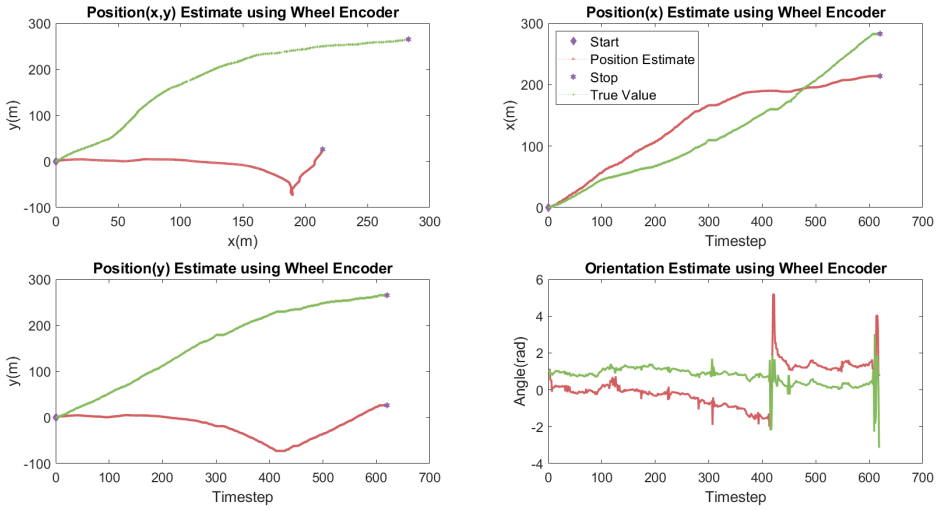


Figure 4.15: Pose Estimation Using Wheel Encoder for Dataset 2

4.5 Sun Sensor

Sun sensors are navigational instruments used, especially by spacecrafts, to detect the Sun's position. Sun sensors are used for attitude control, solar array pointing, gyro-updating, and fail-safe recovery. Essentially, the sun sensor is a device that can determine the attitude between the sensor's coordinate system and the Sun, which comprises the so-called *Sun vector*. After image acquisition and post-processing, the sun sensor determines the unit vector pointing from the sensor to the Sun. This unit vector can be described entirely relative to the sun sensor frame by the measurement $\mathbf{s}_k$ consisting of a rotation about the x-axis by angle ϕ_k and a rotation about the y-axis by θ_k. (It is important to note that these angles ϕ_k and θ_k are the sun sensor measurements or the azimuth and altitude angles to the Sun. These are different from the angle θ used to represent the rover's orientation and ϕ used in the rest of the thesis to describe latitude.) The angles and their representation are depicted in Fig. 4.16. There are sun

sensors that are camera-based and those that are not. In this thesis, the sun sensor used is the SS411 Sinclair Interplanetary sensor which does not have a camera. This sun sensor requires low power and uses limited computational resources [46].

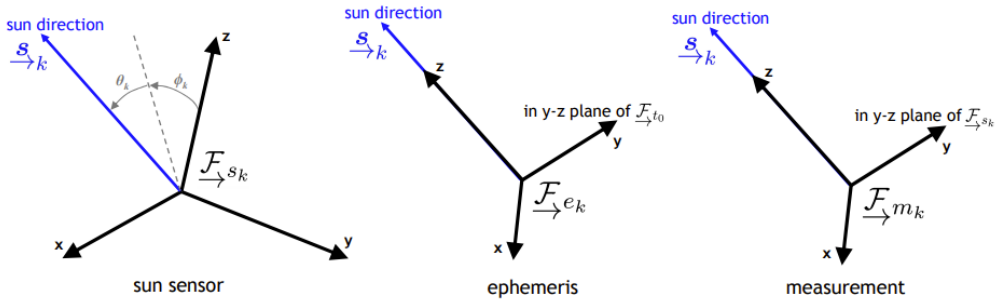


Figure 4.16: Representation of the different sun sensor measurement angles, with the relevant sun sensor measurement and ephemeris frames [66].

4.5.1 Pose (Orientation) using Sun Sensor

Deriving orientation information from the Sun sensor is a more complex process. An overview of the process which is described in this section is illustrated in the block diagram in Fig. 4.17. In what follows, this process is explained and derived in detail.

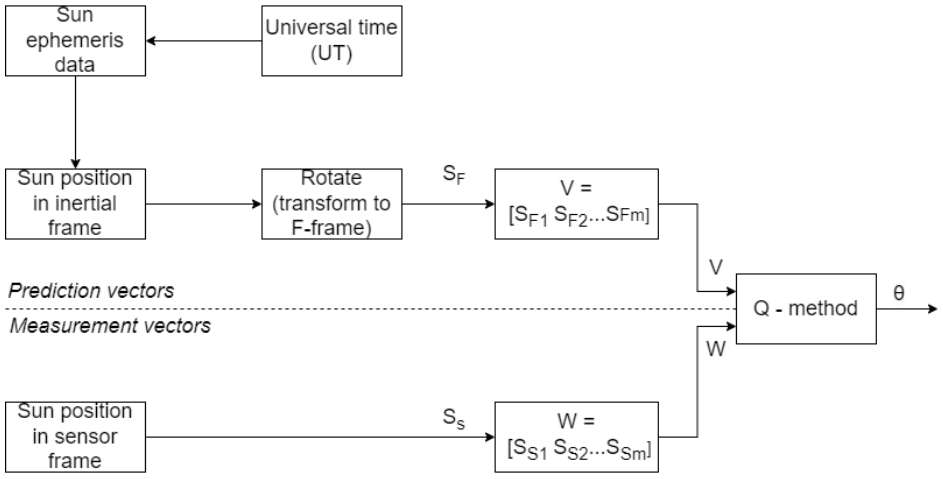


Figure 4.17: Illustration of the Process of Estimating Orientation using Sun Sensor

Sun sensor data is obtained in the sun sensor frame. The heading angle computation using this sensor requires calculating solar ephemeris data, which is a tabulation of computed positions of the Sun. Computations for solar ephemeris are described and verified previously (see § 3.4). Given the definitions of rotation matrices and the

illustration of the different reference frames in Fig. 3.1, one can derive expressions for converting between frames of interest. The transformation from ECF to ECI is the simple rotation $\mathbf{C}_{IF}$, where Ψ is the Greenwich Apparent Sidereal Time (GAST).

$$\mathbf{C}_{IF} = \mathbf{R}_z(\Psi), \tag{4.18}$$

The transformation between frames T and F is a function of the rover longitude, λ and ϕ. From these definitions,

$$\mathbf{C}_{FT} = \mathbf{R}_z(\lambda)\mathbf{R}_y\left(\frac{\pi}{2} - \phi\right)\mathbf{R}_z(\frac{\pi}{2}) \tag{4.19}$$

Finally, the transformation between the sun sensor and topocentric frames can be computed using the three rotations roll γ, pitch β and heading or yaw, α.

$$\mathbf{C}_{TS} = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma) \tag{4.20}$$

The final transformation between the sun sensor and the IMU frame is calculated using the alignment of the two sensors as compared to the rover body.

Consider the rover with longitude, λ and latitude, ϕ. The *predicted Sun vector*, $\mathbf{s}_F$, is computed using the transformation from the *ECI* to *ECF* frame,

$$\mathbf{s}_F = \mathbf{C}_{FI}\mathbf{s}_I \tag{4.21}$$

where, the sun vector prediction, $\mathbf{s}_I$ is the vector notation of the solar ephemeris azimuth and zenith data that were previously calculated in § 3.4 (page 24). For example, considering a calculated predicted azimuth angle, az and a calculated predicted zenith angle, zen, the predicted sun vector in ECI frame is computed using (4.22).

$$\mathbf{s}_I = \begin{bmatrix}\sin(zen)\cos(az) & \sin(zen)\sin(az) & \cos(zen)\end{bmatrix} \tag{4.22}$$

These can be corrected, if required, using the equations that are further provided in § 4.5.1.1. These corrections are necessary in cases of atmospheric refraction of the sun rays.

Consider the sun sensor that provides the *measured sun vector*, s_S. Known information is considered to estimate orientation. The objective is to find the best estimate of the rotation matrix $\mathbf{C}_{SF}$ that can explain the set of observation vectors $\mathbf{u}_{Si}$, obtained from the Sun sensor and the predictions $\mathbf{u}_{Fi}$, calculated as ephemeris data. This is similar to the classic Wahba's problem [67], where the aim is to find $\mathbf{C}_{SF}$ that minimizes a scalar-weighted cost function,

$$J(\mathbf{C}_{SF}) = \frac{1}{2}\sum_{i=1}^{m}\|\mathbf{u}_{Si} - \mathbf{C}_{SF}u_{Fi}\|^2 \tag{4.23}$$

where a_i is the scalar weight, m is the number of measurements, and $\|\cdot\|$ represents the Euclidean norm. To solve this Wahba's problem, Davenport's q-method [68] is used. (The full algorithm is explained in Appendix.) The sets of observations and predictions are concatenated into two matrices:

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix}\sqrt{a_1}\mathbf{u}_{S1} & \sqrt{a_2}\mathbf{u}_{S2} & \dots & \sqrt{a_m}\mathbf{u}_{Sm}\end{bmatrix} \\ \mathbf{V} &= \begin{bmatrix}\sqrt{a_1}\mathbf{u}_{F1} & \sqrt{a_2}\mathbf{u}_{F2} & \dots & \sqrt{a_m}\mathbf{u}_{Fm}\end{bmatrix}\end{aligned} \tag{4.24}$$

The following matrices are then calculated:

$$\mathbf{B} = \mathbf{W}\mathbf{V}^T \quad \mathbf{Q} = \mathbf{B} + \mathbf{B}^T \tag{4.25}$$

The cyclic components of $\mathbf{B}$ are extracted next. Let B_{ij} be the element of $\mathbf{B}$ in row i and column j. Thus,

$$\mathbf{Z} = \begin{bmatrix}B_{23} - B_{32} & B_{31} - B_{13} & B_{12} - B_{21}\end{bmatrix} \tag{4.26}$$

The trace of $\mathbf{B}$ is also required:

$$\sigma = \text{trace}(\mathbf{B}) \tag{4.27}$$

The following 4×4 matrix $\mathbf{K}$ is constructed

$$\mathbf{K} = \begin{bmatrix}\mathbf{Q} \text{ -}\mathbf{1}\sigma & \mathbf{Z}^T \\ \mathbf{Z} & \sigma\end{bmatrix} \tag{4.28}$$

where $\mathbf{1}$ is 3×3 matrix of ones. Let the eigenvector corresponding to the largest eigenvalue of $\mathbf{K}$ be $\mathbf{q}_{SF}$. This is the 4×1 unit quaternion that represents the best fit from frame F to frame S. The quantity $\mathbf{q}_{SF}$ comprises of a vector component $\mathbf{q}_v$ and a scalar $\mathbf{q}_s$:

$$\mathbf{q}_{SF} = \begin{bmatrix}\mathbf{q}_v \\ q_s\end{bmatrix} \tag{4.29}$$

Finally, the desired rotation matrix, $\mathbf{C}_{SF}$ is found using this quaternion

$$\mathbf{C}_{SF} = (q_s^2 - \mathbf{q}_v^T\mathbf{q}_v)\mathbf{1} + 2\mathbf{q}_v\mathbf{q}_v^T - 2q_s\mathbf{q}_v^{\times} \tag{4.30}$$

Here, the $\mathbf{q}_F^{\times}$ term is the skew-symmetric matrix that is formed from the components of the vector $\mathbf{q}_v$.

$$\mathbf{q}_v^{\times} = \begin{bmatrix}0 & -q_{v3} & q_{v2} \\ q_{v3} & 0 & -q_{v1} \\ -q_{v2} & q_{v1} & 0\end{bmatrix} \tag{4.31}$$

The weights a_i are assumed equal with unity values, as considered in [48].

The required rotation matrix, $\mathbf{C}_{SF}$ relates frames F and S. However, the heading angle of the rover α is present in the transformations between T and S. The heading anlge α is extracted by first calculating $\mathbf{C}_{TS}$ using the rover's latitude ϕ, and longitude λ using (4.19). The transformation required from T and S, $\mathbf{C}_{TS}$ can be written as:

$$\mathbf{C}_{TS} = \mathbf{C}_{TF}\mathbf{C}_{FS} = \mathbf{C}_{FT}^{T}\mathbf{C}_{SF}^{T} \tag{4.32}$$

where the two known matrices $\mathbf{C}_{FT}^{T}$ and $\mathbf{C}_{SF}^{T}$ are known. Using (4.20), the final transformation can be written as follows:

$$\mathbf{C}_{TS} = \begin{bmatrix} c_\alpha c_\beta & c_\alpha c_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha c_\beta c_\gamma - s_\alpha c_\gamma \\ s_\alpha c_\beta & -s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & -s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma \\ -s_\beta & c_\beta s_\gamma & c_\beta c_\gamma \end{bmatrix} \tag{4.33}$$

Thus, by equating the computed $\mathbf{C}_{TS}$ in (4.32) and (4.33), the rover heading α is found from the components of $\mathbf{C}_{TS}$:

$$\theta = \text{atan2}(C_{TS_{21}}, C_{TS_{11}}) \tag{4.34}$$

The computational flow is provided in Table 4.4.

The approximate heading angle is determined using the algorithm described above and the Sun sensor measurements from Dateset 1 extracted from those recorded in [53]. A sliding window of three measurements and prediction values is used to construct the W and V vectors. The results are depicted in Fig. 4.18. Evidently, these estimates of orientation and heading angle are accurate, according to the trend and changes in angle; however, a bias is present. This bias may perhaps be attributed to the absence of calibration of the Sun sensor to account for its field of view. No calibration information regarding the sun sensor in the Devon Island rover dataset is available. In addition, the accuracy of the Sun sensor angles has not been verified or commented upon in [53]. Additionally, there are durations where the Sun sensor data are not recorded when the rover is stationary. Furthermore, it is evident from the figure that although the resulting trajectory passes through the given waypoints accurately, the resulting orientation angles have large spikes and sudden changes. These are impossible to occur in reality.

Table 4.4: Computational flow for Orientation Estimation Using Sun Sensor.

Initialisation: time h, m, s, latitude lat, lon	
for $k = 1, 2, \ldots$ **do**	
Compute $JD(k)$ with $h(k), m(k)$ and $s(k)$.	{(**??**) and (3.4.1)}
Compute $GAST(k)$ with $JD(k)$	{(3.9)–(3.13)}
Compute ephemeris in ECI : $azimuth(k)$ and $zenith(k)$ with $JD(k), GAST(k), lat(k)$ and $lon(k)$.	{(3.14)–(**??**)}
Compute $C_{if}(k)$ with $GAST(k)$.	{(4.18)}
Compute $s_I(k)$ with $azimuth(k)$ and $zenith(k)$	{(4.22)}
Compute $s_F(k)$ with $C_{if}(k)$ and $s_I(k)$	{(4.21)}
Compute $s'_F(k)$	{(4.39) and § 4.5.1.1}
end for	
Initialisation: Window length WL	
for $k = WL + 1, WL + 2, \ldots$ **do**	
Construct $W(k)$ with $s_I(k), s_I(k-1), \ldots, s_I(k-WL)$	
Construct $V(k)$ with $s'_F(k), s'_F(k-1), \ldots, s'_F(k-WL)$	{(4.24)}
Compute $\theta_{ss}(k)$ with $W(k)$, $V(k)$, $lat(k)$ and $lon(k)$	{(A.11)–(4.34)}
end for	

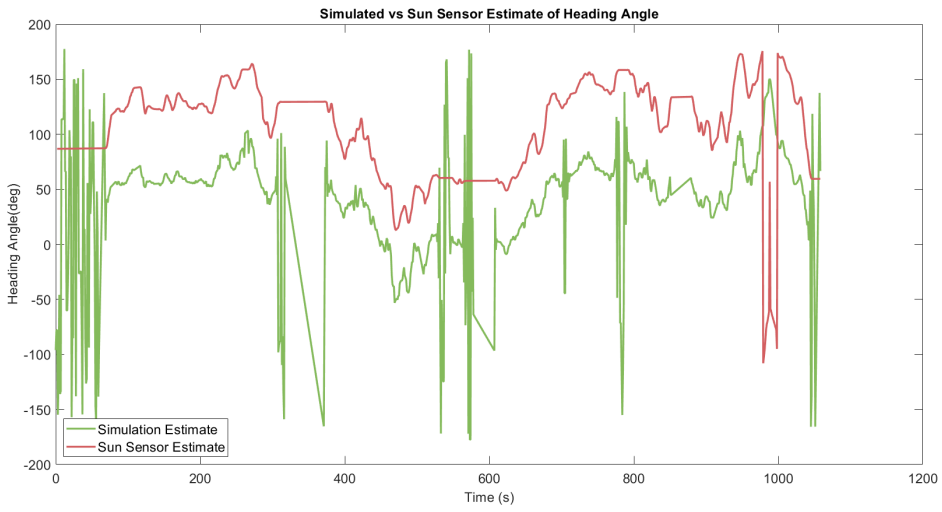


Figure 4.18: Orientation Estimation Using Sun Sensor Measurements and Davenport's q Method for Dataset 1

An inherent bias can be observed in the Sun sensor-based heading estimation method

(Fig. 4.18). The following steps are performed to correct this bias. The simulated estimates of the heading angle are very noisy and include heading angle changes from instant to instant, which is not possible on a real rover. Therefore, the data is filtered using a median filter. (Median filtering is a technique often used to remove random noise [69].) The output of the median filter is illustrated in Fig. 4.19. Finally, a fixed sliding window of 25 data samples is used to correct the bias, analogous to calibration and re-calibration of the sensor in a real-life scenario. (The window size is obtained after conducting several experiments. A larger window does not improve the bias removal, and there is performance deterioration when a smaller window is used.) The resulting data is depicted in Fig. 4.20. It shows the orientation estimate from the sun sensor corrected for its bias. After bias correction, the orientation estimate's output with Dataset 2 is illustrated in Fig. 4.21. The corresponding RMSE values of the orientation estimates are 50.7096 and 22.2837, respectively. Thus, as far as the Sun sensor is concerned, Dataset 2 is better than Dataset 1. This may be attributed to the fact that orientation angle estimates with Dataset 1 appears noisier when compared with those obtained using Dataset 2.

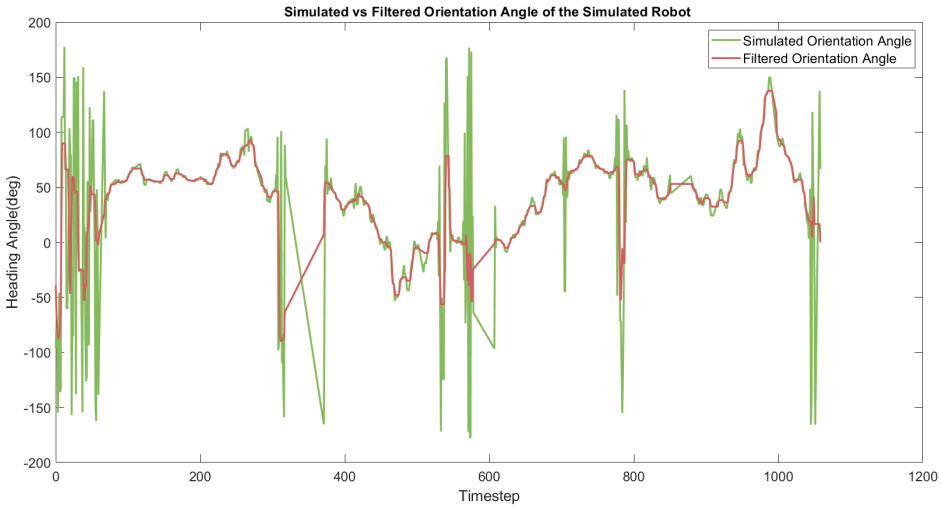


Figure 4.19: Filtered Output of the Simulated Heading Angle, to be used to Correct Bias in Sun Sensor Heading for Dataset 1

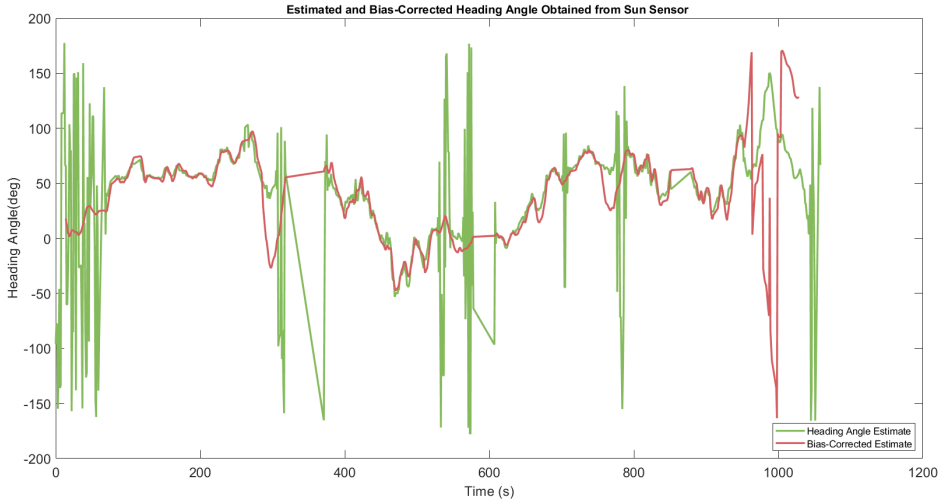


Figure 4.20: Bias Corrected Orientation Estimate Using Sun Sensor Measurements for Dataset 1

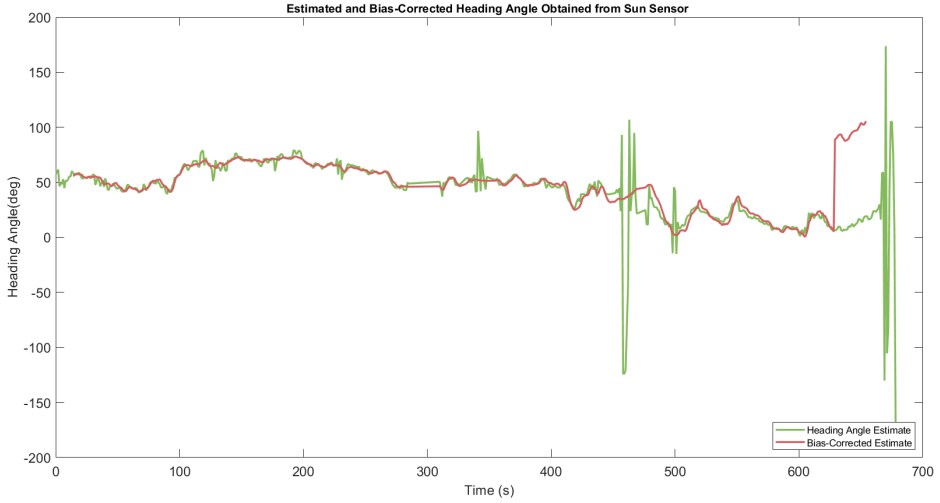


Figure 4.21: Bias Corrected Orientation Estimate Using Sun Sensor Measurements for Dataset 2

4.5.1.1 Refraction Corrections

Due to refraction of sunlight as it enters the atmosphere, there exists an apparent increase in elevation of the sun from the horizon. This phenomenon is inconsequential for bodies without an atmosphere, such as the moon, but must be accounted for in terrestrial trials. The ephemeris predictions $\mathbf{s}_F$ can be corrected to reflect the apparent

position of the Sun. Let the true and apparent elevation angles be ε_{true} and ε_{app}, respectively. Thus, they can be modelled as:

$$\varepsilon_{app} = \varepsilon_{true} + \Delta\varepsilon \tag{4.35}$$

If the rover's position is known, the predicted gravity vector $\mathbf{g}_F$ can also be computed:

$$\mathbf{g}_F = -\begin{bmatrix} c_\phi c_\lambda \\ c_\phi s_\lambda \\ s_\phi \end{bmatrix} \tag{4.36}$$

The true elevation is computed using the Sun and gravity vectors:

$$\varepsilon_{true} = \cos^{-1}(\mathbf{s}_F^T \mathbf{g}_F) - \frac{\pi}{2} \tag{4.37}$$

The elevation correction $\Delta\varepsilon$ is given by,

$$\Delta\varepsilon = \frac{2.97x10^{-4}}{\tan(\varepsilon_{true} + \frac{0.180}{\varepsilon_{true}+0.0892})} \tag{4.38}$$

This apparent change in location due to the refraction through the atmosphere is illustrated in Fig. 4.22.

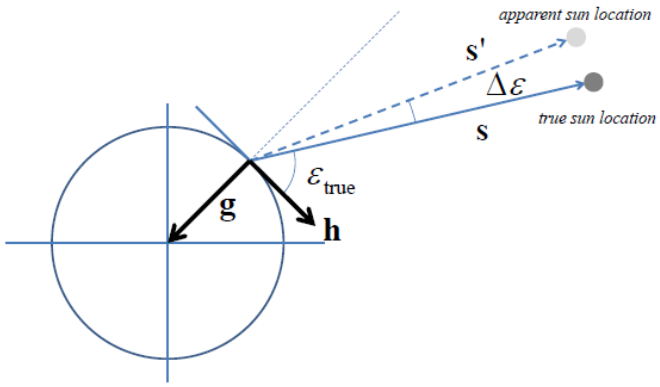


Figure 4.22: Illustration of a cutaway through the Earth depicting the geometry of refraction correction. [48]

The corrected Sun vector, as can be derived from Fig. 4.22, is then given by,

$$\mathbf{s}'_F = \sin\varepsilon_{app}\mathbf{h}_F - \cos\varepsilon_{app}\mathbf{g}_F \tag{4.39}$$

Here, the local horizontal vector towards the sun, $\mathbf{h}_F$, is

$$\mathbf{h}_F = \frac{-\mathbf{g}_F^{\mathrm{x}}\mathbf{g}_F^{\mathrm{x}}s_F}{\cos\varepsilon_{true}} \tag{4.40}$$

where $\mathbf{g}_F^{\times}$ is the skew-symmetric matrix that is formed from the components of the vector $\mathbf{g}_F$:

$$\mathbf{g}_F^{\times} = \begin{bmatrix} 0 & -g_{F3} & g_{F2} \\ g_{F3} & 0 & -g_{F1} \\ -g_{F2} & g_{F1} & 0 \end{bmatrix} \tag{4.41}$$

4.6 Sensor Fusion Techniques

Sensor fusion is a technique used to combine data from multiple sensors to generate a more accurate and robust conclusion about the environment than using individual sensors alone. Different sensor fusion methods can be employed in this thesis, with different estimates from sensors [70]. These are described in the following subsections, using an example of measurements obtained from an odometer, θ_{odom} and a sun sensor, θ_{ss}. However, these techniques could be extended using other sensors or even alternatives.

4.6.1 Simple Fusion

Simple fusion makes use of a threshold. In the context of this thesis, this threshold is based on the angular acceleration:

$$\theta_f = \begin{cases} \theta_{odom}, & |\alpha| < \alpha_t \\ \theta_{ss}, & |\alpha| > \alpha_t \end{cases} \tag{4.42}$$

Thus, if the absolute angular acceleration α is below a threshold α_t, then the fused heading θ_f is determined by the encoder readings θ_{odom} . Otherwise, the fused heading is determined based on the sun sensor module θ_{ss}.

4.6.2 Kalman Filter Based Fusion

The Kalman filter is an efficient recursive means to estimate the state of a process, provided the corresponding state-space representation is available. Essentially, the mean of the squared error is minimised. This filter is quite powerful and has found diverse applications in various fields over the past many decades. The Kalman filter can be used even when the precise nature of the modelled system is unknown [71].

The Kalman filter requires a series of computations. The estimate of the state $\hat{S}_k$ and its corresponding covariance P_k is first computed. It consists of the time update

$$\begin{aligned} S_k &= A\hat{S}_{k-1} \\ P_k &= AP_{k-1}A_k^T + Q_k \end{aligned} \tag{4.43}$$

followed by the measurement update where the above state estimate is corrected with information from the measurements z_k.

$$\begin{aligned} \tilde{y_k} &= z_k - H_k S_k \\ M_k &= H_k P_k H_k^T + R_k \\ K_k &= P_k H_k^T M_k^{-1} \end{aligned} \tag{4.44}$$

Here, Q_k and R_k are the process and measurement noise covariance matrices, respectively, while A_k and H_k are the state transition, and observation matrices for the corresponding state-space model, respectively. Finally, the predicted state $\hat{S}_k$ and covariance P_k are computed.

$$\begin{aligned} \hat{S}_{k|k} &= S_k + K_k \tilde{y_k} \\ P_{k|k} &= (\mathbf{I} - K_k H_k) P_k \end{aligned} \tag{4.45}$$

4.6.3 Linear Weighted Fusion

In this type of fusion, the different estimates are combined linearly with different weights. Thus, the fused orientation angle, θ_f, for each time-step, $\theta_f \in \mathbb{R}$ can be written as:

$$\theta_f = k_{odom}\theta_{odom} + k_{ss}\theta_{ss} \tag{4.46}$$

The weighting assigned to the encoder heading is denoted by k_{odom} and the sun sensor heading weighting is denoted as k_{ss}. The weighting may vary as a function of the robot's angular velocity ω and are unit interval values. This function could be linear or non-linear, and can be defined by:

$$\begin{aligned} k_{odom} &= f(\omega) \\ k_{ss} &= 1 - k_{odom} \end{aligned} \tag{4.47}$$

4.7 Sensor Fusion Using Two Sensors

In this section, the techniques that use two of the three sensors discussed previously implemented in this thesis are described.

4.7.1 Kalman Filter Based Sensor Fusion for Position Estimation

As discussed in § 4.6, one sensor fusion method uses a Kalman filter. In this experiment, the acceleration from the IMU and the wheel encoder-derived positions are used as measurements of the Kalman filter. Thus, sensor fusion occurs within the framework of the Kalman filter itself. The computational flow is given in Table 4.5.

Table 4.5: Computational flow for Position Estimation By Sensor Fusion Using IMU and Wheel Encoders in KF framework.

Matrix Initialisation: State Matrix $A = \begin{bmatrix} 1 & T & \frac{1}{2}T^2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}$, Measurement Matrix $H = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$, Process and Noise Covariance Matrices Q and R

Initialisation : Sampling time T, State Estimate $\hat{S}_0$, State Variance P_0, $z_1, \ldots, z_n$ are the odometer computed positions from Table 4.3 and acceleration measurements

for $k = 1, 2, \ldots$ n **do**

Compute $\hat{S}_k$ with A and S_{k-1}.

Compute P_k with A, P_{k-1} and Q {(4.43)}

Determine $\tilde{y}_k$, M_k and K_k with z_k, H, S_k, P_k and R {(4.44)}

Update $\hat{S}_k$ with $\tilde{y}_k$, K_k and S_k

Update $P_{k|k}$ with K_k, H and P_k {(4.45)}

end for

The simulation results with Dataset 1 are shown in Fig. 4.23. A comparison of Fig. 4.11 and Fig. 4.23 reveal that the fused estimate follows the wheel encoder-derived position measurements. This is because the acceleration values provided by the IMU are double integrated according to the kinematic model previously defined and are highly inaccurate and, hence, unreliable. Due to this, the Kalman filter gives very little importance to the IMU measurements. The results with Dataset 2 shown in Fig. 4.24 are similar to those observed with Dataset 1. The fused data depends more on the wheel measurements. The RMSE values for the Kalman filter-based pose estimation using IMU and wheel encoders are 182.4219 and 36.5536, respectively for the x- and y-position estimates with Dataset 1. The corresponding values with Dataset 2 are 38.4621 and 202.6116. These values are quite similar to those obtained when only the wheel encoder is used and agree with the conclusion that the Kalman filter has entirely ignored the IMU measurements. This brings about an important advantage of fusing information from different sensors: a particular sensor is automatically ignored if it does not provide proper information.

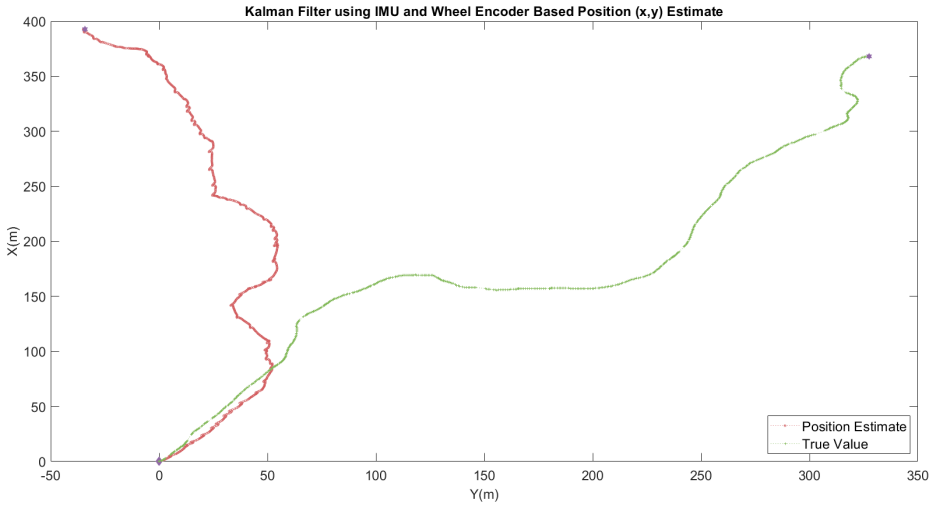


Figure 4.23: KF Based Position Estimation Using Wheel Encoder and IMU Measurements

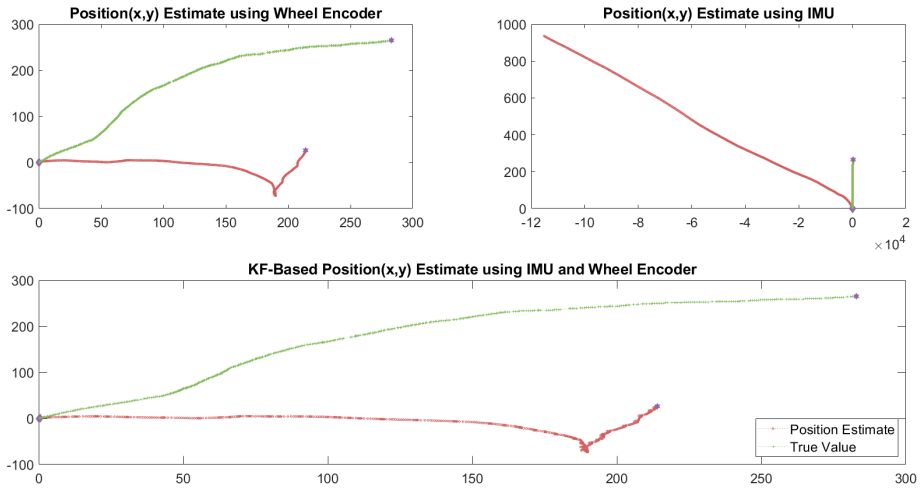


Figure 4.24: KF Based Position Estimation Using Wheel Encoder and IMU Measurements (Dataset 2)

4.7.2 Simple Fusion for Pose Computation using Sun Sensor and Wheel Encoder

In § 4.6.1, the simple fusion technique was introduced. In this section, this technique is simulated using Dataset 3. This dataset has data points at which the wheel encoder orientation accuracy is still relatively high. Thus, there is an opportunity to select between the sun sensor orientation and odometer orientation, according to the manner

of sensor fusion. For the case of dataset 1 and dataset 2, the fused readings would nearly always follow the sun sensor headings. The computational flow is provided in Table 4.6.

Table 4.6: Computational Flow for Orientation Estimation by Simple Fusion Using Sun Sensor and Wheel Encoder.

Initialisation: Block Size $BL = 25$, odometer orientation θ_{odom} from Table 4.3, sun sensor heading θ_{ss} from Table 4.4, threshold angle α_l and $\theta_{fused}(1 : BL) = \theta_{ss}(1 : BL)$.
for $k = BL + 1, BL + 2, \ldots$ n **do**
  Determine $Bias_{odom}$ and $Bias_{ss}$.
  Compare θ_{odom} with α_l.
  if $Bias_{odom} \leq \alpha_l$ **then**
    $\theta_{fused} = \theta_{odom}$
  else
    $\theta_{fused} = \theta_{ss}$
  end if {(4.42)}
  Compute x_k and y_k with $\theta_{fused}(k)$. {Table 4.3}
end for

The simple fusion method requires a threshold. Here, a threshold value of 15° is selected. The fused heading is computed in blocks of 25 data points. Thus, if the average of the bias of the previous 25 samples of the odometer orientation is less than 15°, the fused angle reading would be the odometer orientation reading. Otherwise, the sun sensor heading would be the fused heading angle. The results are shown in Fig. 4.25. This figure compares the heading angles of the wheel encoder and sun sensor individually and in combination, according to the simple fusion rule. It is observed that the simple fusion heading is more accurate than just using the measurements from the individual sensors. Finally, the position is calculated using the odometry algorithm, using the fused heading angle as the orientation angle. These results are illustrated in Fig. 4.26. The results obtained using this method are found to be quite accurate for the data subset. The RMSE values for simple fusion-based position estimation on Dataset 3 using IMU and wheel encoders are 35.4698 and 64.0565 respectively for the x- and y-position estimates. Evidently, the simple fusion method provides a better position estimate when compared to those obtained with the individual sensors. Clearly, fusing information from multiple sensors attempts to provide a better scenario than when using sensors individually.

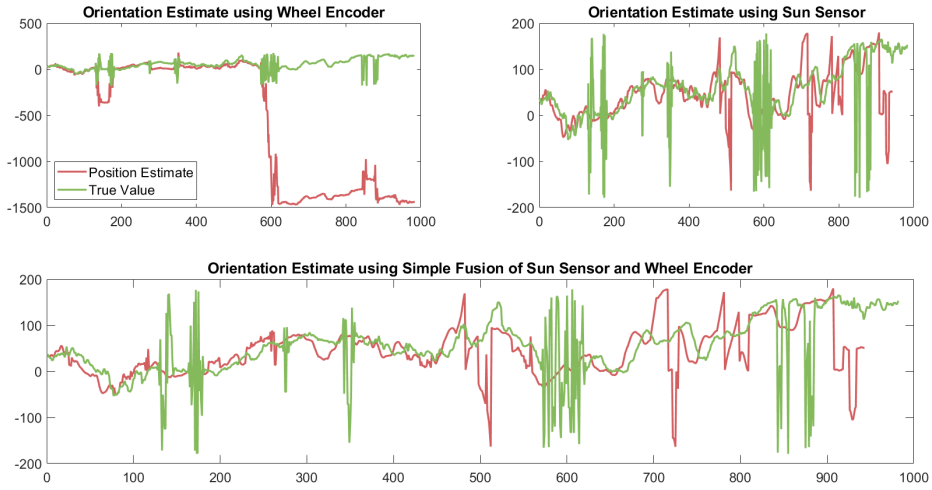


Figure 4.25: Comparison of Based on Wheel Encoder, Sun Sensor, and Using Simple Fusion of both (Dataset 3)

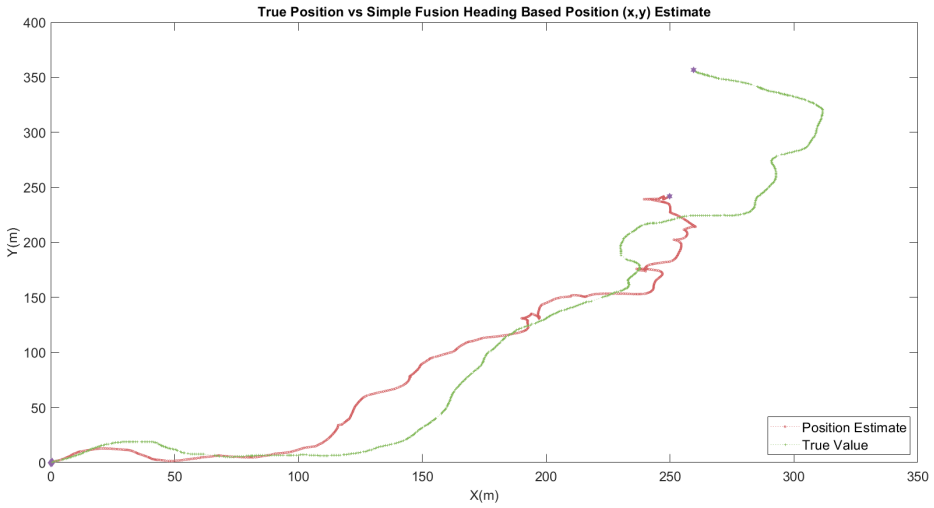


Figure 4.26: Simple Fusion of Heading Based Position (x,y) Estimation Using Wheel Encoder and Sun Sensor Measurements (Dataset 3)

4.8 Sensor Fusion Using Three Sensors

In this experiment, the heading angle estimates from different sensors are fused. The fusion method used here is the linear weighted fusion explained earlier. It consists of a linear combination of the estimates from the Sun sensor, the IMU and the wheel

encoder.

$$\theta_f = w_{odom}\theta_{odom} + w_{ss}\theta_{ss} + w_{imu}\theta_{imu} \tag{4.48}$$

where θ_{odom}, θ_{ss}, and θ_{imu} are the estimates of the angle derived using a wheel encoder, Sun sensor and IMU, respectively. The three associated weights w_{odom}, w_{ss} and w_{imu} determine the relative importance of the three estimates. The computational flow is depicted in Table 4.7.

Table 4.7: Computational flow for Pose Estimation by Linear Weighted Fusion Using Sun Sensor, IMU and Wheel Encoder.

Initialisation: Heading Angles θ_{odom}, θ_{ss} and θ_{imu}; weights w_{odom}, w_{ss} and w_{imu}
for $k = 1, 2, \ldots$ n **do**
  Compute θ_f with θ_{odom}, θ_{ss}, θ_{imu} and w_{odom}, w_{ss} and w_{imu} {(4.48)}
  Compute x_k and y_k with θ_f. {Table 4.3}
end for

In the first experiment, the three estimates are fused as follows:

$$\theta_f = 0.09\,\theta_{odom} + 0.9\,\theta_{ss} + 0.01\,\theta_{imu} \tag{4.49}$$

(The weights obtained here are as a result of several experiments to ensure that the fused estimate results in the least position estimates.) The fused estimate is then used as the heading angle to determine the rover's pose. The results are shown in Fig. 4.27. Clearly, there is an improvement in the position estimate when compared to methods dealt with before. The estimates are better than those with individual sensors or with fused data with three sensors. Thus, the errors in the estimates decrease as the number of sensors increase.

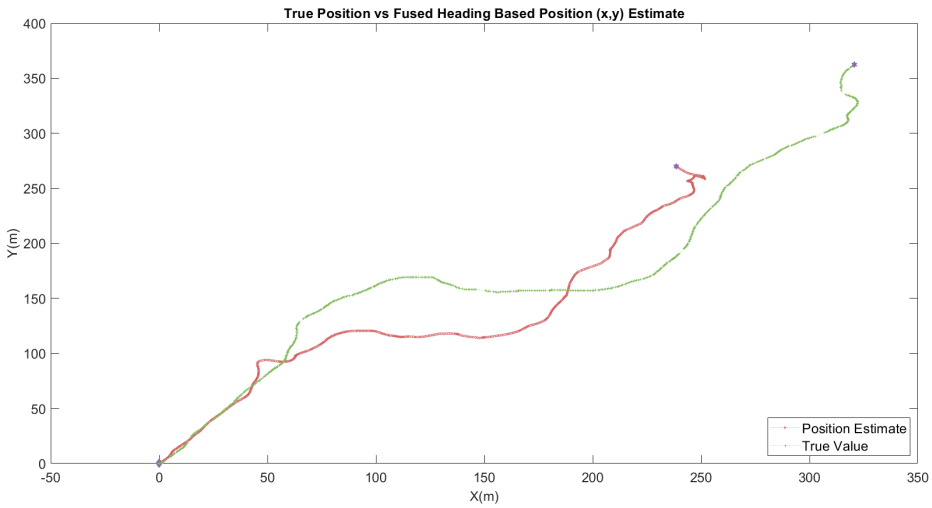


Figure 4.27: Position Estimation Based on Fused Heading Information

The choice of the weights w_{odom}, w_{ss} and w_{imu} is important, and they need to be selected carefully. The choices in (4.49) have been obtained after several experiments. To illustrate this, consider a second choice of weights as indicated in (4.50):

$$\theta_f = 0.35\,\theta_{odom} + 0.6\,\theta_{ss} + 0.05\,\theta_{imu} \tag{4.50}$$

Here, the weights corresponding to the odometer and IMU are increased by 4 to 5 times, while the weight of the sun sensor heading reading is reduced appropriately. To differentiate, the weights in (4.49) are referred to as Weights 1, and the weights in (4.50) are referred to as Weights 2. The results obtained from fusion using Weights 1 and Weights 2 are compared in Fig. 4.28 for Dataset 1. Similarly, the results with Weights 1 and Weights 2 are compared in Fig. 4.29 for Dataset 2. From these figures, it is observed that using Weights 1 yields more accurate position estimates. The RMSE values for these experiments are provided in Table 4.8. Evidently, these values are lesser than those with individual or two sensors. Thus, there is an advantage to using information from as many sensors as possible. In this case, information from IMU, Sun sensor and wheel encoder have been used. Further, the RMSE values with Weights 1 provide a better estimate when compared with those obtained with Weights 2. Therefore, determining the relative importance of the information from multiple sensors is significant.

Table 4.8: RMSE Values with Three Sensors.

Dataset	Weights	x-position	y-position
Dataset 1	Weights 1	38.2773	41.6996
	Weights 2	89.6077	19.0359
Dataset 2	Weights 1	9.2921	46.5453
	Weights 2	29.3801	100.6538

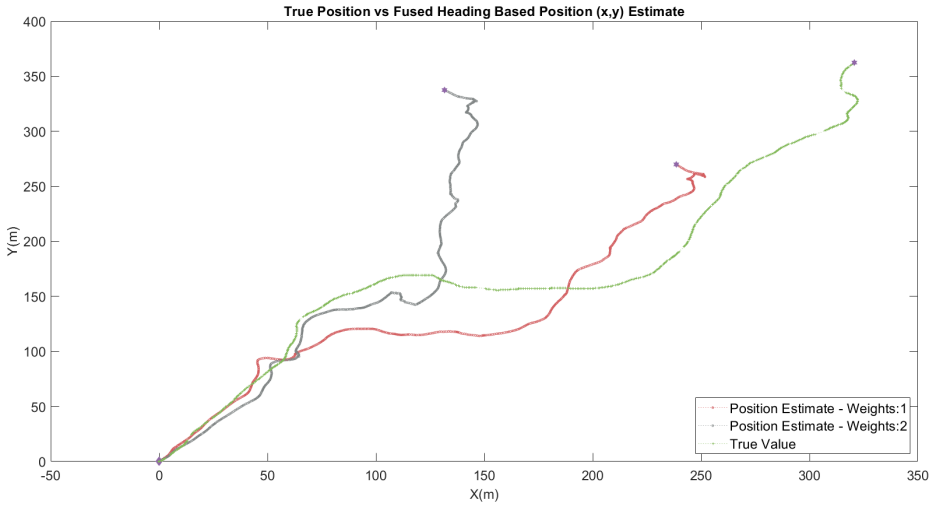


Figure 4.28: Comparison of Position Estimates using 2 Sets of Weights for Fused Heading Information on Dataset 1

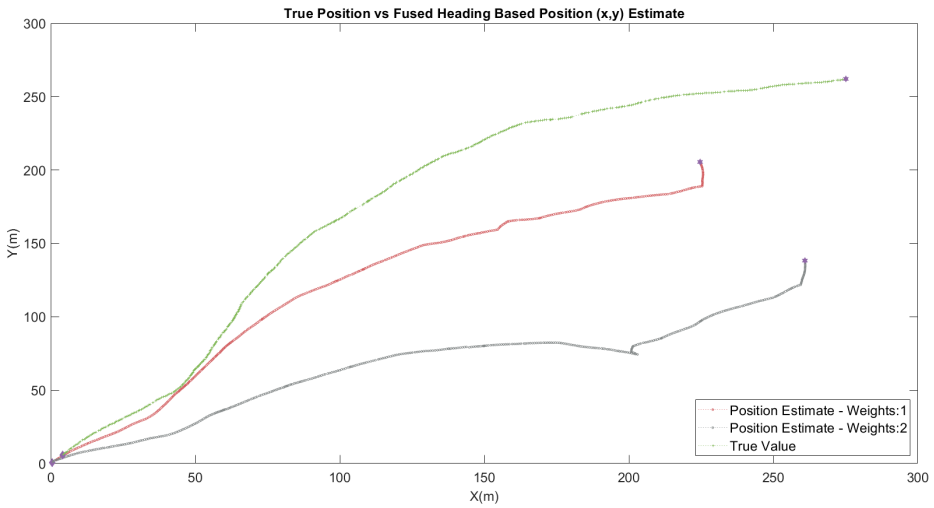


Figure 4.29: Comparison of Position Estimates using 2 Sets of Weights for Fused Heading Information on Dataset 2

4.8.1 Optimization of the Linear Weights

If the ground truth heading is available, it is possible to perform a simple least squares optimization to find optimal weights. In this section, this least squares optimization problem is formulated. Consider N measurements of the 3 heading angles, $\theta_{imu_1}, \ldots, \theta_{imu_N}$, $\theta_{we_1}, \ldots, \theta_{we_N}$, $\theta_{ss_1}, \ldots, \theta_{ss_N}$. The optimal linear weighted addition of this information should provide results that are close to the ground truth, $\theta_{true_1}, \ldots, \theta_{true_N}$. Thus, this can be formulated as the following equation

$$\begin{bmatrix} \theta_{imu_1} & \theta_{odom_1} & \theta_{ss_1} \\ \theta_{imu_2} & \theta_{odom_2} & \theta_{ss_2} \\ . & . & . \\ . & . & . \\ \theta_{imu_N} & \theta_{odom_N} & \theta_{ss_N} \end{bmatrix} \begin{bmatrix} w_{imu} \\ w_{odom} \\ w_{ss} \end{bmatrix} = \begin{bmatrix} \theta_{true_1} \\ \theta_{true_2} \\ . \\ . \\ \theta_{true_N} \end{bmatrix} \tag{4.51}$$

These matrices are henceforth called $\boldsymbol{\theta}_{meas}$, $\mathbf{W}$ and $\boldsymbol{\theta}_{true}$. Thus,

$$\boldsymbol{\theta}_{meas}\mathbf{W} = \boldsymbol{\theta}_{true} \tag{4.52}$$

To find the weights, we can find the least-squares optimal solution given by

$$\mathbf{W} = (\boldsymbol{\theta}_{meas})^{\dagger} \boldsymbol{\theta}_{true} \tag{4.53}$$

where $X^{\dagger} = \left(X^T X\right)^{-1} X^T$ is the Moore-Penrose pseudo-inverse the matrix X [72]. This can result in negative weights. Accordingly, constraints are placed on the weights, and

the above is formulated as the following constrained convex optimization problem:

$$\begin{aligned} \min_{\mathbf{W}} \quad & \|\boldsymbol{\theta}_{meas}\boldsymbol{W} - \boldsymbol{\theta}_{true}\|_2^2 \\ \text{subject to} \quad & w_{imu} + w_{odom} + w_{ss} = 1, \\ & w_{imu} \geq 0, w_{odom} \geq 0, w_{ss} \geq 0. \end{aligned}$$

The resulting weights ensures a convex combination of the measurements.

For the Dataset1, the resulting weights using MATLAB function *lsqlin* are as follows:

$$\theta_f = 0.05\,\theta_{odom} + 0.95\,\theta_{ss} + 0.00\,\theta_{imu} \tag{4.54}$$

It may be noted that this closely resembles (4.49).

4.9 Discussions

This section summarizes the results of pose estimation using the individual sensors and the sensor fusion methods. In addition to the results from the experiments that are detailed throughout this chapter, the simple fusion method was also applied to Dataset 1, in order to have a fair comparison of all the fusion methods later on in this section. The RMSE values are shown in Tables 4.9 and 4.10. While the former summarizes the values for the individual sensors, the latter does so when multi-sensor fusion is incorporated. In these tables, the sensors are wheel encoder (WE), Sun sensor (SS) and inertial measurement unit (IMU). (Whenever an experiment or result is not applicable, the corresponding entry is a dash —.) From Table 4.9, it is clear that the IMU measurements are not trustworthy. The results with WE alone are erroneous. Further, the orientation results with SS are better than those obtained using WE. The results in Table 4.10 reveal that the best option is to use all the information. The RMSE values with linear weighted (LW) fusion give the best overall estimate when compared with simple fusion (SF) or Kalman filter (KF) based fusion. When there are only two sensors, it is clear that the fusion method opts for a better estimate. This is evident when the RMSE values in Tables 4.9 and 4.10 are compared. Moreover, working with a proper dataset is essential where all the sensors are correctly calibrated. This can be observed from the RMSE values obtained with linear weighted fusion with Dataset 1 (LW-1) compared with those obtained with linear weighted fusion with Dataset 2 (LW-2). Both datasets are derived from the same Devon Island Rover dataset. However, different portions yield different results. This may be attributed to sensor calibration.

The results obtained using the different sensor fusion methods are analyzed next from another perspective, by evaluating the largest error present in the estimated position, over the entire data. These are tabulated in Table 4.11. From this table, it is

Table 4.9: RMSE Values for Pose from Individual Sensors

Dataset	Estimate	IMU	WE	SS
Dataset 1	x	79639.46	182.51	—
	y	897.05	36.54	—
	Orientation	—	65.04	50.71
Dataset 2	x	52504.50	38.64	—
	y	264.42	202.67	—
	Orientation	—	51.78	22.28

Table 4.10: RMSE Values for All Fusion Methods

Method	Sensors	Dataset 1		Dataset 2		Dataset 3	
		x	y	x	y	x	y
SF	WE, SS	35.4512	63.0093	—	—	35.4698	64.0565
KF	WE, IMU	182.4219	36.5536	38.4621	202.6116	—	—
LW-1	WE, SS, IMU	42.5998	47.2395	14.9213	51.1437	—	—
LW-2		93.3597	22.8366	24.2155	105.1069	—	—

observed that the maximum error in position in the x and y directions for the Linear Weighted Fusion method, on both datasets using the more optimal weights is between 49m to 92m. This is almost 60% of the maximum error reported in [45]. Finally, even using the simple fusion method, the maximum error reported for Dataset 3 was found

Table 4.11: Maximum Position Error for All Fusion Methods

Method	Sensors	Dataset 1		Dataset 2		Dataset 3	
		x	y	x	y	x	y
SF	WE, SS	63.5752	123.7937	—	—	58.9095	119.4314
KF	WE, IMU	362.3356	63.3664	69.3892	302.1480	—	—
LW-1	WE, SS, IMU	81.7866	92.3172	49.8294	72.2903	—	—
LW-2		188.6931	49.3560	43.2561	154.9878	—	—

to be 119m and on Dataset 1 123m, which is still less than the 150m reported in [45]. This indicates a significant improvement in the accuracy of position estimates using simple (non-camera) based sensors, when compared to existing literature.

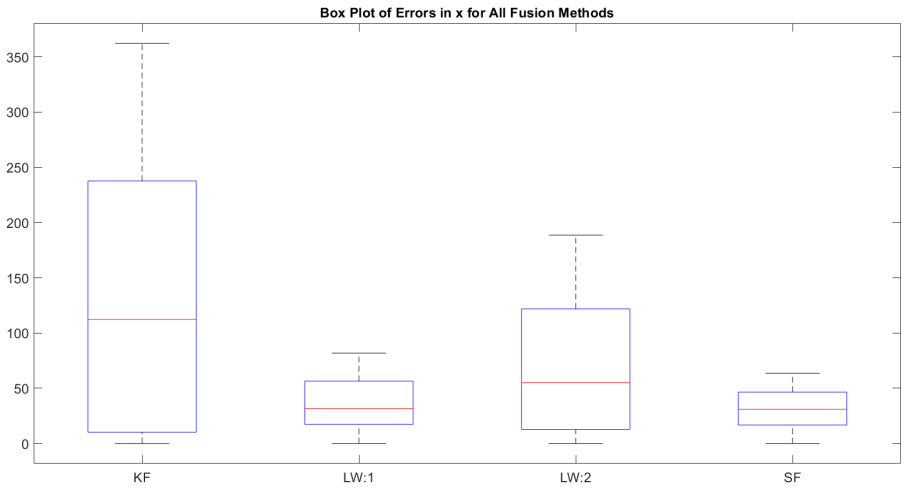


Figure 4.30: Comparison of errors in x direction for the 3 fusion methods (Dataset 1)

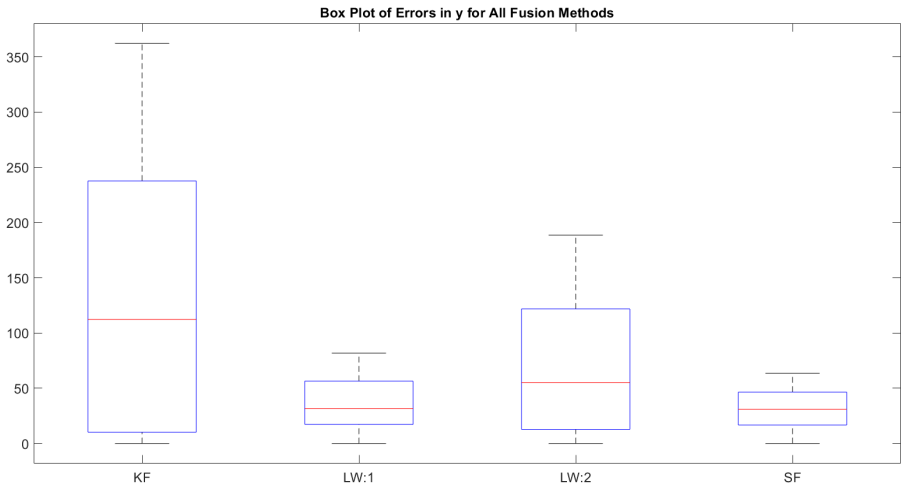


Figure 4.31: Comparison of errors in y direction for the 3 fusion methods (Dataset 1)

The box and whisker plots of the errors along the x- and y-directions are respectively shown in Fig. 4.30 and Fig. 4.31. With two sensors (namely, KF and SF), the interquartile range for SF is much lower than KF. It may be recalled that the KF uses IMU measurements, which is quite unreliable for the current experiment. The resulting

errors are much smaller since SF relies on a wheel encoder and sun sensor. When three sensors are used to determine the fused heading angle using a convex combination of the weights, it is clear that the interquartile range is reasonably small. Further, the maximum errors are also smaller. Furthermore, LW1 results in better estimates when compared to LW2. It may be recalled that the weights corresponding to LW1 were chosen after several experiments, and they compared well with the set of weights obtained as a solution to a constrained optimization problem. The box and whisker plots corresponding to LW1 and SF are similar, and this is due to the unreliability of the IMU measurements. However, one should use all the available measurements in a general setting. Suppose the IMU measurements can be corrected with additional information, as is done in typical aerospace applications. In that case, the resulting convex combination of the measurements is expected to yield smaller errors in position estimates. While the box and whisker plots of the SF method appear better and comparable to the LW1 results, it is important to also note that the RMSE values and maximum position error values of the linear-weight method with LW1 weights are overall lower than the SF method.

A meaningful comparison of the performance of linear weighted fusion to the literature is obtained using the results provided by the authors in [43]. In this paper, the authors investigate the localization of the Yutu rover, part of the Chang'e-3 system that landed on the moon in December 2013. The rover travelled around 43m from its landing site on the first lunar day. Using the base localization system with dead reckoning, the authors achieved an accuracy of around 14%. Using visual odometry-based correction resulted in the error dropping to around 5%. In order to compare these results with those presented in this thesis, waypoints are selected from Dataset 1 such that they result in a path around 65m. The ground truth and the position estimates using the linear weighted fusion method with LW1 weights are illustrated in Fig. 4.32.

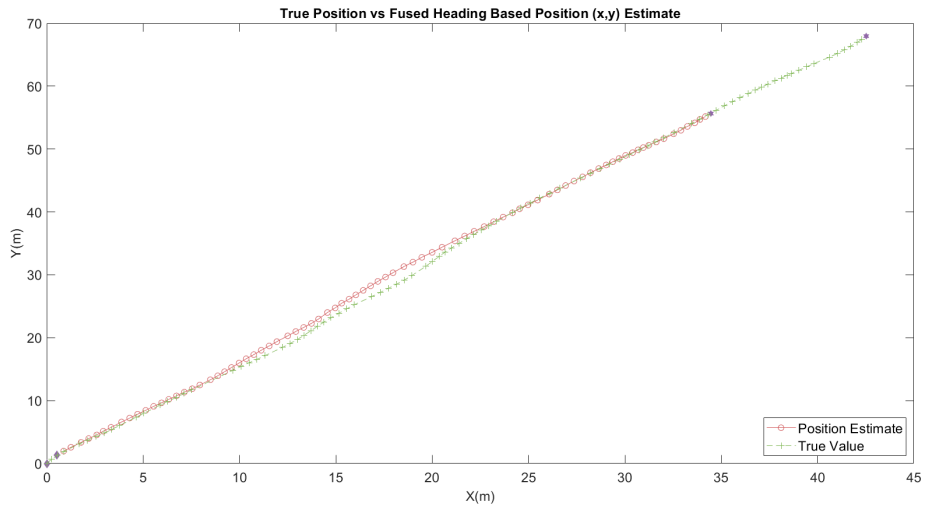


Figure 4.32: Position Estimation Using the Linear Weighted Fusion Method with LW1 Weights for a Path Length Similar to the Yutu Rover

The resulting RMSE values in the x- and y-directions, respectively, are 5.4129 and 8.1634. Further, the maximum position errors in the x- and y-directions, respectively, are 8.1681m and 12.3118m. These errors are comparable to those obtained with the Yutu rover that the authors deemed suitable for the moon mission. The results are significant as the linear weighted fusion method examined in this thesis, without using vision-based localization, can produce results comparable to those with visual odometry.

Finally, the authors in [46] also state without proof that the radius of the planetary body affects the accuracy of localization. In other words, a smaller radius results in a more accurate estimate. Since the work in this thesis makes use of Earth's measurements, it is possible that in operations on the moon, which has a radius less than one-thirds that of the Earth, the accuracy achieved using the methods in this thesis is much higher.

5 Conclusions and Future Work

A fundamental problem in autonomous robotics is that of robot navigation. This enables it to move through environments which may be hostile and uncertain. Before it can navigate, a robot must know where it is. This thesis focused on the problem of robot localization. A robot with a higher payload can carry as many sensors as possible and has sufficient onboard computing power to implement sophisticated algorithms. However, a lunar-like environment is considered here, which poses many constraints and challenges. In such environments, one cannot use GPS sensors due to the lack of sufficient satellites to triangulate. In addition, payload constraints prevent the use of cameras for localization problems. Accordingly, in this thesis, only three sensors are used for this problem. The three sensors are the wheel encoder (WE), Sun sensor (SS) and inertial measurement unit (IMU).

The method of extracting position and orientation information using each of the three sensors is presented. In IMU, the acceleration information along the different axes is integrated to extract the position along that direction. This is prone to inaccuracies as the errors accumulate and grow with time. The WE works on counting the number of times the wheel rotates per unit time. Essentially, the velocities of the two wheels are integrated with the orientation to determine the positions. Similar to the IMU, the errors do accumulate. However, they are less than IMU as the WE data is integrated only once. The Sun sensor provides the orientation of the rover. The computations are somewhat involved.

The position estimates using each of the sensors are obtained using each of the three sensors. The IMU results in substantial errors in the position estimates relative to the other two sensors. Wheel encoders provide estimates with errors. Evidently, it is difficult to trust the results when each of the sensors works individually.

Accordingly, the possibility of fusing the information from multiple sensors is investigated. Three methods for fusion are tested. The simplest method is to choose the better of two sensors when the orientation estimates are compared to a pre-defined threshold value. The second method is to use a Kalman filter to fuse the information provided by two sensors. Both methods were effective in choosing the better of the two estimates. In the case of the former, the better one could be either of the estimates for the considered dataset. The KF completely rejected the IMU measurements for a simi-

lar dataset due to the significant inaccuracies. The weighted fusion technique is a simple and effective method of fusing the information of two or more sensors. Essentially, a convex combination of the sensor information is utilized. Thus, unlike other techniques, the information from the three sensors can be provided with appropriate importance. For the considered dataset, the estimates are reasonable even though there are localization errors. These localization errors were quantified using RMSE, as well as the largest position error.The results obtained using the simple fusion and linear weighted fusion method (using more optimal weights) depicted maximum positioning errors that were much smaller, by almost 40%, when compared to the the errors obtained in literature. At present, the literature reports errors as large as 150 m, as mentioned in [45]. The linear weighted fusion method also produced results with accuracies comparable to the Yutu Rover [43] even without the use of visual odometry. Additionally, the onboard computation of the information from the sensors is considered. It is noted in § 2.3.3 that the computational resources available on other rovers were capable of extracting information from the SS, WE and IMU sensors. In addition, they were able to handle computationally intensive sensors and algorithms. In this thesis, the only additional required computations are three multiplications and two additions to compute the convex combination of sensors. This brings out the significance of these results as they are more accurate when compared to literature for the same class of sensors, but are computationally low-cost.

The principal challenge in this thesis is the availability of a proper dataset to test the different methods. At present, the datasets are derived from the Devon Island Rover dataset. It was the only dataset that involved the three sensors of interest. However, different portions of the datasets lead to different and inconsistent errors. While for one portion of the dataset, the errors in the x-direction are larger than those along the y-direction, the opposite is true for another portion of the dataset. It appears there are calibration errors. In addition, the IMU measurements and the wheel encoder measurements could not be verified.

5.1 Future Work

Several challenges arose during this thesis. The lack of a verified and accurate dataset is one of the challenges where the data is assuredly calibrated sensors. At present, the dataset is from a rover travelling on Earth. The limitations and constraints of the different methods are better analyzed from the data collected from the Lunar Zebro.

The IMU is a popular sensor that is used abundantly in pose estimation. However,

it has to be often recalibrated to the accumulation of errors to obtain meaningful data. An analysis of the different methods to achieve this is to be carried out to determine the process that is best suited for the considered lunar rover. Currently, the IMU measurements do not carry any role in rover localization in this thesis. This can be changed with improved IMU measurements.

There are errors specific to wheel encoders, such as wheel slippage. In order to improve the results, these errors must be adequately accounted for. Further, as mentioned earlier, the Lunar Zebro has unique C-type wheels. This will require slight modifications on the odometry calculations. The additional constraints posed by this error must be considered to improve the accuracy. Optimizing the obtained heading angle by testing for multiple window lengths of the measurement and predicted vectors may provide a better result.

Research through multiple tests can be conducted to characterize the threshold angle in the simple fusion method. Furthermore, using an adaptive solution, without a reference signal or ground-truth, for the sets of weights must be investigated as they may provide a more accurate result in the linear weighted fusion method.

Finally, more filters and methods to fuse the information from the three sensors can be investigated. Along with this, methods to recalibrate the pose estimates from dead reckoning methods would provide more accurate results.

Bibliography

[1] *Moon Mission (Lunar Zebro)*. https://zebro.space/missions/moon_mission/.

[2] L.J. Cox. "Blanche: Position Estimation For An Autonomous Robot Vehicle". In: IEEE, pp. 432–439. DOI: 10.1109/IROS.1989.637941. URL: http://ieeexplore.ieee.org/document/637941/.

[3] J.J. Leonard and H.F. Durrant-Whyte. "Mobile robot localization by tracking geometric beacons". In: *IEEE Transactions on Robotics and Automation* 7 (3 June 1991), pp. 376–382. ISSN: 1042296X. DOI: 10.1109/70.88147. URL: http://ieeexplore.ieee.org/document/88147/.

[4] R. Luo and B. Hong. "Coevolution based adaptive Monte Carlo localization (CEAMCL)". In: *International Journal of Advanced Robotic Systems* 1 (3 2004), p. 19. DOI: 10.5772/5634.

[5] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. "Localization strategies for autonomous mobile robots: A review". In: *Journal of King Saud University - Computer and Information Sciences* 34.8, Part B (2022), pp. 6019–6039. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2021.02.015. URL: https://www.sciencedirect.com/science/article/pii/S1319157821000550.

[6] Rudy Negenborn. "Robot localization and Kalman filters". In: *Utrecht Univ., Utrecht, Netherlands, Master's thesis INF/SCR-0309* (2003).

[7] J. Borenstein et al. "Mobile robot positioning: Sensors and techniques". In: *Journal of Robotic Systems* 14 (4 Apr. 1997), pp. 231–249. ISSN: 0741-2223. DOI: 10.1002/(SICI)1097-4563(199704)14:4<231::AID-ROB2>3.0.CO;2-R. URL: https://onlinelibrary.wiley.com/doi/10.1002/(SICI)1097-4563(199704)14:4<231::AID-ROB2>3.0.CO;2-R.

[8] Liqiang Feng. "Measurement and correction of systematic odometry errors in mobile robots". In: *IEEE Transactions on Robotics and Automation* 12.6 (1996), pp. 869–880.

[9] J. Hackett and M. Shah. "Multi-sensor fusion: a perspective". In: *IEEE International Conference on Robotics and Automation*. Cincinnati, OH, USA, 1990. DOI: 10.1109/robot.1990.126184.

[10] S. Laal et al. “Feasibility of tracking human kinematics with simultaneous localization and mapping (slam)”. In: *Sensors* 22 (23 2022), p. 9378. DOI: 10.3390/s22239378.

[11] V. Sezer and Z. Yengin. “A novel data association technique to improve run-time efficiency of slam algorithms”. In: *Anadolu University Journal of Science and Technology-a Applied Sciences and Engineering* (2019). DOI: 10.18038/aubtda.487629.

[12] A. Kuleshov et al. “Machine learning in appearance-based robot self-localization”. In: (2017). DOI: 10.1109/icmla.2017.0-171.

[13] L. Klein et al. “A machine learning approach to robot localization using fiducial markers in robotatfactory 4.0 competition”. In: *Sensors* 23 (6 2023), p. 3128. DOI: 10.3390/s23063128.

[14] P. Filho et al. “A new strategy for mobile robots localization based on omnidirectional sonar images and machine learning”. In: (2019). DOI: 10.5753/sibgrapi.est.2019.8321.

[15] C. Li et al. “Cloud computing based localization for mobile robot systems”. In: (2014). DOI: 10.1109/cacs.2014.7097194.

[16] I. Li et al. “Cloud-based improved monte carlo localization algorithm with robust orientation estimation for mobile robots”. In: *Engineering Computations* 36 (1 2018), pp. 178–203. DOI: 10.1108/ec-03-2017-0081.

[17] Luis Payá, Arturo Gil, and Óscar Reinoso. “A State-of-the-Art Review on Mapping and Localization of Mobile Robots Using Omnidirectional Vision Sensors”. In: *Journal of Sensors* 2017 (Apr. 2017), pp. 1–20. DOI: 10.1155/2017/3497650.

[18] Andrea Motroni et al. “Robot Localization via Passive UHF-RFID Technology: State-of-the-Art and Challenges”. In: *2020 IEEE International Conference on RFID (RFID)*. 2020, pp. 1–8. DOI: 10.1109/RFID49298.2020.9244884.

[19] Sergio Cebollada et al. “A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data”. In: *Expert Systems with Applications* 167 (2021), p. 114195. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.114195. URL: https://www.sciencedirect.com/science/article/pii/S095741742030926X.

[20] Pavel Pascacio et al. *Collaborative indoor positioning systems: A systematic review*. Feb. 2021. DOI: 10.3390/s21031002.

[21] Yanying Gu, Anthony Lo, and Ignas Niemegeers. “A survey of indoor positioning systems for wireless personal networks”. In: *IEEE Communications Surveys and*

Tutorials 11 (1 2009), pp. 13–32. ISSN: 1553877X. DOI: 10.1109/SURV.2009.090103.

[22] Rainer Mautz. "Indoor positioning technologies". en. Habilitation Thesis. Zurich: ETH Zurich, 2012. DOI: 10.3929/ethz-a-007313554.

[23] Anahid Basiri et al. "Indoor location based services challenges, requirements and usability of current solutions". In: *Computer Science Review* 24 (May 2017), pp. 1–12. ISSN: 15740137. DOI: 10.1016/j.cosrev.2017.03.002.

[24] Germán Martín Mendoza-Silva, Joaquín Torres-Sospedra, and Joaquín Huerta. *A meta-review of indoor positioning systems*. Oct. 2019. DOI: 10.3390/s19204507.

[25] Jiahao Huang et al. "Indoor Positioning Systems of Mobile Robots: A Review". In: *Robotics* 12 (2 Apr. 2023). ISSN: 22186581. DOI: 10.3390/robotics12020047.

[26] Yaxuan Yan et al. "Real-Time Localization and Mapping Utilizing Multi-Sensor Fusion and Visual–IMU–Wheel Odometry for Agricultural Robots in Unstructured, Dynamic and GPS-Denied Greenhouse Environments". In: *Agronomy* 12 (8 Aug. 2022). ISSN: 20734395. DOI: 10.3390/agronomy12081740.

[27] Martin Brossard, Axel Barrau, and Silvere Bonnabel. "AI-IMU Dead-Reckoning". In: *IEEE Transactions on Intelligent Vehicles* 5 (4 Dec. 2020), pp. 585–595. ISSN: 23798858. DOI: 10.1109/TIV.2020.2980758.

[28] Ming Zhang et al. "A Lightweight and Accurate Localization Algorithm Using Multiple Inertial Measurement Units". In: (Sept. 2019). URL: http://arxiv.org/abs/1909.04869.

[29] Curtis Boirum. *Improving Localization of Planetary Rovers with Absolute Bearing by Continuously Tracking the Sun*. 2015.

[30] Larry Matthies et al. "Lunar Rover Localization Using Craters as Landmarks". In: (Mar. 2022). URL: http://arxiv.org/abs/2203.10073.

[31] Shreyansh Daftry et al. "LunarNav: Crater-based Localization for Long-range Autonomous Lunar Rover Navigation". In: (Jan. 2023). URL: http://arxiv.org/abs/2301.01350.

[32] Eric E. Palmer et al. "Mercator—Independent rover localization using stereophotoclinometry and panoramic images". In: *Earth and Space Science* 3 (12 2016), pp. 488–509. ISSN: 23335084. DOI: 10.1002/2016EA000189.

[33] D.M. Helmick, Yang Cheng, and S.I. Roumeliotis. "Path following using visual odometry for a Mars rover in high-slip environments". In: vol. 2. IEEE, 2004, pp. 772–789. ISBN: 0-7803-8155-6. DOI: 10.1109/AERO.2004.1367679. URL: http://ieeexplore.ieee.org/document/1367679/.

[34] John Enright, Tim Barfoot, and Marcela Soto. "Star tracking for planetary rovers". In: 2012. ISBN: 9781457705564. DOI: 10.1109/AERO.2012.6187042.

[35] Deborah Sigel and David Wettergreen. *Star Tracker Celestial Localization System for a Lunar Rover*. 2007.

[36] J.D. Gammell et al. "Rover odometry aided by a star tracker". In: Mar. 2013, pp. 1–10. ISBN: 978-1-4673-1812-9. DOI: 10.1109/AERO.2013.6496953.

[37] Yinhu Zhan, Shaojie Chen, and Xu Zhang. "Adaptive celestial positioning for the stationary Mars rover based on a self-calibration model for the star sensor". In: *Journal of Navigation* 75 (Aug. 2021), pp. 1–16. DOI: 10.1017/S0373463321000680.

[38] *Spirit*. https://robotsguide.com/robots/spirit.

[39] Rongxing Li et al. "MER Spirit rover localization: Comparison of ground image–and orbital image–based methods and science applications". In: *J. Geophys. Res* 116 (Feb. 2011), E00F16–E00F16. DOI: 10.1029/2010JE003773.

[40] K.S. Ali et al. "Attitude and position estimation on the Mars exploration rovers". In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. Vol. 1. 2005, 20–27 Vol. 1. DOI: 10.1109/ICSMC.2005.1571116.

[41] *Curiosity*. https://robotsguide.com/robots/curiosity.

[42] Weishu Gong. "Discussions on localization capabilities of MSL and MER rovers". In: *Annals of GIS* 21.1 (2015), pp. 69–79. DOI: 10.1080/19475683.2014.992367. eprint: https://doi.org/10.1080/19475683.2014.992367. URL: https://doi.org/10.1080/19475683.2014.992367.

[43] Minglei Li et al. "Stereo Vision Technologies for China's Lunar Rover Exploration Mission". In: *International Journal of Robotics and Automation* 31 (Mar. 2016), pp. 128–136. DOI: 10.2316/Journal.206.2016.2.206-4440.

[44] R. Volpe. "Mars rover navigation results using sun sensor heading determination". In: *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*. Vol. 1. 1999, 460–467 vol.1. DOI: 10.1109/IROS.1999.813047.

[45] Y. Kuroda et al. "Accurate localization in combination with planet observation and dead reckoning for lunar rover". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. Vol. 2. 2004, 2092–2097 Vol.2. DOI: 10.1109/ROBOT.2004.1308132.

[46] John Enright, Paul Furgale, and Tim Barfoot. "Sun sensing for planetary rover navigation". In: *2009 IEEE Aerospace conference*. 2009, pp. 1–12. DOI: 10.1109/AERO.2009.4839311.

[47] Muhammad Ilyas et al. "Absolute Navigation Information Estimation for Micro Planetary Rovers". In: *International Journal of Advanced Robotic Systems* 13.2 (2016), p. 42. DOI: 10.5772/62250. eprint: https://doi.org/10.5772/62250. URL: https://doi.org/10.5772/62250.

[48] Paul Furgale, John Enright, and Timothy Barfoot. "Sun Sensor Navigation for Planetary Rovers: Theory and Field Testing". In: *IEEE Transactions on Aerospace and Electronic Systems* 47.3 (2011), pp. 1631–1647. DOI: 10.1109/TAES.2011.5937255.

[49] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[50] Ivan A. Calle Flores. "Improved Odometry Motion Models for Differential-like Mobile Robots". In: *2020 IEEE ANDESCON*. 2020, pp. 1–6. DOI: 10.1109/ANDESCON50619.2020.9272088.

[51] *Sun Calculator*. https://gml.noaa.gov/grad/solcalc/azel.html.

[52] Jean H. Meeus. *Astronomical Algorithms*. Willmann-Bell, Incorporated, 1991. ISBN: 0943396352.

[53] Paul Furgale et al. "The Devon Island rover navigation dataset". In: *The International Journal of Robotics Research* 31.6 (2012), pp. 707–713. DOI: 10.1177/0278364911433135. eprint: https://doi.org/10.1177/0278364911433135. URL: https://doi.org/10.1177/0278364911433135.

[54] *Waypoint Trajectory*. https://nl.mathworks.com/help/nav/ref/waypointtrajectory-system-object.html.

[55] Yaakov Bar-Shalom, Thiagalingam Kirubarajan, and X.-Rong Li. *Estimation with Applications to Tracking and Navigation*. USA: John Wiley Sons, Inc., 2002. ISBN: 0471221279.

[56] Behrooz Lotfi and Loulin Huang. "An approach for velocity and position estimation through acceleration measurements". In: *Measurement* 90 (2016), pp. 242–249. ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2016.04.011. URL: https://www.sciencedirect.com/science/article/pii/S0263224116300689.

[57] Satish R. Jondhale and Rajkumar S. Deshpande. "Tracking Target with Constant Acceleration Motion Using Kalman Filtering". In: *2018 International Conference*

On Advances in Communication and Computing Technology (ICACCT). 2018, pp. 581–587. DOI: 10.1109/ICACCT.2018.8529628.

[58] Udita Bhattacherjee et al. "Experimental Study of Outdoor UAV Localization and Tracking using Passive RF Sensing". In: *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental Evaluation & CHaracterization*. WiNTECH '21. New Orleans, LA, USA: Association for Computing Machinery, 2021, 31–38. ISBN: 9781450387033. DOI: 10.1145/3477086.3480832. URL: https://doi.org/10.1145/3477086.3480832.

[59] Mingyang Li et al. "A Lightweight and Accurate Localization Algorithm Using Multiple Inertial Measurement Units". In: *IEEE Robotics and Automation Letters* 5 (Jan. 2020), pp. 1–1. DOI: 10.1109/LRA.2020.2969146.

[60] Lei Cheng et al. "Positioning and navigation of mobile robot with asynchronous fusion of binocular vision system and inertial navigation system". In: *International Journal of Advanced Robotic Systems* 14.6 (2017), p. 1729881417745607. DOI: 10.1177/1729881417745607. eprint: https://doi.org/10.1177/1729881417745607. URL: https://doi.org/10.1177/1729881417745607.

[61] Manon Kok, Jeroen Hol, and Thomas Schön. *Using Inertial Sensors for Position and Orientation Estimation*. Jan. 2017. ISBN: 9781680833577. DOI: 10.1561/9781680833577.

[62] Yuming Yin et al. "Sensor Fusion of GNSS and IMU Data for Robust Localization via Smoothed Error State Kalman Filter". In: *Sensors* 23.7 (2023). ISSN: 1424-8220. DOI: 10.3390/s23073676. URL: https://www.mdpi.com/1424-8220/23/7/3676.

[63] Xiaoli Meng, Heng Wang, and Bingbing Liu. "A Robust Vehicle Localization Approach Based on GNSS/IMU/DMI/LiDAR Sensor Fusion for Autonomous Vehicles". In: *Sensors* 17.9 (2017). ISSN: 1424-8220. DOI: 10.3390/s17092140. URL: https://www.mdpi.com/1424-8220/17/9/2140.

[64] Garrett Hemann, Sanjiv Singh, and Michael Kaess. "Long-range GPS-denied aerial inertial navigation with LIDAR localization". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1659–1666. DOI: 10.1109/IROS.2016.7759267.

[65] *"Wheeled Locomotion"*. https://ecs-pw-facweb.ecs.csus.edu/~fbelkhou/Lect2E187.pdf.

[66] Andrew Lambert et al. "Visual odometry aided by a sun sensor and inclinometer". In: Apr. 2011, pp. 1 –14. DOI: 10.1109/AERO.2011.5747268.

[67] Grace Wahba. “A Least Squares Estimate of Satellite Attitude”. In: *SIAM Review* 7.3 (1965), pp. 409–409. DOI: 10.1137/1007077. eprint: https://doi.org/10.1137/1007077. URL: https://doi.org/10.1137/1007077.

[68] P.B. Davenport. *A Vector Approach to the Algebra of Rotations with Applications*. NASA technical note. National Aeronautics and Space Administration, 1968. URL: https://books.google.nl/books?id=oBUssxoijvcC.

[69] Makoto Ohki, Michael E. Zervakis, and Anastasios N. Venetsanopoulos. “3-D Digital Filters”. In: *Multidimensional Systems: Signal Processing and Modeling Techniques*. Ed. by C.T. Leondes. Vol. 69. Control and Dynamic Systems. Academic Press, 1995, pp. 49–88. DOI: https://doi.org/10.1016/S0090-5267(05)80038-6. URL: https://www.sciencedirect.com/science/article/pii/S0090526705800386.

[70] Praneel Chand. “Integrating an electronic compass for position tracking on a wheeled tricycle mobile robot”. In: *Drone Systems and Applications* 10.1 (2022), pp. 179–199. DOI: 10.1139/dsa-2021-0049. eprint: https://doi.org/10.1139/dsa-2021-0049. URL: https://doi.org/10.1139/dsa-2021-0049.

[71] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*. Tech. rep. 95-041. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 1995. URL: http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html.

[72] Michel Verhaegen and Vincent Verdult. *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, 2007.

Davenport's q-Method

A

In 1965, Grace Wahba formulated the problem of finding a rotation between two coordinate systems. Given a set of N vector measurements in the body coordinate system, an optimal attitude matrix $\mathbf{A}$ would minimize the following loss function:

$$L(A) = \frac{1}{2}\sum_{i=1}^{N} w_i \left\| u_i - \mathbf{A}v_i \right\|^2, \tag{A.1}$$

where u_i is the ith vector measurement in the body frame, v_i is the ith vector in the reference frame, w_is are a set of N scalar weights for each observation, N is the number of measurements, and $\|\cdot\|$ represents the Euclidean norm.

Davenport proposed a solution in 1968 that solves Wahba's problem, yielding a unique optimal solution. The corresponding gain function is defined as:

$$g(\mathbf{A}) = 1 - L(\mathbf{A}) = \sum_{i=1}^{N} w_i \mathbf{U}^T \mathbf{A}\mathbf{V}. \tag{A.2}$$

This gain function $g(\mathbf{A})$ is maximum when the loss function $L(\mathbf{A})$ is at a minimum. Thus, the aim is to find the optimal attitude matrix $\mathbf{A}$ which maximizes $g(A)$.

$$\begin{aligned} g(\mathbf{A}) &= \sum_{i=1}^{N} w_i \,\mathrm{trace}(\mathbf{U}_i^T \mathbf{A}\mathbf{V}_i) \\ &= \mathrm{trace}(\mathbf{A}\mathbf{B}^T), \end{aligned} \tag{A.3}$$

where $\mathbf{B}$ is the attitude profile matrix

$$\mathbf{B} = \sum_{i=1}^{N} w_i \mathbf{U}\mathbf{V}. \tag{A.4}$$

The attitude matrix is parametrized in terms of quaternion $\mathbf{q}$:

$$\mathbf{A}(\mathbf{q}) = (q_w^2 - \mathbf{q}_v^T \mathbf{q}_v)\mathbf{1} + 2\mathbf{q}_v\mathbf{q}_v^T - 2q_w\mathbf{q}_v^{\times}. \tag{A.5}$$

Here, the $\mathbf{q}_v^{\times}$ term is the skew-symmetric matrix that is formed from the components of the vector $\mathbf{q}_v$.

$$\mathbf{q}_v^{\times} = \begin{bmatrix} 0 & -q_{v3} & q_{v2} \\ q_{v3} & 0 & -q_{v1} \\ -q_{v2} & q_{v1} & 0 \end{bmatrix} \tag{A.6}$$

The gain function, in terms of quaternions, becomes

$$\mathbf{g(q)} = (q_w^2 - \mathbf{q}_v^T\mathbf{q}_v)\,\text{trace}\left(\mathbf{B}^T\right) + 2\,\text{trace}\left(\mathbf{q}_v\mathbf{q}_v^T\mathbf{B}^T\right) + 2q_w\,\text{trace}\left(\mathbf{q}_v^{\text{x}}\mathbf{B}^T\right). \tag{A.7}$$

This can be simplified to a bilinear relationship of the form:

$$\mathbf{g(q)} = \mathbf{q}^T\mathbf{K}\mathbf{q}, \tag{A.8}$$

where the 4×4 matrix $\mathbf{K}$ is constructed as:

$$\mathbf{K} = \begin{bmatrix} \sigma & \mathbf{Z}^T \\ \mathbf{Z} & \mathbf{Q}\ \mathbf{-1}\sigma \end{bmatrix} \tag{A.9}$$

where $\mathbf{1}$ is 3×3 matrix of ones. The following are the intermediate values that are used:

$$\sigma = \text{trace}(\mathbf{B}) \tag{A.10}$$

$$\mathbf{Q} = \mathbf{B} + \mathbf{B}^T \tag{A.11}$$

The cyclic components of $\mathbf{B}$ are extracted next. Let B_{ij} be the element of $\mathbf{B}$ in row i and column j. Thus,

$$\mathbf{Z} = \begin{bmatrix} B_{23} - B_{32} & B_{31} - B_{13} & B_{12} - B_{21} \end{bmatrix} \tag{A.12}$$

The optimal quaternion $\hat{\mathbf{q}}$, which parametrizes the optimal attitude matrix, is an eigenvector of $\mathbf{K}$. Using Lagrange multipliers, $g(\mathbf{q})$ is maximum if the eigenvector corresponding to the largest eigenvalue λ is chosen. Thus,

$$\mathbf{K}\hat{\mathbf{q}} = \lambda\hat{\mathbf{q}} \tag{A.13}$$

Code

B

```
%% Retrieving the Data
clear all; clc;

%Setting plot properties
markstart = "diamond"; markend = "hexagram"; markerest = ":o";
    marktruth = ":+";

blue = [114 147 203]./255;
red = [211 94 96]./255;
black = [128 133 133]./255;
green = [132 186 91]./255;
purple = [144 103 167]./255;

%Processing the Data
[t,ss_az,ss_zen,gps_x,gps_y,gps_z,gps_lat,gps_long] = readvars("
    data_final_1.csv");
gps_lon = -gps_long;
gps_lon_nonmod = gps_long;

[h,m,s] = hms(t);
times = h.*3600 + m.*60 + s;
times = times - times(1)*ones(length(times),1);

%Setting the waypoints
waypoints = [];

for i = 1:length(gps_x)
    waypoints = [waypoints; gps_x(i)-gps_x(1) gps_y(i)-gps_y(1)
        gps_z(i)-gps_z(1)];
end
```

```
figure;
plot(waypoints(:,1),waypoints(:,2),'LineWidth',2,'Color',blue)
set(gca,'FontSize',14)
xlabel('X(m)','FontSize',14)
ylabel('Y(m)','FontSize',14)
title('Rover path using GPS Coordinates as Waypoints','FontSize'
    ,14)

% SENSORS

imuFs = 1;
encoderFs = 1;

%Getting WayPoint Trajectory to simulate vehicle
groundTruth = waypointTrajectory('SampleRate', imuFs, ...
    'Waypoints', waypoints, ...
    'TimeOfArrival', times);

%IMU Sensor
imu = imuSensor('SampleRate',groundTruth.SampleRate,'
    ReferenceFrame','ENU');

%Differential Drive Wheel Encoder Sensor : LZ works on
    Differential Drive
encoder = wheelEncoderDifferentialDrive('SampleRate',groundTruth
    .SampleRate);

[position,orientation,velocity,acceleration,angularVelocity] =
    lookupPose(groundTruth,times);

orientation_Euler = eulerd(orientation,"ZYX","frame"); %in
    degrees
orient_Euler = (quat2eul(orientation)); %in radians

[accel,gyro] = imu(acceleration,angularVelocity,orientation);
ticks = encoder(velocity, angularVelocity, orientation);
```

```
orient_truth = orientation_Euler(:,1);

%% SUN SENSOR

leng = length(h);

Julian_Day = zeros(leng,1);
gasts = zeros(leng,1);
azimuths_I = zeros(leng,1);
zeniths_I = zeros(leng,1);
s_f_corr = zeros(3,leng);

for i = 1: leng
    Julian_Day(i) = JD_calc(h(i),m(i),s(i));
end

for i = 1:leng
    gasts(i) = gast_angle(Julian_Day(i));
end

for i = 1:leng
    [azimuths_I(i),zeniths_I(i)] = calc_eph(Julian_Day(i), gasts
       (i), gps_lat(i), gps_lon(i));
end

%Transform and correct to get S_f'

for i = 1:leng

    C_if = rotz(gasts(i));
    Ssi =  sun_vector(azimuths_I(i),zeniths_I(i));

    % Ssi = [sin(deg2rad(b)) * cos(deg2rad(a)), sin(deg2rad(b))
       * sin(deg2rad(a)), cos(deg2rad(b))]';

    S_f = C_if' * Ssi; %predicted sun vector
    G_f = [cos(deg2rad(gps_lat(i))) * cos(deg2rad(gps_lon(i)));
```

```
        cos(deg2rad(gps_lat(i))) * sin(deg2rad(gps_lon(i))); sin(
        deg2rad(gps_lat(i)))];

96      %Correction of S_f

        etrue = acos(S_f' * G_f) - pi/2;
        deltae = (2.97 * 10^-4) / (tan (etrue + (0.180/(etrue +
            0.0892)) ));
        eapp = etrue + deltae;
101
        gf_skew = skew(G_f);

        h_f = -gf_skew * gf_skew* S_f / cos(etrue); %radian
        S_F_corr = sin(eapp)*h_f - cos(eapp)*G_f; %radian
106     s_f_corr(:,i) = S_F_corr;

    end

    % SUN SENSOR ALGORITHM : Calculation of Heading Angle : Window
        length 3
111
    orient_truth = orientation_Euler(:,1);
    times_cut3 = times(3:end);
    heading_angle3 = zeros(leng,1);

116 for cnt = 3:length(gps_y)
    %Construting W Vector - These are in the sun sensor frame
        us0 = sun_vector(ss_az(cnt),ss_zen(cnt));
        us1 = sun_vector(ss_az(cnt - 1),ss_zen(cnt - 1));
        us2 = sun_vector(ss_az(cnt - 2),ss_zen(cnt - 2));
121
        W = [us2 us1 us0];

        uf0 = s_f_corr(:,cnt);
        uf1 = s_f_corr(:,cnt-1);
126     uf2 = s_f_corr(:,cnt-2);
```

```
        V = [uf2 uf1 uf0];
        heading_angle3(cnt) = heading_calc(W,V,gps_lat(cnt),gps_lon(
            cnt));
    end

    heading_angle3 = heading_angle3(3:end);

    figure;
    plot(times,orient_truth,'Color',green,'LineWidth',2);
    set(gca,'FontSize',14)
    hold on;
    plot(times_cut3,heading_angle3,'Color',red,'LineWidth',2)
    xlabel('Time (s)','FontSize',14)
    ylabel('Heading Angle(deg)','FontSize',14)
    legend('Simulation Estimate','Sun Sensor Estimate','FontSize'
        ,14,'Location','southwest')
    title('Simulated vs Sun Sensor Estimate of Heading Angle','
        FontSize',14)

    %Filtering the curve
    truth_angle_mod2 = medfilt1(orientation_Euler(:,1),10);
    figure;
    plot(times,orientation_Euler(:,1), 'Color',green,'LineWidth',2)
    set(gca,'FontSize',14)
    hold on;
    plot(times,truth_angle_mod2,'Color',red,'LineWidth',2)
    hold off;
    legend("Simulated Orientation Angle","Filtered Orientation Angle
        ",'FontSize',14)
    title('Simulated vs Filtered Orientation Angle of the Simulated
        Robot ','FontSize',14)
    xlabel("Timestep")
    ylabel("Heading Angle(deg)")

    %Bias Calculation and Correction
    Windowlength = 25;
```

```
orient_truth_cut = truth_angle_mod2(15:end);
heading_mod = heading_angle3(13:length(heading_angle3) -
    Windowlength,1);
times_cutmod = times_cut3(13:length(heading_angle3) -
    Windowlength,1);


Bias = zeros(length(heading_mod),1);

for idx = 1:length(heading_mod) - Windowlength
    Blockh = heading_mod(idx:idx+Windowlength);
    Blocko = orient_truth_cut(idx:idx+Windowlength);
    Bias(idx) = mean(Blockh-Blocko);
end

heading_corrected = heading_mod - Bias;

%Correct range
for i = 1:length(heading_corrected)
    if heading_corrected(i)>180
        heading_corrected(i)= heading_corrected(i)-180;
    elseif heading_corrected(i)<-180
        heading_corrected(i)= heading_corrected(i)+180;
    end
end

figure;
plot(times,orient_truth,'Color',green,'LineWidth',2);
set(gca,'FontSize',14)
hold on;
plot(times_cutmod,heading_corrected,'Color',red,'LineWidth',2)
xlabel('Time (s)','FontSize',14)
ylabel('Heading Angle(deg)','FontSize',14)
xlabel('Time (s)','FontSize',14)
ylabel('Heading Angle(deg)','FontSize',14)
legend('Heading Angle Estimate','Bias-Corrected Estimate','
    FontSize',10,'Location','southeast')
```

```
title('Estimated and Bias-Corrected Heading Angle Obtained from
   Sun Sensor ','FontSize',12)

% RMSE
rmse_ss_only = rmse(heading_corrected,orient_truth(15:end-
   Windowlength));

%% WHEEL ENCODER

L = encoder.TrackWidth;
t = 1;

odom_x = zeros(length(ticks),1);
odom_y = zeros(length(ticks),1);
odom_orient = zeros(length(ticks),1);

odom_x(1) = waypoints(2,1);
odom_y(1) = waypoints(2,2);
odom_orient(1) = orient_Euler(2,1);

for i = 1:length(ticks)
    if ticks(i,1) == 0
        ticks(i,1) = 1;
    end

    if ticks(i,2) == 0
        ticks(i,2) = 1;
    end

    nu_l = ticks(i,1)/t * (pi/180) * (360/2048);
    nu_r = ticks(i,2)/t * (pi/180) * (360/2048);

    V_l = nu_l * encoder.WheelRadius(1);
    V_r = nu_r * encoder.WheelRadius(2);

    nu = (V_l + V_r)/2;
    omega = (V_r - V_l)/L;
```

```
        k00 = nu * cos(odom_orient(i));
        k01 = nu * sin(odom_orient(i));
        k02 = omega;

        k10 = nu * cos(odom_orient(i) + t/2 * k02);
        k11 = nu * sin(odom_orient(i) + t/2 * k02);
        k12 = omega;

        k20 = nu * cos(odom_orient(i) + t/2 * k12);
        k21 = nu * sin(odom_orient(i) + t/2 * k12);
        k22 = omega;

        k30 = nu * cos(odom_orient(i) + t * k22);
        k31 = nu * sin(odom_orient(i) + t * k22);
        k32 = omega;

        odom_x(i+1) = odom_x(i) + t/6 * (k00 + 2*(k10 +k20) + k30);
        odom_y(i+1) = odom_y(i) + t/6 * (k01 + 2*(k11 +k21) + k31);
        odom_orient(i+1) = odom_orient(i) + t/6 * (k02 + 2*(k12 +k22
           ) + k32);

    end

    odom_x = odom_x(2:end);
    odom_y = odom_y(2:end);
    odom_orient = odom_orient(2:end);

    %RMSE AND PLOTS

    rmse_enc_xposn = rmse(odom_x,waypoints(:,1));
    rmse_enc_yposn = rmse(odom_y,waypoints(:,2));
    rmse_enc_orient = rmse(odom_orient,orient_truth);


    % Plot WHEEL ENCODER
    samples = 1:length(waypoints)';
```

```
figure;
plot(odom_x(1),odom_y(1),markstart,'MarkerFaceColor',purple,'
    MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(odom_x(2:end-1),odom_y(2:end-1),markerest,'Markersize',2,'
    Color',red);
hold on;
plot(odom_x(end),odom_y(end),markend, 'MarkerFaceColor',purple,'
    MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(waypoints(1,1),waypoints(1,2),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(2:end-1,1),waypoints(2:end-1,2),marktruth,'
    Markersize',2, 'Color',green);
hold on;
plot(waypoints(end,1),waypoints(end,2),markend,'MarkerFaceColor'
    ,purple,'MarkerEdgeColor',purple)
xlabel('Y(m)','FontSize',14)
ylabel('X(m)','FontSize',14)
legend('Start','Position Estimate','Stop','','True Value','','
    FontSize',14,'Location','southeast')
title('True vs Wheel Encoder Position (x,y) Estimate','FontSize'
    ,14)
hold off;

figure;
plot(samples(1),odom_x(1),markstart,'MarkerFaceColor',purple,'
    MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(samples(2:end-1),odom_x(2:end-1),markerest,'Markersize',2,'
    Color',red)
hold on;
```

```
plot(samples(end),odom_x(end),markend, 'MarkerFaceColor',purple,
    'MarkerEdgeColor',purple)
hold on;
plot(samples(1),waypoints(1,1),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
hold on;
plot(samples(2:end-1),waypoints(2:end-1,1),marktruth,'Markersize
    ',2, 'Color',green)
hold on;
plot(samples(end),waypoints(end,1),markend,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
xlabel('Timestep','FontSize',14)
ylabel('X(m)','FontSize',14)
legend('Start','Position Estimate','Stop','','True Value','','
    FontSize',14,'Location','northwest')
title('True vs Wheel Encoder Position (x) Estimate','FontSize'
    ,14)
hold off;

figure;
plot(samples(1),odom_y(1),markstart,'MarkerFaceColor',purple,'
    MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(samples(2:end-1),odom_y(2:end-1),markerest,'Markersize',2,'
    Color',red)
hold on;
plot(samples(end),odom_y(end),markend, 'MarkerFaceColor',purple,
    'MarkerEdgeColor',purple)
hold on;
plot(samples(1),waypoints(1,2),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
hold on;
plot(samples(2:end-1),waypoints(2:end-1,2),marktruth,'Markersize
    ',2, 'Color',green)
hold on;
plot(samples(end),waypoints(end,2),markend,'MarkerFaceColor',
```

```
    purple,'MarkerEdgeColor',purple)
xlabel('Timestep','FontSize',14)
ylabel('Y(m)','FontSize',14)
legend('Start','Position Estimate','Stop','','True Value','','
    FontSize',14,'Location','southeast')
title('True vs Wheel Encoder Position (y) Estimate','FontSize'
    ,14)
hold off;

figure;
plot(odom_orient,'Color',red,'LineWidth',2);
set(gca,'FontSize',14)
hold on;
plot(orient_Euler(:,1),'Color',green,'LineWidth',2);
xlabel('Timestep','FontSize',14)
ylabel('Heading Angle(rad)','FontSize',14)
legend('Orientation Estimate','True Value','FontSize',14,'
    Location','northeast')
title('True vs Wheel Encoder Heading Angle Estimate','FontSize'
    ,14)
hold off;

%% IMU Position Only Using Kalman Filter

dt = 1;
n = length(accel);

time = 0:dt:dt*(n-1);

ax = accel(1:n,1);
ay = accel(1:n,2);
az = accel(1:n,3);

Ax = [1 dt 0.5*(dt^2)
      0 1 dt
      0 0 1];
Ay = Ax;
```

```
Az = Ax;

Cx = [0 0 1];
Cy = Cx;
Cz = Cx;
G = [0.5*(dt^2) dt 1]';
% var_a = 67.53e-06;
var_a = 1;
Q = G * G' * sqrt(var_a);
% Q = (1e-4)*eye(3);

R = eye(1);
P = 5000*eye(3);

%For x axis
thetax = zeros(3,n);
thx = [0;0.05;0];  %position,velocity,acceleration

y_mod = ax(1);

for i=1:n-1

    %Time Update
    th_currx = Ax * thx;
    P = Ax * P * Ax'+ Q;

    %Measurement Update

    K = P * Cx' /( Cx * P * Cx' + R);
    th_currx = th_currx + (K * (y_mod - Cx * th_currx));
    P = (eye(3) - (K*Cx)) * P;

    thetax(:,i) = th_currx;
    thx = th_currx;

    y_mod = ax(i+1);
end
```

```
%For y axis
P = 5000*eye(3);
thetay = zeros(3,n);
thy = [0;0.05;0];

y_mod = ay(1);

for i = 1:n-1

    %Time Update
    th_curry = Ay * thy;
    P = Ay * P * Ay'+ Q;

    %Measurement Update
    K = P * Cy' /( Cy * P * Cy' + R);
    th_curry = th_curry + (K * (y_mod - Cy * th_curry));
    P = (eye(3) - (K*Cy)) * P;

    thetay(:,i) = th_curry;
    thy = th_curry;

    y_mod = ay(i+1);
end

thetax_imu = thetax;
thetay_imu = thetay;

%RMSE AND PLOTS

rmse_imu_x = rmse(thetax_imu(1,:)',waypoints(:,1));
rmse_imu_y = rmse(thetay_imu(1,:)',waypoints(:,2));

%PLOTS

figure;
plot(thetax_imu(1,1),thetay_imu(1,1),markstart,'MarkerFaceColor'
```

```
    ,purple,'MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(thetax_imu(1,2:end-1),thetay_imu(1,2:end-1),markerest,'
    Markersize',2, 'Color',red)
hold on;
plot(thetax_imu(1,end),thetay_imu(1,end),markend,'
    MarkerFaceColor',purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(1,1),waypoints(1,2),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(2:end-1,1),waypoints(2:end-1,2),marktruth,'
    Markersize',2, 'Color',green)
hold on;
plot(waypoints(end,1),waypoints(end,2),markend,'MarkerFaceColor'
    ,purple,'MarkerEdgeColor',purple)
xlabel('Y(m)','FontSize',14)
ylabel('X(m)','FontSize',14)
legend('','Position Estimate','','','True Value','','FontSize'
    ,14,'Location','northwest')
title('KF Based Position (x,y) Estimate using only IMU
    Measurements')
hold off;

figure;
plot(samples(1),thetax_imu(1,1),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(samples(2:end-1),thetax_imu(1,2:end-1),markerest,'
    Markersize',2,'Color',red)
hold on;
plot(samples(end),thetax_imu(1,end-1),markend, 'MarkerFaceColor'
    ,purple,'MarkerEdgeColor',purple)
hold on;
plot(samples(1),waypoints(1,1),markstart,'MarkerFaceColor',
```

```
    purple,'MarkerEdgeColor',purple)
hold on;
plot(samples(2:end-1),waypoints(2:end-1,1),marktruth,'Markersize
    ',2, 'Color',green)
hold on;
plot(samples(end),waypoints(end,1),markend,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
xlabel('Timestep','FontSize',14)
ylabel('X(m)','FontSize',14)
legend('','Position Estimate','','','True Value','','FontSize'
    ,14,'Location','southwest')
title('True vs IMU Position (x) Estimate','FontSize',14)
hold off;

figure;
plot(samples(1),thetay_imu(1,1),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(samples(2:end-1),thetay_imu(1,2:end-1),markerest,'
    Markersize',2,'Color',red)
hold on;
plot(samples(end),thetay_imu(1,end-1),markend, 'MarkerFaceColor'
    ,purple,'MarkerEdgeColor',purple)
hold on;
plot(samples(1),waypoints(1,1),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
hold on;
plot(samples(2:end-1),waypoints(2:end-1,1),marktruth,'Markersize
    ',2, 'Color',green)
hold on;
plot(samples(end),waypoints(end,1),markend,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
xlabel('Timestep','FontSize',14)
ylabel('Y(m)','FontSize',14)
legend('','Position Estimate','','','True Value','','FontSize'
    ,14,'Location','northwest')
```

```
title('True vs IMU Position (y) Estimate','FontSize',14)
hold off;

%% Combine IMU AND WHEEL ENCODER

dt = 1;
n = length(accel) ;

time = 0:dt:dt*(n-1);

ax = accel(1:n,1);
ay = accel(1:n,2);
az = accel(1:n,3);

Ax = [1 dt 0.5*(dt^2)
      0 1 dt
      0 0 1];
Ay = Ax;
Az = Ax;

Cx = [1 0 0; 0 0 1];
Cy = Cx;
Cz = Cx;
G = [0.5*(dt^2) dt 1]';
% var_a = 67.53e-06;
var_a = 1;
Q = G * G' * sqrt(var_a);
% Q = (1e-4)*eye(3);

R = eye(2);
P = 5000*eye(3);

%For x axis

thetax = zeros(3,n);
thx = [-10;0.05;-0.0001];  %position,velocity,acceleration
```

```
y_mod = [odom_x(1); ax(1)];

for i=1:n-1

    %Time Update
    th_currx = Ax * thx;
    P = Ax * P * Ax'+ Q;

    %Measurement Update

    K = P * Cx' /( Cx * P * Cx' + R);
    th_currx = th_currx + (K * (y_mod - Cx * th_currx));
    P = (eye(3) - (K*Cx)) * P;

    thetax(:,i) = th_currx;
    thx = th_currx;

    y_mod = [odom_x(i+1);ax(i+1)];
end

%For y axis
P = 5000*eye(3);

thetay = zeros(3,n);
thy = [-0.2;-0.02;0.00005];

y_mod = [odom_y(1);ay(1)];

for i = 1:n-1

    %Time Update
    th_curry = Ay * thy;
    P = Ay * P * Ay'+ Q;

    %Measurement Update

    K = P * Cy' /( Cy * P * Cy' + R);
```

```
        th_curry = th_curry + (K * (y_mod - Cy * th_curry));
        P = (eye(3) - (K*Cy)) * P;

        thetay(:,i) = th_curry;
        thy = th_curry;

        y_mod = [odom_y(i+1);ay(i+1)];
    end

    %RMSE VALUES AND MAXIMUM POSITION ERROR

    rmse_imu_enc_kf_x = rmse(thetax(1,1:end-1)',waypoints(1:end-1,1)
       );
    max_imu_enc_kf_x = max(abs(thetax(1,1:end-1)' - waypoints(1:end
       -1,1) ));

    rmse_imu_enc_kf_y = rmse(thetay(1,1:end-1)',waypoints(1:end-1,2)
       );
    max_imu_enc_kf_y = max(abs(thetay(1,1:end-1)' - waypoints(1:end
       -1,2) ));

    figure;
    plot(thetax(1,1),thetay(1,1),markstart,'MarkerFaceColor',purple,
       'MarkerEdgeColor',purple)
    set(gca,'FontSize',14)
    hold on;
    plot(thetax(1,2:end-2),thetay(1,2:end-2),markerest, 'Markersize'
       ,2, 'Color',red)
    hold on;
    plot(thetax(1,end-1),thetay(1,end-1),markend,'MarkerFaceColor',
       purple,'MarkerEdgeColor',purple)
    hold on;
    plot(waypoints(1,1),waypoints(1,2),markstart,'MarkerFaceColor',
       purple,'MarkerEdgeColor',purple)
    hold on;
    plot(waypoints(2:end-1,1),waypoints(2:end-1,2),marktruth,'
       Markersize',2, 'Color',green)
```

```
hold on;
plot(waypoints(end,1),waypoints(end,2),markend,'MarkerFaceColor'
   ,purple,'MarkerEdgeColor',purple)
xlabel('Y(m)','FontSize',14)
ylabel('X(m)','FontSize',14)
legend('','Position Estimate','','','True Value','','FontSize'
   ,14,'Location','southeast')
title('Kalman Filter using IMU and Wheel Encoder Based Position
   (x,y) Estimate','FontSize',14)
hold off;

%% Linear Weighted Fusion of Heading Angles : Weights 1

alpha_s = 0.9; alpha_enc = 0.09; alpha_imu = 0.01;


%Gyroscope to heading angle

gyro_deg = rad2deg(gyro(:,3));
head_imu = cumtrapz(times,gyro_deg);


theta_fused = zeros(length(heading_corrected),1);
for cnt = 1:length(heading_corrected)
    theta_fused(cnt) = alpha_s * heading_corrected(cnt) +
       alpha_imu * head_imu(cnt+15) + alpha_enc * rad2deg(
       odom_orient(cnt+15));
end
theta_fused_rad = deg2rad(theta_fused);

%Fused_theta odometry
odom_x_fused = zeros(length(theta_fused_rad),1);
odom_y_fused = zeros(length(theta_fused_rad),1);
odom_orient_fused = zeros(length(theta_fused_rad),1);

odom_x_fused(1:5) = odom_x(1:5);
odom_y_fused(1:5) = odom_y(1:5);
```

```
odom_orient_fused(1:5) = theta_fused_rad(1:5);

for i = 5:length(theta_fused_rad)
nu_l = ticks(i,1)/t * (pi/180) * (360/2048);
nu_r = ticks(i,2)/t * (pi/180) * (360/2048);

V_l = nu_l * encoder.WheelRadius(1);
V_r = nu_r * encoder.WheelRadius(2);

nu = (V_l + V_r)/2;
omega = (V_r - V_l)/L;

k00 = nu * cos(theta_fused_rad(i));
k01 = nu * sin(theta_fused_rad(i));
k02 = omega;

k10 = nu * cos(theta_fused_rad(i) + t/2 * k02);
k11 = nu * sin(theta_fused_rad(i) + t/2 * k02);
k12 = omega;

k20 = nu * cos(theta_fused_rad(i) + t/2 * k12);
k21 = nu * sin(theta_fused_rad(i) + t/2 * k12);
k22 = omega;

k30 = nu * cos(theta_fused_rad(i) + t * k22);
k31 = nu * sin(theta_fused_rad(i) + t * k22);
k32 = omega;

odom_x_fused(i+1) = odom_x_fused(i) + t/6 * (k00 + 2*(k10 +k20)
   + k30);
odom_y_fused(i+1) = odom_y_fused(i) + t/6 * (k01 + 2*(k11 +k21)
   + k31);
odom_orient_fused(i+1) = odom_orient_fused(i) + t/6 * (k02 + 2*(
   k12 +k22) + k32);

end
odom_orient_fused = rad2deg(odom_orient_fused);
```

```
figure;
plot(odom_x_fused(1),odom_y_fused(1),markstart,'MarkerFaceColor',purple,'MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(odom_x_fused(2:end-1),odom_y_fused(2:end-1),markerest, 'Markersize',2, 'Color',red )
hold on;
plot(odom_x_fused(end),odom_y_fused(end),markend,'MarkerFaceColor',purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(15,1),waypoints(15,2),markstart,'MarkerFaceColor',purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(16:length(heading_corrected)+14,1),waypoints(16:length(heading_corrected)+14,2),marktruth,'Markersize',2, 'Color',green)
hold on;
plot(waypoints(length(heading_corrected)+15,1),waypoints(length(heading_corrected)+15,2),markend,'MarkerFaceColor',purple,'MarkerEdgeColor',purple)
xlabel('X(m)','FontSize',14)
ylabel('Y(m)','FontSize',14)
title('True Position vs Fused Heading Based Position (x,y) Estimate','FontSize',14)
legend('','Position Estimate','','','True Value','','FontSize',14,'Location','southeast')

%% Linear Weighted Fusion: Weights Set 2

alpha_s = 0.6; alpha_enc = 0.35; alpha_imu = 0.05;


%gyroscope to heading angle
dt = 1;
```

```
theta_fused1 = zeros(length(heading_corrected),1);
for cnt = 1:length(heading_corrected)
    theta_fused1(cnt) = alpha_s * heading_corrected(cnt) +
       alpha_imu * head_imu(cnt+15) + alpha_enc * rad2deg(
       odom_orient(cnt+15));
end
theta_fused1_rad = deg2rad(theta_fused1);

%Fused_theta odometry
odom_x_fused1 = zeros(length(theta_fused1_rad),1);
odom_y_fused1 = zeros(length(theta_fused1_rad),1);
odom_orient_fused1 = zeros(length(theta_fused1_rad),1);

odom_x_fused1(1:5) = odom_x(1:5);
odom_y_fused1(1:5) = odom_y(1:5);
odom_orient_fused1(1:5) = theta_fused1_rad(1:5);

for i = 5:length(theta_fused1_rad)
nu_l = ticks(i,1)/t * (pi/180) * (360/2048);
nu_r = ticks(i,2)/t * (pi/180) * (360/2048);

V_l = nu_l * encoder.WheelRadius(1);
V_r = nu_r * encoder.WheelRadius(2);

nu = (V_l + V_r)/2;
omega = (V_r - V_l)/L;

k00 = nu * cos(theta_fused1_rad(i));
k01 = nu * sin(theta_fused1_rad(i));
k02 = omega;

k10 = nu * cos(theta_fused1_rad(i) + t/2 * k02);
k11 = nu * sin(theta_fused1_rad(i) + t/2 * k02);
k12 = omega;

k20 = nu * cos(theta_fused1_rad(i) + t/2 * k12);
k21 = nu * sin(theta_fused1_rad(i) + t/2 * k12);
```

```
706 k22 = omega;

    k30 = nu * cos(theta_fused1_rad(i) + t * k22);
    k31 = nu * sin(theta_fused1_rad(i) + t * k22);
    k32 = omega;
711
    odom_x_fused1(i+1) = odom_x_fused1(i) + t/6 * (k00 + 2*(k10 +k20
       ) + k30);
    odom_y_fused1(i+1) = odom_y_fused1(i) + t/6 * (k01 + 2*(k11 +k21
       ) + k31);
    odom_orient_fused1(i+1) = odom_orient_fused1(i) + t/6 * (k02 +
       2*(k12 +k22) + k32);

716 end
    odom_orient_fused1 = rad2deg(odom_orient_fused1);

    figure;
    plot(odom_x_fused(1),odom_y_fused(1),markstart,'MarkerFaceColor'
       ,purple,'MarkerEdgeColor',purple)
721 set(gca,'FontSize',14)
    hold on;
    plot(odom_x_fused(2:end-1),odom_y_fused(2:end-1),markerest, '
       Markersize',2,'Color',red)
    hold on;
    plot(odom_x_fused(end),odom_y_fused(end),markend, '
       MarkerFaceColor',purple,'MarkerEdgeColor',purple)
726 hold on;
    plot(odom_x_fused1(1),odom_y_fused1(1),markstart,'
       MarkerFaceColor',purple,'MarkerEdgeColor',purple)
    set(gca,'FontSize',14)
    hold on;
    plot(odom_x_fused1(2:end-1),odom_y_fused1(2:end-1),markerest, '
       Markersize',2,'Color',black)
731 hold on;
    plot(odom_x_fused1(end),odom_y_fused1(end),markend, '
       MarkerFaceColor',purple,'MarkerEdgeColor',purple)
    hold on;
```

```
plot(waypoints(15,1),waypoints(15,2),markstart,'MarkerFaceColor'
    ,purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(16:length(heading_corrected)+ 14,1),waypoints(16:
    length(heading_corrected)+14,2),marktruth,'Markersize',2, '
    Color',green)
hold on;
plot(waypoints(length(heading_corrected)+15,1),waypoints(length(
    heading_corrected)+15,2),markend,'MarkerFaceColor',purple,'
    MarkerEdgeColor',purple)
xlabel('X(m)','FontSize',14)
ylabel('Y(m)','FontSize',14)
title('True Position vs Fused Heading Based Position (x,y)
    Estimate','FontSize',14)
legend('','Position Estimate - Weights:1','','','Position
    Estimate - Weights:2','','','True Value','','FontSize',14,'
    Location','southeast')

%% OPTIMAL WEIGHTS

theta_measurement = zeros(600,3);

for cnt = 1:length(theta_measurement)
    theta_measurement(cnt,:) = [head_imu(cnt+15), rad2deg(
       odom_orient(cnt+15)), heading_corrected(cnt)];
end

theta_true = (orient_truth(15:cnt+14));

%weights= lsqnonneg(theta_measurement,theta_true);

weights = lsqlin(theta_measurement,theta_true,[],[],ones(1,3),1,
    zeros(3,1),0.95*ones(1,3));
weight_imu = weights(1); weight_we = weights(2); weight_ss =
    weights(3);

theta_fused_opt = zeros(length(heading_corrected),1);
```

```
for cnt = 1:length(heading_corrected)
    theta_fused_opt(cnt) = weight_ss * heading_corrected(cnt) +
       weight_imu * head_imu(cnt+15) + weight_we * rad2deg(
       odom_orient(cnt+15));
end

theta_fused_rad_opt = deg2rad(theta_fused_opt);

%Fused_theta odometry
odom_x_fused_opt = zeros(length(theta_fused_rad_opt),1);
odom_y_fused_opt = zeros(length(theta_fused_rad_opt),1);
odom_orient_fused_opt = zeros(length(theta_fused_rad_opt),1);

odom_x_fused_opt(1:5) = odom_x(1:5);
odom_y_fused_opt(1:5) = odom_y(1:5);
odom_orient_fused_opt(1:5) = theta_fused_rad_opt(1:5);

for i = 5:length(theta_fused_rad_opt)
nu_l = ticks(i,1)/t * (pi/180) * (360/2048);
nu_r = ticks(i,2)/t * (pi/180) * (360/2048);

V_l = nu_l * encoder.WheelRadius(1);
V_r = nu_r * encoder.WheelRadius(2);

nu = (V_l + V_r)/2;
omega = (V_r - V_l)/L;

k00 = nu * cos(theta_fused_rad_opt(i));
k01 = nu * sin(theta_fused_rad_opt(i));
k02 = omega;

k10 = nu * cos(theta_fused_rad_opt(i) + t/2 * k02);
k11 = nu * sin(theta_fused_rad_opt(i) + t/2 * k02);
k12 = omega;

k20 = nu * cos(theta_fused_rad_opt(i) + t/2 * k12);
k21 = nu * sin(theta_fused_rad_opt(i) + t/2 * k12);
```

```
    k22 = omega;
796
    k30 = nu * cos(theta_fused_rad_opt(i) + t * k22);
    k31 = nu * sin(theta_fused_rad_opt(i) + t * k22);
    k32 = omega;

801 odom_x_fused_opt(i+1) = odom_x_fused_opt(i) + t/6 * (k00 + 2*(
       k10 +k20) + k30);
    odom_y_fused_opt(i+1) = odom_y_fused_opt(i) + t/6 * (k01 + 2*(
       k11 +k21) + k31);
    odom_orient_fused_opt(i+1) = odom_orient_fused_opt(i) + t/6 * (
       k02 + 2*(k12 +k22) + k32);

    end
806 odom_orient_fused_opt = rad2deg(odom_orient_fused_opt);

    figure;
    plot(odom_x_fused_opt(1),odom_y_fused_opt(1),markstart,'
       MarkerFaceColor',purple,'MarkerEdgeColor',purple)
    set(gca,'FontSize',14)
811 hold on;
    plot(odom_x_fused_opt(2:end-1),odom_y_fused_opt(2:end-1),
       markerest, 'Markersize',2, 'Color',red )
    hold on;
    plot(odom_x_fused_opt(end),odom_y_fused_opt(end),markend,'
       MarkerFaceColor',purple,'MarkerEdgeColor',purple)
    hold on;
816 plot(waypoints(15,1),waypoints(15,2),markstart,'MarkerFaceColor'
       ,purple,'MarkerEdgeColor',purple)
    hold on;
    plot(waypoints(16:length(heading_corrected)+14,1),waypoints(16:
       length(heading_corrected)+14,2),marktruth,'Markersize',2, '
       Color',green)
    hold on;
    plot(waypoints(length(heading_corrected)+15,1),waypoints(length(
       heading_corrected)+15,2),markend,'MarkerFaceColor',purple,'
       MarkerEdgeColor',purple)
```

```
821 xlabel('X(m)','FontSize',14)
    ylabel('Y(m)','FontSize',14)
    title('True Position vs Fused Heading Based Position (x,y)
       Estimate','FontSize',14)
    legend('','Position Estimate','','','True Value','','FontSize'
       ,14,'Location','southeast')

826 %% RMSE: Linear Weighted Fusion

    rmse_fuse_xposn = rmse(odom_x_fused(2:end),waypoints(15:length(
       heading_corrected)+14,1));
    rmse_fuse_yposn = rmse(odom_y_fused(2:end),waypoints(15:length(
       heading_corrected)+14,2));
    max_diff_fuse_xposn = max(abs(odom_x_fused(2:end) - waypoints
       (15:length(heading_corrected)+14,1) ));
831 max_diff_fuse_yposn = max(abs(odom_y_fused(2:end) - waypoints
       (15:length(heading_corrected)+14,2) ));

    rmse_fuse1_xposn = rmse(odom_x_fused1(2:end),waypoints(15:length
       (heading_corrected)+14,1));
    rmse_fuse1_yposn = rmse(odom_y_fused1(2:end),waypoints(15:length
       (heading_corrected)+14,2));
    max_diff_fuse1_xposn = max(abs(odom_x_fused1(2:end) - waypoints
       (15:length(heading_corrected)+14,1) ));
836 max_diff_fuse1_yposn = max(abs(odom_y_fused1(2:end) - waypoints
       (15:length(heading_corrected)+14,2) ));

    rmse_fuse_opt_xposn = rmse(odom_x_fused_opt(2:end),waypoints(15:
       length(heading_corrected)+14,1));
    rmse_fuse_opt_yposn = rmse(odom_y_fused_opt(2:end),waypoints(15:
       length(heading_corrected)+14,2));
    max_diff_fuse_opt_xposn = max(abs(odom_x_fused_opt(2:end) -
       waypoints(15:length(heading_corrected)+14,1) ));
841 max_diff_fuse_opt_yposn = max(abs(odom_y_fused_opt(2:end) -
       waypoints(15:length(heading_corrected)+14,2) ));

    %% SIMPLE FUSION
```

```
Windowlength = 25;
orient_truth_cut = truth_angle_mod2(15:end);
odom_orient_deg = rad2deg(odom_orient);
times_cutmod = times_cut3(13:length(heading_angle3) -
   Windowlength,1);

theta_simple_fused = zeros(length(heading_corrected),1);

for idx = Windowlength+1:length(heading_corrected)
    Blockh = heading_corrected(idx-Windowlength:idx);
    Blocko = orient_truth_cut(idx-Windowlength:idx);
    Blockw = odom_orient_deg(15+idx-Windowlength:idx+15);
    Biasho(idx) = mean(abs(Blockh-Blocko));
    Biaswo(idx) = mean(abs(Blockw-Blocko));
end

theta_simple_fused(1:Windowlength) = heading_corrected(1:
   Windowlength);
for indx = Windowlength+1: length(heading_corrected)
    if Biaswo(indx)<15
        theta_simple_fused(indx) = odom_orient_deg(15+indx);
    else
        theta_simple_fused(indx) = heading_corrected(indx);
    end

end

theta_simple_fused_rad = deg2rad(theta_simple_fused);

%Fused_theta odometry
odom_x_fused_simple = zeros(length(theta_simple_fused_rad),1);
odom_y_fused_simple = zeros(length(theta_simple_fused_rad),1);
odom_orient_fused_simple = zeros(length(theta_simple_fused_rad)
   ,1);

odom_x_fused_simple(1:5) = odom_x(1:5);
```

```
odom_y_fused_simple(1:5) = odom_y(1:5);
odom_orient_fused_simple(1:5) = theta_simple_fused_rad(1:5);

for i = 5:length(theta_simple_fused_rad)
nu_l = ticks(i,1)/t * (pi/180) * (360/2048);
nu_r = ticks(i,2)/t * (pi/180) * (360/2048);

V_l = nu_l * encoder.WheelRadius(1);
V_r = nu_r * encoder.WheelRadius(2);

nu = (V_l + V_r)/2;
omega = (V_r - V_l)/L;

k00 = nu * cos(theta_simple_fused_rad(i));
k01 = nu * sin(theta_simple_fused_rad(i));
k02 = omega;

k10 = nu * cos(theta_simple_fused_rad(i) + t/2 * k02);
k11 = nu * sin(theta_simple_fused_rad(i) + t/2 * k02);
k12 = omega;

k20 = nu * cos(theta_simple_fused_rad(i) + t/2 * k12);
k21 = nu * sin(theta_simple_fused_rad(i) + t/2 * k12);
k22 = omega;

k30 = nu * cos(theta_simple_fused_rad(i) + t * k22);
k31 = nu * sin(theta_simple_fused_rad(i) + t * k22);
k32 = omega;

odom_x_fused_simple(i+1) = odom_x_fused_simple(i) + t/6 * (k00 +
    2*(k10 +k20) + k30);
odom_y_fused_simple(i+1) = odom_y_fused_simple(i) + t/6 * (k01 +
    2*(k11 +k21) + k31);
odom_orient_fused_simple(i+1) = odom_orient_fused_simple(i) + t
   /6 * (k02 + 2*(k12 +k22) + k32);

end
```

```
odom_orient_fused_simple = rad2deg(odom_orient_fused_simple);


figure;
plot(odom_x_fused_simple(1),odom_y_fused_simple(1),markstart,'
    MarkerFaceColor',purple,'MarkerEdgeColor',purple)
set(gca,'FontSize',14)
hold on;
plot(odom_x_fused_simple(2:end-1),odom_y_fused_simple(2:end-1),
    markerest, 'Markersize',2,'Color',red)
hold on;
plot(odom_x_fused_simple(end),odom_y_fused_simple(end),markend,'
    MarkerFaceColor',purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(1,1),waypoints(1,2),markstart,'MarkerFaceColor',
    purple,'MarkerEdgeColor',purple)
hold on;
plot(waypoints(2:length(heading_corrected)-1,1),waypoints(2:
    length(heading_corrected)-1,2),marktruth,'Markersize',2, '
    Color',green)
hold on;
plot(waypoints(length(heading_corrected),1),waypoints(length(
    heading_corrected),2),markend,'MarkerFaceColor',purple,'
    MarkerEdgeColor',purple)
xlabel('X(m)','FontSize',14)
ylabel('Y(m)','FontSize',14)
legend('','Position Estimate','','','True Value','','FontSize'
    ,14,'Location','southeast')
title('True Position vs Simple Fusion Heading Based Position (x,
    y) Estimate','FontSize',14)

rmse_simple_fuse_xposn = rmse(odom_x_fused_simple(2:end),
    waypoints(15:length(heading_corrected)+14,1));
rmse_simple_fuse_yposn = rmse(odom_y_fused_simple(2:end),
    waypoints(15:length(heading_corrected)+14,2));

max_simple_fuse_xposn = max(abs(odom_x_fused_simple(2:end) -
```

```
    waypoints(15:length(heading_corrected)+14,1) ));
max_simple_fuse_yposn = max(abs(odom_y_fused_simple(2:end) -
    waypoints(15:length(heading_corrected)+14,2) ));

%% BOX PLOT: x & y direction

bpx1 = abs(thetax(1,:)- waypoints(:,1)') ;
bpx2 = abs(odom_x_fused(2:end)'- waypoints(15:length(
    heading_corrected)+14,1)');
bpx3 = abs(odom_x_fused1(2:end)' - waypoints(15:length(
    heading_corrected)+14,1)');
bpx4 = abs(odom_x_fused_simple(2:end)' - waypoints(15:length(
    odom_x_fused_simple)+13,1)');
bpx = [bpx1 bpx2 bpx3 bpx4];
grp = [zeros(1,888),ones(1,849),2*ones(1,849),3*ones(1,849)];
figure;
boxplot(bpx,grp,'Labels',{'KF','LW:1','LW:2','SF'});
set(gca,'FontSize',14)
title('Box Plot of Errors in x for All Fusion Methods','FontSize
    ',14)

bpy1 = abs(thetay(1,:)- waypoints(:,2)') ;
bpy2 = abs(odom_y_fused(2:end)'- waypoints(15:length(
    heading_corrected)+14,2)');
bpy3 = abs(odom_y_fused1(2:end)' - waypoints(15:length(
    heading_corrected)+14,2)');
bpy4 = abs(odom_y_fused_simple(2:end)' - waypoints(15:length(
    odom_x_fused_simple)+13,2)');
bpy = [bpy1 bpy2 bpy3 bpy4];
grp = [zeros(1,888),ones(1,849),2*ones(1,849),3*ones(1,849)];
figure;
boxplot(bpx,grp,'Labels',{'KF','LW:1','LW:2','SF'});
set(gca,'FontSize',14)
title('Box Plot of Errors in y for All Fusion Methods','FontSize
    ',14)

%% FUNCTIONS : Heading Angle
```

```
function heading = heading_calc(W,V,lat,lon)
B = W * V';
Q = B + B';

Z = [B(2,3)-B(3,2) , B(3,1)-B(1,3),  B(1,2)-B(2,1)];
sigma = trace(B);

K = [Q - ones(3,3)*sigma , Z';...
     Z , sigma];

[V,D] = eigs(K,1);

q_v = V(1:3,1);
q_s = V(4);

q_v_skew = skew(q_v);
C_sf = (q_s^2 - (q_v' * q_v))* ones(3,3) + 2 * (q_v * q_v') - 2
   * q_s * q_v_skew;

%C_ft = rotz(lon) * roty(pi/2 - lat) * rotz(pi/2);
C_ft = rotz(lon) * roty(rad2deg(pi/2) - lat) * rotz(rad2deg(pi
   /2));

C_ts = C_ft' * C_sf';


heading = rad2deg(atan2(C_ts(2,1),C_ts(1,1)));
end

%% JULIAN DAY

function JD = JD_calc(h,m,s)

Y = 2008;
M1 = 7;
```

```
A = floor(Y/100);
B = 2 - A + floor(A/4);

%The (h + 5) is used as the measurements are in local EST time
   and for
%solar position calculations, we must convert to standard UTC.
   The offset
%to UTC for EST would be +5, as the time zone is (-5)

D = 12 + ((h + 5)/24.0) + (m/1440.0) + s/86400.0; % h min s ms
JD = floor(365.25 * (Y + 4716)) + floor(30.6001 * (M1+1)) + D +
   B - 1524.5;
end


%% GAST ANGLE
function gast_mod = gast_angle(JD)
T = (JD - 2451545.0)/36525;
%GAST
theta_gast = 280.46061837 + 360.98564736629 * (JD - 2451545.0) +
    0.000387933 * T^2 - T^3 / 38710000;

% This part is from Chapter 22 for ecliptic
omega_g  = 125.04452 - 1934.136261 * T + 0.0020708 * T^2 + T^3 /
    450000;

L  = 280.4665 + 36000.7698 * T; % in deg; mean lon of Sun
L1 = 218.3165 + 481267.8813 * T; % mean lon of Moon

%To dms
delta_psi = dms2degrees([0 0 -17.20]) * sin(deg2rad(omega_g)) -
   dms2degrees([0 0 1.32]) * sin(deg2rad(2*L)) - dms2degrees([0
   0 0.23]) * sin(deg2rad(2*L1)) + dms2degrees([0 0 0.21]) * sin
   (deg2rad(2*omega_g));
%To dms
delta_eps = dms2degrees([0 0 9.20]) * cos(deg2rad(omega_g)) +
   dms2degrees([0 0 0.57]) * cos(deg2rad(2*L)) - dms2degrees([0
```

```
    0 0.10]) * cos(deg2rad(2*L1)) - dms2degrees([0 0 0.09]) * cos
    (deg2rad(2*omega_g));

%To dms
eps0 = dms2degrees([23 26 21.448]) - dms2degrees([0 0 46.8150])
    * T - dms2degrees([0 0 0.00059]) * T^2 + dms2degrees([0 0
    0.001813]) * T^3;
%To dms
eps_g = eps0 + delta_eps;

%Continuation of GAST
corr = delta_psi * cos(deg2rad(eps_g))/15;
gast = theta_gast + corr;
gast_mod = correctdegrange(gast);
end

%% CORRECTED DEGREE RANGE
function d = correctdegrange(angle)
d = rem(angle,360);
if d<0
    d = d + 360;
end
end

%% CONSTRUCT SUN VECTOR
function svec = sun_vector(a,b)
svec = [sin(deg2rad(b)) * cos(deg2rad(a)), sin(deg2rad(b)) * sin
    (deg2rad(a)), cos(deg2rad(b))]';
end

%% SKEW MATRICES
function sk = skew(G_f)
sk = [0 -G_f(3) G_f(2);...
           G_f(3) 0  -G_f(1);...
           -G_f(2) G_f(1) 0];
end
```

```
%% ROTATION MATRICES
function rx = rotx(angle)
rx = [1 0 0; 0 cos(deg2rad(angle)) -sin(deg2rad(angle)); 0 sin(
   deg2rad(angle)) cos(deg2rad(angle))];
end

function ry = roty(angle)
ry = [cos(deg2rad(angle)) 0 sin(deg2rad(angle)); 0 1 0 ; -sin(
   deg2rad(angle)) 0 cos(deg2rad(angle))];
end

function rz = rotz(angle)
rz = [cos(deg2rad(angle)) -sin(deg2rad(angle)) 0; sin(deg2rad(
   angle)) cos(deg2rad(angle)) 0; 0 0 1];
end

%% EPHEMERIS
function [a,b] = calc_eph(JD,theta_gast,lat,long)

T = (JD - 2451545.0)/36525;

L0 = 280.46646 + 36000.76983 * T + 0.0003032 * T^2; %Geometric
   Mean Longitude of the Sun
L0 = correctdegrange(L0);

M = 357.52911 + 35999.05029  * T - 0.0001537 * T^2; % Mean
   Anomaly of the Sun
M = correctdegrange(M);

e = 0.016708634 - 0.000042037 * T - 0.0000001267 * T^2; %
   Eccentricity of the Earth's Orbit
C = +(1.914602 - 0.004817 * T - 0.000014 * T^2) * sin(deg2rad(M)
   )...
    + (0.019993 - 0.000101 * T) * sin(deg2rad(2 * M)) ...
    + 0.000289 * sin(deg2rad(3 * M)); %Sun's Equation of the
       Centre
```

```
Ltrue = L0 + C;
Ltrue = correctdegrange(Ltrue); %Sun's true geometric longitude

n = M + C; %Sun's true anomaly
n = correctdegrange(n);

R = 1.000001018 * (1 - e^2) / (1 + e * cos(deg2rad(n))); %Radius
    Vector of Sun

%Apparent Longitude of the Sun
omega = 125.04 - 1934.136 * T;
Lapp = Ltrue - 0.00569 - 0.00478 * sin(deg2rad(omega));

%Obliquity of the Ecliptic 22.3
U = T/100;
eps0 = dms2degrees([23 26 21.448]) - 4680.93/3600.0 * U ...
    - 1.55 * U^2 + 1999.25 * U^3 ...
    -31.38 * U^4 - 249.67 * U^5 ...
    -39.05 * U^6 +7.12 * U^7 ...
    +27.87 * U^8 + 5.79 * U^9 +2.45 * U^10;

%Solar Coordinates
%Correction for parallax
eps0_corr = eps0 + 0.00256 * cos(deg2rad(omega));

%Sun's right ascension and declination
asc = rad2deg(atan2(cos(deg2rad(eps0_corr))* sin(deg2rad(Lapp)),
    cos(deg2rad(Lapp))));
decl = rad2deg(asin(sin(deg2rad(eps0_corr))* sin(deg2rad(Lapp)))
   );

HA = theta_gast - long - asc;

az = rad2deg( atan(sin(deg2rad(HA)) / (cos(deg2rad(HA)) * sin(
   deg2rad(lat)) - tan(deg2rad(decl)) * cos(deg2rad(lat))) ));

alt = rad2deg(asin(sin(deg2rad(lat)) * sin(deg2rad(decl)) + cos(
```

```
    deg2rad(lat)) * cos(deg2rad(decl)) * cos(deg2rad(HA)) ));
b = 90 - alt; %Zenith Angle

HA = HA + 180;
HA = correctdegrange(HA);

%Default: Clock wise from South
%az = az + 180; %To make it from North-clock wise
a =  az;

%az_corr = correctdegrange(az);
%a = az_corr; % Azimuth angle

end
```