

Master's Programme in Electronics and Electrical Engineering

# Evaluation of Triangle Based Star Identification Algorithms under Controlled Sky Image Simulations

---

**Rauli Manninen**

© 2025

This work is licensed under a [Creative Commons](https://creativecommons.org/licenses/by-nc-sa/4.0/) “Attribution-NonCommercial-ShareAlike 4.0 International” license.



---

**Author** Rauli Manninen

---

**Title** Evaluation of Triangle Based Star Identification Algorithms under Controlled Sky Image Simulations

---

**Degree programme** Electronics and Electrical Engineering

---

**Major** Space Science and Technology

---

**Supervisor** Prof. Esa Kallio

---

**Advisor** MSc Olli Knuuttila

---

**Date** 22 December 2025

**Number of pages** 68

**Language** English

---

**Abstract**

This thesis investigates the performance of two triangle based star sensor algorithms within a controlled simulation environment, namely the Liebe triangle algorithm and the planar triangle algorithm. Star recognition algorithms play a critical role in satellite operations due to the increasing demands on pointing accuracy and the fundamental importance of reliable attitude determination. As new star recognition algorithms are continuously proposed in scientific literature, often introducing novel ideas and variations, assessing their true effectiveness remains challenging. Performance comparisons are frequently hindered by differing simulation conditions, test setups, and evaluation criteria, making it difficult to determine which algorithms or algorithmic components are genuinely effective and how they compare against alternative approaches. The primary objective of this thesis is to perform a detailed comparison of two selected algorithms by developing a controlled and configurable simulation environment that enables a fair and systematic evaluation. The simulation framework models the behavior of a star sensor and the visible night sky to generate synthetic star images, incorporating adjustable parameters such as camera characteristics, field of view, positional noise, and the presence of false stars. These simulated images serve as input for the implemented algorithms, allowing their performance to be analyzed under varying conditions and configurations. Comparative results are obtained from multiple simulation runs in which both algorithms are evaluated using identical initial parameters to ensure consistency and fairness. The results highlight the overall performance of each algorithm and provide insight into which internal components contribute most effectively to successful star recognition. The study demonstrates that a controlled testing environment enables meaningful algorithm comparison, and the results indicate that the Liebe triangle algorithm performs significantly more robustly than the planar triangle algorithm, particularly in challenging scenarios.

---

**Keywords** Star Recognition, Star Sensor Algorithms, Star Tracker, Attitude Determination, Spacecraft Navigation

---

---

**Tekijä** Rauli Manninen

---

**Työn nimi** Kolmioihin perustuvien tähtien tunnistusalgoritmien arviointi  
kontrolloiduissa taivaskuvasimulaatioissa

---

**Koulutusohjelma** Elektroniikan ja nanoteknologian maisteriohjelma

---

**Pääaine** Avaruustiede ja -teknologia

---

**Työn valvoja** Prof. Esa Kallio

---

**Työn ohjaaja** DI Olli Knuutila

---

**Päivämäärä** 22.12.2025

**Sivumäärä** 68

**Kieli** englanti

---

### **Tiivistelmä**

Tässä diplomityössä tutkitaan kahden tähtisensorin tähtientunnistusalgoritmin suorituskykyä kontrolloidussa simulaatioympäristössä. Tutkittavat algoritmit ovat Lieben kolmioalgoritmi sekä planar triangle -algoritmi. Tähtientunnistusalgoritmit ovat keskeisessä roolissa satelliittien toiminnassa kasvavien suuntaustarkkuusvaatimusten sekä luotettavan asennonmäärityksen merkityksen vuoksi. Tieteellisessä kirjallisuudessa julkaistaan jatkuvasti uusia tähtientunnistusalgoritmeja sekä niiden uusia variaatioita, lähes aina poikkeavissa simulaatioolosuhteissa. Tästä johtuen eri menetelmien välisen suorituskyvyn ja todellisen tehokkuuden vertailukelpoinen arviointi on osoittautunut haastavaksi. Algoritmien suorituskyvyn vertailua vaikeuttavat usein toisistaan poikkeavat simulaatio-olosuhteet, testausasetelmat ja arviointikriteerit, mikä tekee erilaisten ratkaisujen ja yksittäisten algoritmien osien keskinäisestä vertailusta vaikeaa. Tämän työn päätavoitteena on suorittaa kahden valitun algoritmin yksityiskohtainen vertailu kehittämällä kontrolloitu ja muunneltava simulaatioympäristö, joka mahdollistaa reilun ja systemaattisen arvioinnin. Simulaatioympäristö mallintaa tähtisensorin toimintaa ja näkyvää yötaivasta tuottaen synteettisiä tähtikuvia, joissa voidaan muuttaa muun muassa kameran ominaisuuksia, näkökenttää, sekä hallita kohinaa, näkyvien tähtien sekä väärin tähtien määrää. Näitä simuloituja tähtikuvia käytetään toteutettujen algoritmien syöteinä, mikä mahdollistaa niiden suorituskyvyn analysoinnin erilaisissa olosuhteissa ja konfiguraatioissa. Vertailudata on kerätty useista simulaatioajoista, joissa molemmat algoritmit arvioidaan identtisillä alkuparametreilla vertailun johdonmukaisuuden ja reiluuden varmistamiseksi. Tulokset havainnollistavat algoritmien kokonaisvaltaista suorituskykyä ja tarjoavat tietoa siitä, mitkä sisäiset algoritmikomponentit vaikuttavat tehokkaimmin onnistuneeseen tähtientunnistukseen. Tutkimus osoittaa, että kontrolloitu testausympäristö mahdollistaa mielekkään algoritmien välisen vertailun, ja tulosten perusteella Lieben kolmioalgoritmi suoriutuu selvästi paremmin kuin planar triangle -algoritmi erityisesti haastavissa tilanteissa.

---

**Avainsanat** tähtien tunnistus, tähtien tunnistus algoritmi, tähtisensori, satelliitti

---

## **Preface**

I want to thank Professor Esa Kallio and my advisor Olli Knuuttila for their guidance and for introducing me to the field of star recognition algorithms. I also want to thank my family and friends for their support, encouragement, and patience throughout my studies. Their help has been invaluable during this work and through the years leading up to it.

Espoo, 22 December 2025

Rauli Manninen

# Contents

<b>Abstract</b>	<b>3</b>
<b>Abstract (in Finnish)</b>	<b>4</b>
<b>Preface</b>	<b>5</b>
<b>Contents</b>	<b>6</b>
<b>Symbols and abbreviations</b>	<b>8</b>
<b>1 Introduction</b>	<b>9</b>
1.1 Lost-in-Space . . . . .	9
1.2 Inconsistency of Star Recognition Algorithms . . . . .	9
1.3 Research Objectives and Approach . . . . .	10
1.4 Thesis Structure . . . . .	11
<b>2 Star Recognition Algorithms</b>	<b>12</b>
2.1 Attitude Sensors and Star Recognition . . . . .	12
2.2 Overview of Star Recognition Algorithms . . . . .	14
2.2.1 Classical Geometry-Based Approaches . . . . .	15
2.2.2 Machine Learning Based Models . . . . .	20
2.3 General Algorithm Pipeline . . . . .	21
2.3.1 Image Acquisition and Preprocessing . . . . .	21
2.3.2 Star Catalog and database . . . . .	22
2.3.3 Feature Matching . . . . .	25
2.3.4 Filtering and pivoting . . . . .	26
2.3.5 Verification and Estimation . . . . .	28
2.4 Challenges in Star Recognition . . . . .	28
2.5 Algorithms in this Thesis . . . . .	30
2.5.1 Liebe's Triangle Algorithm . . . . .	31
2.5.2 Planar Triangle Algorithm . . . . .	33
<b>3 Algorithm Implementation and Simulations</b>	<b>35</b>
3.1 Simulation Environment . . . . .	35
3.1.1 Simulated Star Sensor Images . . . . .	35
3.1.2 Different Simulations for Data Gathering . . . . .	38
3.2 Algorithm Implementation in Python . . . . .	39
3.2.1 Catalog and Database Construction . . . . .	39
3.2.2 Liebe Algorithm Implementation . . . . .	43
3.2.3 Planar Triangle Algorithm Implementation . . . . .	46
3.3 Simulation Structure and Data Gathering . . . . .	48

<b>4</b>	<b>Results and Discussion</b>	<b>51</b>
4.1	Overall accuracy and Success Rates . . . . .	51
4.2	Performance under Different Conditions . . . . .	54
4.2.1	Effect of FOV . . . . .	54
4.2.2	Effect of False Stars . . . . .	55
4.2.3	Effect of Noise . . . . .	56
4.2.4	Effect of Amount of Visible Stars . . . . .	57
4.3	Failure Analyses . . . . .	59
4.4	Algorithm Comparison . . . . .	61
4.5	Critical Reflection . . . . .	62
<b>5</b>	<b>Conclusion</b>	<b>63</b>
5.1	Summary of Work . . . . .	63
5.2	Main findings . . . . .	63
5.3	Future Work . . . . .	64
	<b>References</b>	<b>65</b>

# Symbols and abbreviations

## Symbols

$I_i$	Gray value of pixel $i$
$\vec{a}$	Vector
$A$	Area
$J$	Polar Moment

## Operators

$\sum_i$	sum over index $i$
$\vec{a} \cdot \vec{b}$	Vector dot product
$ \vec{a} $	Vector magnitude

## Abbreviations

BSC	Bright Star Catalog
CG	Center of Gravity algorithm
DCNN	Deep Convolutional Neural Network
DEC	Declination
ESA	European Space Agency
FOV	Field of View
GF	Gaussian Function
HIP	Hipparcos catalog Identification Number
PCA	Principle Component Analysis
PTP	Planar Triangle Principle Component Analysis
RA	Right Ascension
TF	Triangle Feature

# 1 Introduction

## 1.1 Lost-in-Space

Consider a spacecraft that has lost track of its orientation after a period of uncontrolled tumbling. Ground control no longer knows where the satellite is pointing, yet mission operations require precise knowledge of its attitude. To recover, the spacecraft relies on a star tracker: a camera designed to image a field of stars. The resulting picture contains stars as bright points of light scattered across the detector. At this point, the main problem emerges: what stars are these, and how can the spacecraft determine its orientation from them?

Without any prior pointing information, the raw image of stars is essentially meaningless. The spacecraft must autonomously identify the observed stars, match them to entries in an onboard catalog, and from that deduce its orientation with respect to the celestial sphere. This challenge is known as the lost-in-space problem [1]. The solution comes in the form of star recognition algorithms, also referred to as lost-in-space algorithms [1]. These methods are capable of autonomously determining the camera's orientation relative to the celestial sphere using only the detected stars, without any prior knowledge of the pointing direction [2]. The algorithms typically rely on geometrical features between the stars, such as angular distances or relative angles between the stars. This makes it possible to identify star groups and then match them to cataloged stars.

Star recognition is an important task not only for recovering lost satellites, but also during nominal operations, where star trackers continuously provide accurate attitude information. Among the various sensor technologies available for spacecraft, star trackers remain the most accurate sensors for attitude determination [1] [3]. In other words, star recognition is not merely a contingency measure for lost spacecraft, but a continuous function supporting precise control throughout the mission. Beyond spacecraft attitude estimations, star recognition also has significant applications in other domains. It is used in the calibration of telescopes (both ground-based and space-based), in sky surveys, and in the broader field of blind astrometry, which underpins astronomical navigation and large-scale imaging pipelines [4].

## 1.2 Inconsistency of Star Recognition Algorithms

Over the past three decades, a wide range of star recognition algorithms for star trackers have been proposed. Classical methods, such as the Liebe triangle algorithm [5][6], pyramid algorithm [7], and the planar triangle algorithm [8], rely on geometry and inter-star relationships in the recognition process. More recently, advancements in computational capabilities have enabled new approaches based on machine learning and deep convolutional neural networks (DCNNs) [1].

The diversity of available algorithms is reflected in their fundamental design assumptions. Key factors include the camera’s field of view (FOV), the properties of the reference star catalog, the algorithm’s robustness to noise and false detections, and its ability to handle error cases, like false or missing stars [1][4]. For example, an algorithm intended for the calibration of a ground-based telescope must account for atmospheric effects, while such variations are largely absent in spaceborne applications. As a result, the performance of star recognition algorithms can vary significantly depending on the application context.

While many studies have introduced and demonstrated the operation of individual algorithms, the evaluations are often conducted under different conditions, using varying image datasets, noise models, and performance metrics. This lack of a consistent testing framework makes it difficult to draw direct comparisons between algorithms and to assess their relative strengths and weaknesses in a fair manner.

### **1.3 Research Objectives and Approach**

The aim of this study is to implement and systematically compare two well-known and widely-cited star recognition algorithms: the Liebe triangle algorithm and the planar triangle algorithm. These algorithms, while both relying on geometric properties between stars, employ different methods for star identification. The primary objective is to provide a fair and unbiased comparison of their performance under controlled conditions, assessing their accuracy, robustness, and computational efficiency, as well as identifying the situations in which each algorithm may fail. A secondary objective is to validate the developed simulation framework as a tool for testing and comparing star recognition algorithms in future research.

To achieve these objectives, a simulation environment is constructed using Python, allowing control over main image parameters, such as FOV, number of visible stars, magnitude and positional noise as well as the presence of false or missing stars. These parameters reflect realistic star tracker conditions, including potential occlusion or errors that may occur in both spaceborne and ground-based systems. Both algorithms are tested on identical simulated images to ensure full comparability, and their implementation follows the original publications as closely as possible. Since the original source code for these algorithms is not widely available, the algorithms are reconstructed based on the documented methodologies as accurately as possible, with catalog size and structure matched to the original design.

This work contributes to a faithful reconstruction of two reference algorithms, a flexible simulation framework for controlled testing, and a systematic analysis of algorithm performance across identical scenarios. The study highlights strengths, weaknesses, and limitations, providing insights into how each method handles noise, limited FOV, and false or missing stars. While the focus is on these two classical algorithms, more advanced approaches, such as deep learning, specifically DCNN, or machine learning methods, are beyond the scope of this work. The evaluation is entirely simulation-based, allowing for a controlled comparison without relying on real spacecraft data.

## **1.4 Thesis Structure**

The rest of this thesis is structured as follows. Chapter 2 provides an overview of star recognition algorithms, including their theoretical background, functional structure, and common challenges. It also examines in detail the mathematical principles and operation of the two algorithms simulated in this study. Chapter 3 addresses the implementation of the simulation environment and the algorithms. Chapter 4 presents the simulation results and compares the performance of the algorithms under identical conditions. Finally, Chapter 5 concludes the thesis, summarizing and highlighting the main findings of this study.

## 2 Star Recognition Algorithms

This chapter reviews the theory behind star recognition algorithms. Section 2.1 examines general methods for spacecraft attitude determination and provides an overview of what star recognition algorithms are. Section 2.2 presents a closer look at different algorithms, including classical approaches and briefly more modern machine learning–based solutions. Next, Section 2.3 describes in detail the general operation of a typical star recognition algorithm, covering the pipeline from image acquisition, feature detection, matching, to verification. Section 2.4 discusses the main challenges faced by these algorithms. Finally, Section 2.5 focuses on the specific algorithms selected for comparison and simulation in this thesis: the Liebe algorithm and the Planar Triangle algorithm.

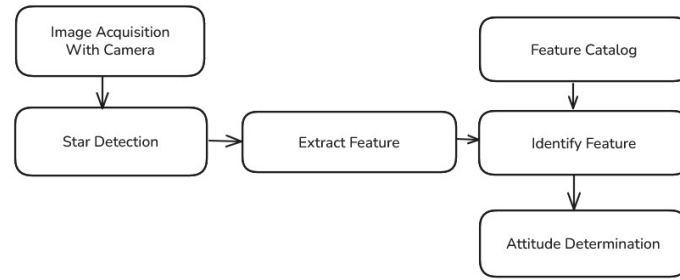
### 2.1 Attitude Sensors and Star Recognition

Attitude refers to the orientation of an object within a known reference frame. Attitude determination has become one of the most critical aspects for satellites [9]. To determine attitude, satellites rely on attitude sensors that provide orientation relative to a reference frame, whether it is the Earth, the Sun, the Moon, star, or another celestial body. Several types of attitude sensors exist, including horizon sensors, Sun sensors, magnetometers, and, more recently, star sensors or star trackers [10]. Their key characteristics are summarized in Table 1.

**Table 1:** Comparison of common attitude sensors. [9][10][12]

Sensor	Ref. Frame	Accuracy	Orbits	Disadvantages
Earth sensor	Earth Horizon	0.03°	Planetary orbit	Non-precise reference; degraded by clouds and solar interference
Sun sensor	Sun	0.1°	Any, Sun visible	Requires Sun in FOV; limited accuracy
Magnetometer	Geomagnetism	1°	Earth LEO Orbit	Usable only near Earth; Limited by magnetic field strength and modeling accuracy
Star tracker	Star	0.001°	Any	Complex; sensitive to bright objects and noise. Slow update rate

Horizon sensors, also known as Earth sensors, determine attitude by detecting the Earth’s horizon; they are therefore suitable only for spacecraft orbiting a planet or moon [11]. Their accuracy is limited, particularly in the presence of high cloud cover and solar interference [9][12].



**Figure 1:** Typical Star Tracker Pipeline.

Sun sensors determine attitude relative to the Sun, making solar visibility a mandatory requirement for their operation [13]. They are small, inexpensive, and powered by solar energy. However, their accuracy is generally lower than that of horizon sensors, and they are limited by the Sun's visibility [9][14]. Magnetometers rely on measurements of Earth's magnetic field to resolve attitude, but their accuracy is the poorest among the alternatives due to variations and inaccuracies in the magnetic field models [9].

Star trackers have emerged in the 21<sup>st</sup> century as a highly reliable and popular alternative, surpassing the previously mentioned attitude sensors [10]. These sensors use the star coordinate system and the visible celestial sphere as their reference. Unlike many other sensors, a star tracker is not limited to Earth orbit; it can also be employed for attitude determination of spacecraft operating throughout the solar system. Moreover, a star tracker does not require prior knowledge of the spacecraft's orientation, as it is capable of determining attitude fully autonomously. Star trackers provide extremely high accuracy (up to  $0.001^\circ$ ), enable autonomous operation, and are indispensable in deep space exploration, among other applications [15][9]. However, in addition to the imaging sensor, they also require star recognition algorithms, which identify reference stars and enable the determination of spacecraft attitude.

Once a star tracker camera has captured an image of the star field, the image is processed by the star tracker's onboard computer to extract attitude information. Without a star recognition algorithm, the star tracker would be nothing more than a camera; attitude determination cannot be performed independently. The algorithm serves as the "brain" of the star tracker, using the information from the image to produce accurate attitude data. In short, stars are first identified in the image, and once their positions are known within the image frame, the observed stars can be matched against an onboard catalog to determine which stars are being observed. Once the pointing direction toward specific stars is established, the spacecraft's attitude can be calculated [16]. This typical star tracker process, from image to attitude data, is shown in Figure 1.

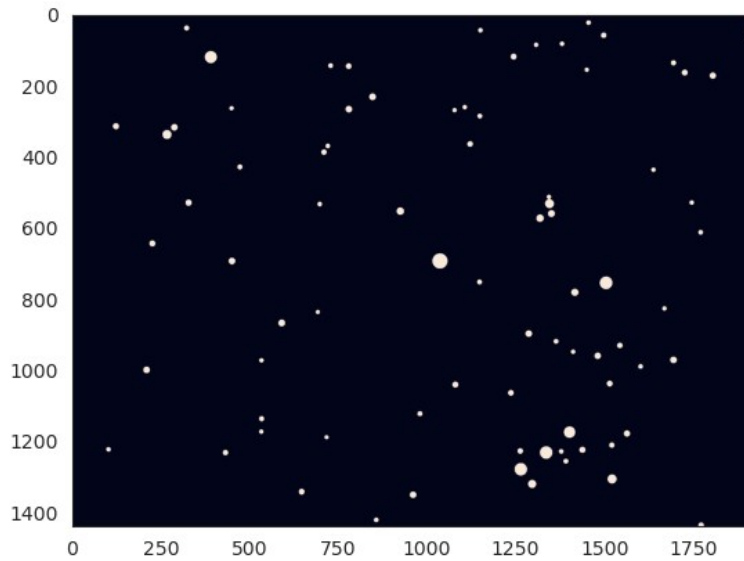
However, this process is not entirely straightforward. There are thousands of stars in the sky, all appearing as points of light. Therefore, it is necessary to compare patterns of stars, constellations, and geometric relationships between stars, similar to how a human can recognize Orion or the Big Dipper in the night sky using previous pattern based knowledge. Additional challenges arise from the limited FOV of the camera, noise caused by positional or magnitude uncertainties, and errors such as planets or satellites being mistakenly identified as stars, or different star patterns appearing very similar to each other [8][4]. These challenges are discussed in more detail in Section 2.4.

Arguably, the operation of the star recognition algorithm is the most critical component of a star tracker [17]. Of course, it is important that the camera functions properly, withstands the harsh conditions of space, and captures accurate images. However, the key functionality resides in the recognition algorithm itself. For this reason, research into star recognition algorithms has increased significantly in the 21st century, with new and improved algorithms being published frequently. Modern algorithms aim to optimize speed to provide real-time attitude information without delay caused by spacecraft motion, improve robustness to handle errors and avoid incorrect attitude estimates, and maximize accuracy to achieve precise attitude determination [9][18].

## 2.2 Overview of Star Recognition Algorithms

There are numerous star recognition algorithms; however, in terms of operation, they generally follow a similar path. First, the star tracker camera captures an image of the star field. From this image, stars are detected, and their positions are extracted, for example, into a cartesian coordinate system within the image frame. Using multiple stars, unique geometric features can be formed, such as the angular separation between stars and their inter-star distances. These unique features are then compared against a pre-computed feature catalog stored onboard the star tracker computer, and the correct match for the observed star pattern is identified. In this way, the stars visible in the image can be associated with entries in the celestial coordinate system through the catalog. The process can be simplified by dividing it into two phases; first, feature extraction; and second, catalog search. At different stages of the algorithm, various strategies can be employed. For example, the way inter-star features are defined and compared can vary significantly, as can the number of stars considered or the choice of which geometric properties to use [9]. Another important distinction lies in the database lookup step: the classical way, where algorithms rely on an onboard catalog in which the features are stored, whereas more modern approaches can eliminate this process entirely, for example, through the use of machine learning and neural networks [17].

This provides a useful basis for dividing algorithms into two broad categories: those that do not use machine learning or neural networks and those that do. In the following sections, both of these categories will be explored in more detail, with emphasis on classical algorithms, given their importance for the simulations and comparisons carried out later in this thesis.



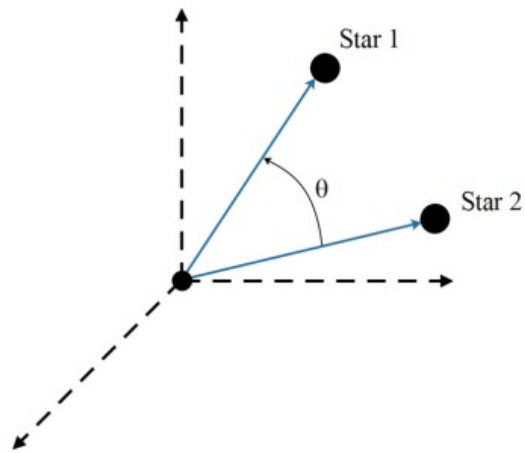
**Figure 2:** Typical star tracker image containing a part of Orion constellation in cartesian coordinate system.

### 2.2.1 Classical Geometry-Based Approaches

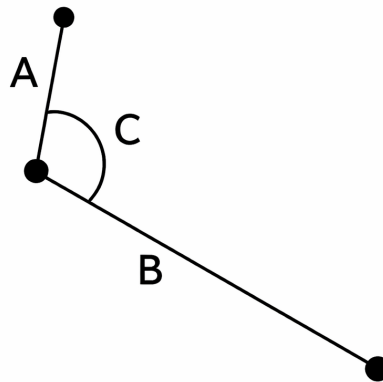
Feature- or pattern-based classical algorithms are among the earliest methods used in star trackers. Such algorithms were first proposed in the late 1970s, for example, by Salomon in 1976 [19] and Junkins and Strikwerda in 1977 [20], who are cited as the earliest users of star pattern recognition [16]. In these approaches, the internal angles of star triangles were used as the main feature. However, these early algorithms were unable to operate without a coarse prior estimate of the spacecraft’s attitude.

The first algorithm capable of solving the full lost-in-space problem was developed by Liebe in 1992 [5], which used the angular separations of the two nearest stars and the angle between them as feature parameters. Since then, several different capable algorithms have been introduced, including the Planar Triangle Algorithm [8], the Pyramid Algorithm [7], Planar Triangle Principal Component Analysis (PTP) [2], and the Grid Algorithm [21]. Each of these methods relies on slightly different techniques to extract and utilize features.

As mentioned earlier, the operation of these algorithms generally consists of two stages. The first step is feature extraction, where features are formed between the visible stars detected in the camera image and are represented in a two-dimensional cartesian coordinate system, as seen in Figure 2. When, for example, the  $x$  and  $y$  coordinates of the stars have been determined, it becomes possible to construct a variety of geometric features between them, which are later required in the catalog matching stage. The simplest feature is the angle formed between two stars and the camera, known as the angular separation, visualized in Figure 3, which is also one of the parameters used in Liebe’s algorithm.

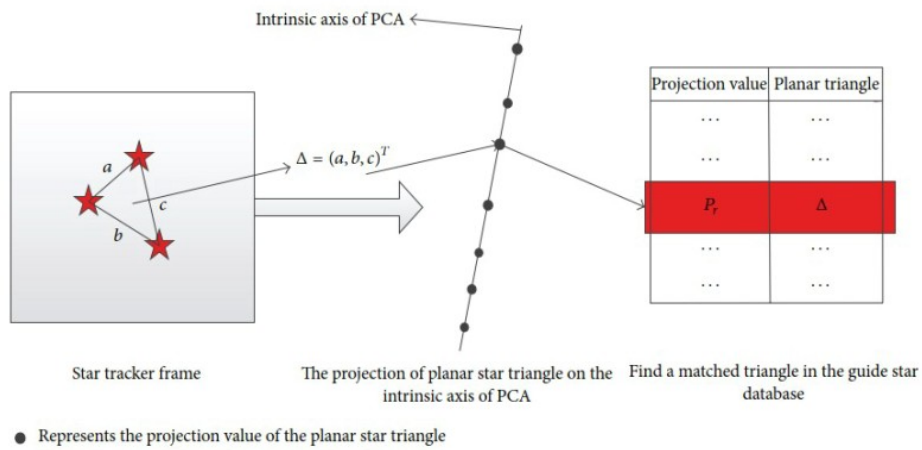


**Figure 3:** Geometry of Separation Angle ( $\theta$ ) between two stars as seen by the camera. The camera is located at the origin. [9]



**Figure 4:** Liebe's feature triangle, consisting two separation angles **A** and **B** (separation angle is illustrated in Figure 3) , and one internal angle **C**. [5]

By increasing the number of stars to three, the stars can be considered the vertices of a triangle, greatly expanding the number of possible features. For example, in Liebe's algorithm, one of the internal angles of the triangle is used alongside two angular separations, as illustrated in Figure 4. In addition to triangle angles, other invariants, such as the triangle area or the polar moment, can also be employed as features, as in the Planar Triangle Algorithm. Another option is to use the triangle side length vectors, which can be calculated using the Euclidean distance. This approach is used, for example, in the PTP algorithm, where a vector is formed from the triangle's sides, and its projection is calculated using Principal Component Analysis (PCA). This process is visualized in Figure 5.



**Figure 5:** Illustration of the projection geometry used in the Planar Triangle Principal (PTP) algorithm. [2]

The number of stars used does not need to be limited to three; algorithms can also utilize four or more stars in constructing features. For instance, in the Pyramid Algorithm, three stars are first selected to form a triangle, and the internal angles of this triangle are calculated and used as the first set of features. A fourth star is then chosen as a reference star, and the angles between this reference star and the stars of the triangle are used as three additional unique features. A variation of this method, the Triangular Dipyrmaid Algorithm [9], extends the concept by using five stars instead of four, where two reference stars form an additional six angles with the stars in the triangle, adding six more unique features on top of the internal angles of the triangle. It can be observed that there are no strict general rules for feature formation, and a wide variety of solutions have been explored, both in terms of the number of stars used and the methods applied.

In the second stage, the extracted features are matched to the star coordinate system using onboard databases that are generated from star catalogs, such as Tycho-2 [22], Hipparcos [23], or the Bright Star catalog (BSC) [24]. Catalogs are essentially lists containing all precomputed feature patterns of the visible stars, along with their corresponding star identifiers or coordinates. The catalog must include all stars expected to be visible in the image captured by the camera. If the image contains many stars that are not present in the catalog, their identification becomes impossible [1].

In catalog construction, the feature values are generated in the same way as in the first stage of the algorithm. For example, if the angles of a triangle are used as unique features, the catalog must contain the angles of all triangles formed by the stars. However, the number of combinations is typically limited; not all stars are combined with every other star. Instead, constraints are applied, such as the maximum angular distance corresponding to the camera's FOV, and a minimum distance to avoid nearly overlapping stars [4]. In this way, it is possible to construct all valid triangle combinations that could appear in the camera image, regardless of where the camera is pointed in space.

Once a feature is extracted from the image during the first stage, it is compared against the catalog. The simplest method is a brute-force search, where the list is scanned sequentially and the feature values are compared. Another, faster search algorithm is the k-vector technique by Spratling and Mortari [25]. When a match is found, the corresponding star identifiers in the catalog provide the link between the image features and the celestial coordinate system, and the spacecraft attitude can be derived.

Classical methods remain widely used and benefit from a strong space heritage, but they also suffer from recurring challenges. Different star groups can produce nearly identical features; they are sensitive to false or missing stars, and large catalogs introduce practical issues, such as memory constraints and long search times. Although these geometry-based algorithms continue to form the backbone of most operational star trackers, recent years have seen an increasing interest in machine learning approaches that aim to address some of these limitations.

## 2.2.2 Machine Learning Based Models

Recent advancements in neural networks and artificial intelligence have enabled new approaches to solving the lost-in-space problem alongside traditional classical algorithms. A neural network is a machine learning model designed to mimic the functioning of biological brains. It consists of layers of nodes that are interconnected. Each connection between nodes is weighted, and these weights are updated during network training using backpropagation. By providing input data, the network can be trained to recognize complex patterns or make predictions [26].

The idea of utilizing neural networks for star identification is not new; it was first proposed as early as 1988 [27]. However, the first actual algorithm was developed by Hong and Dickerson in 2000 [28]. At the time, the hardware capable of running such an algorithm was not suitable for deployment on spacecraft. It is only in recent years that the use of neural networks in star identification has grown, thanks to advancements in both software and computational power.

The utilization of machine learning in star identification has clearly become a growing trend over the past five years, with numerous studies published on the topic. Notable examples include: "Efficient Star Identification Using a Neural Network" [1], "A Convolutional Neural Network Approach to Star Sensor Image Processing Algorithms" [29] and "Real-Time Convolutional Neural Network-Based Star Detection and Centroiding Method for CubeSat Star Tracker." [30] These studies represent just a small fraction of the algorithms that have used neural networks for star identification in recent years. The use of neural networks in star identification is highly appealing, as neural network-based algorithms eliminate the need for database searches, resulting in constant search times. This characteristic makes them practical for onboard star trackers, where computational efficiency is crucial [1].

A representative example is the study "An artificial intelligence enhanced star identification algorithm" by Hao Wang and John Crassidis [33], where the DCNN approach was used as an image classification problem. Instead of relying on geometric features, the raw star image is fed directly into a neural network that has been trained on a large database of labeled star images, where each label corresponds to the correct sky region or star pattern.

Through its convolutional layers, the DCNN automatically learns features of the star image, such as local intensities and spatial arrangements, without the need for manual feature design (such as triangle angles or angular separations). Pooling layers contribute to translational and rotational robustness, while deeper layers capture increasingly abstract representations of star groups.

Once trained, the DCNN outputs a direct prediction of which stars or star patterns are present in the image, effectively eliminating the traditional feature extraction and catalog lookup stage, since the network already contains the catalog information internally after training. As a result, the algorithm achieves significantly higher speed as no time is spent on catalog searches. Neural networks can also be trained to tolerate

common error sources in star recognition, such as false stars, as well as magnitude and positional noise. Nevertheless, challenges remain: training requires extensive datasets, current memory demands are high, and most demonstrations have been limited to simulations rather than operational spaceborne systems.

## 2.3 General Algorithm Pipeline

In the previous sections, the lost-in-space problem was introduced, and a general overview was provided of how star trackers and their algorithms operate. While the recognition process follows the same overall steps, the details of these steps may vary across algorithms, except in the case of machine learning–based methods, which differ more fundamentally from classical approaches. The purpose of this chapter is to examine the algorithmic process in greater depth, progressing step by step from image acquisition to the production of a verified pointing solution.

### 2.3.1 Image Acquisition and Preprocessing

The image is the starting point of the entire recognition process, and it can be obtained in two ways: either from a satellite’s star tracker using a real hardware camera or, as in the case of this study, through simulation software that emulates the camera. The physical properties of the camera largely determine the characteristics of the captured images and, consequently, influence the performance of the algorithm. The most important parameters of the image are the field of view (FOV), magnitude limit, and pixel resolution. Simulations also possess these physical properties, although they are not exactly the same as real hardware.

The FOV defines how much of the sky is covered by a single image. A wider FOV allows more of the sky to be visible, which, in turn, increases the number of stars that may be included in a single frame. Conversely, a very narrow FOV (e.g., a few degrees) leads to fewer visible stars. Typical star tracker cameras on satellites have a relatively wide FOV, usually in the range of 15–30°, as demonstrated by several commercial units such as the TERMA T3 Star Tracker [31] and the Jena Optronik ASTRO CL [32]. Wider FOV ensures that multiple stars are captured in each image. However, there are also use cases where star recognition is performed with a much smaller FOV; for example, in the analysis of telescope images. The FOV is arguably one of the most important parameters of a camera, as it directly impacts both the design of the catalog and the operation of the recognition algorithm.

The magnitude limit determines the number and type of stars that appear in the image. Each star has a registered brightness, or magnitude, which provides a measure of how bright the celestial body appears. The magnitude scale is logarithmic, and the lower the numerical value, the brighter the object. For example, Venus can reach a magnitude of  $-5$  at its brightest, while Sirius, the brightest star in the night sky, has a magnitude of  $-1.46$ . In terms of a star tracker, the higher the magnitude limit, the dimmer the stars the camera can detect. This parameter is critical: ideally, the camera

should capture a sufficient number of stars, but not so many that the catalog becomes too large and problematic; a topic discussed further in the next section.

Finally, pixel resolution describes the number of pixels available in the image. Resolution does not directly affect the operation of the algorithm but primarily influences the precision with which stars can be registered as (x,y) coordinates. In very low-resolution images, a star may cover several tens of pixels, making precise centroiding difficult. Fortunately, various preprocessing methods have been developed to address this issue.

Preprocessing is the stage where the work transitions from the physical hardware to the software side of star trackers. The primary tasks in preprocessing are detecting stars in the image and determining their (x,y) coordinates within the frame through centroiding. In addition, centroiding methods can provide reasonably accurate estimates of the apparent magnitudes of the detected stars [34], which are later useful for filtering and solution validation.

Star centroiding algorithms form a large research field of their own, and numerous methods have been proposed. These can generally be divided into two categories: gray-scale centroid methods and fitting methods [35]. Gray-scale centroid methods determine the center of a star spot by calculating its center of gravity using the center of gravity algorithm (CG) [36]. In this approach, the centroid coordinates are obtained from the weighted sum of the pixel intensity values within the star spot, as shown in Equation 1 [35]:

$$(x_c, y_c) = \left( \frac{\sum_i W_i I_i x_i}{\sum_i W_i I_i}, \frac{\sum_i W_i I_i y_i}{\sum_i W_i I_i} \right). \quad (1)$$

Here  $i$  represents the  $i$ -th pixel,  $I_i$  is the gray value of the pixel,  $x_i$  and  $y_i$  are the coordinates of the pixel, and  $W_i$  is the weight of the  $i$ -th pixel. The alternative approach is the fitting method, where the centroid is determined by fitting the intensity distribution of the star spot to a Gaussian function (GF) [37]. This method models the point spread function of the star image and can achieve sub-pixel accuracy. These two approaches are the most common in star centroiding, but various other methods can also be implemented to determine coordinates within the FOV, as well as their magnitudes. The next step in the recognition process is the feature extraction and matching phase. However, to better understand the algorithm design, the following section will first examine the construction and role of star catalogs, after which we will return to the details of the algorithm's operation.

### 2.3.2 Star Catalog and database

Star catalogs and databases are a critical part of star recognition algorithms. They reside on board the satellite and contain all the reference information needed for star identification. The terminology can vary somewhat, and catalogs and databases are occasionally used interchangeably. Typically, however, a star catalog refers to a list of stars compiled from astronomical surveys. For example, the Tycho-2 catalog

contains information on approximately 2.5 million of the brightest stars, including their coordinates and magnitudes [22].

While such catalogs provide the fundamental data, a simple list of coordinates and magnitudes is not sufficient for the matching phase. For this purpose, a star database is constructed. Database stores the precomputed features required for recognition and matching. In the design of the system, the database may either directly include all the necessary coordinate information, which eliminates the need for a separate catalog, or it may be linked to the catalog through star indices. In the latter case, the database provides indices for candidate stars, while their detailed properties are retrieved from the catalog using those indices.

There are several well-known star catalogs commonly used in recognition algorithms, such as the previously mentioned Tycho-2 catalog [22], the Bright Star Catalog (BSC) [24], and the Hipparcos catalog [23]. Their sizes vary considerably: Tycho-2 is the largest, containing about 2.5 million stars; Hipparcos is significantly smaller, with around 118,000 stars; and BSC includes only slightly more than 9,000 stars. In practice, however, the number of stars included in an algorithm’s onboard catalog is often further reduced to keep the database size manageable. For example, the catalog constructed for Liebe’s algorithm contains 1,539 of the brightest stars [5], while the catalog for the Planar Triangle Principal Component (PTP) algorithm includes 5,103 stars [2], both of which are much smaller than the full astronomical catalogs mentioned.

The appropriate size of a catalog is closely linked to the properties of the star tracker camera, in particular, the FOV. In any randomly oriented image, it is necessary for a sufficient number of catalog stars to be visible to enable feature formation and matching. In theory, for triangle-based algorithms, three stars would be enough to form a feature. In practice, however, many similar triangles may be found in the database, which introduces ambiguity into the identification process. For this reason, it is preferable for more stars to be visible in each image. On average, having more than ten catalog stars in each frame significantly improves the reliability of identification.

The average number of catalog stars visible in a given image can be estimated directly from the catalog size and the FOV using Equation 2 [6]:

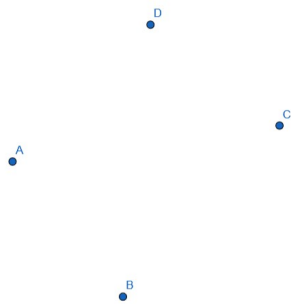
$$N_{\text{FOV}} = \frac{N_{\text{cat}} - N_{\text{cat}} \cdot \cos\left(\frac{\alpha}{2}\right)}{2}. \quad (2)$$

Here  $N_{\text{cat}}$  denotes the total number of stars in the catalog,  $\alpha$  is the circular FOV of the camera, and  $N_{\text{FOV}}$  is the number of catalog stars in the FOV. Applying this formula to an example case similar to Liebe’s algorithm, with a catalog of 1539 stars and a FOV of 30, yields an average of approximately 26 catalog stars per image. If the FOV is halved to 15 and the catalog size is kept the same, the average number of visible catalog stars is decreased to around 6. Naturally, exceptions occur since the brightest stars are not uniformly distributed across the sky. But from this example, we can see that in design it is crucial that FOV and catalog size are kept reasonable in relation to

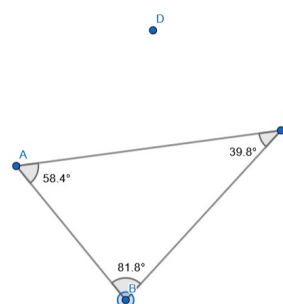
each other.

Once the catalog size has been defined and the stars selected, the next step is to construct the database from the star catalog, which is the more critical component for the matching phase. The database must directly contain the chosen features, such as the internal angles of triangles formed by stars, their areas, or the ratios of their sides. For example, an algorithm that uses triangle angles as unique features requires a database that includes all reasonable combinations of catalog stars that can be used to form such angles.

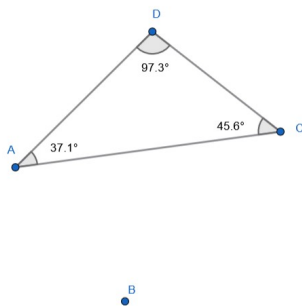
Naturally, some logic must be applied when generating these triangles. It is not useful to construct feature triangles from stars located on opposite sides of the sky, since they could never appear together in the FOV of a star tracker camera. Instead, triangles should be built according to certain criteria, such as a maximum separation equal to the camera's FOV, ensuring that all three stars could realistically appear in the same image. Similarly, triangles that are too small or produce excessively acute or obtuse angles can be discarded, retaining only valid triangles for the database.



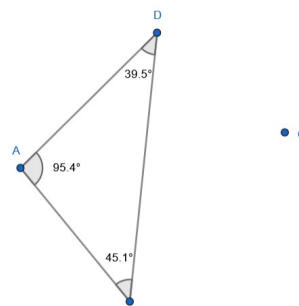
(a) Stars labeled A, B, C, and D.



(b) First feature triangle for star A.



(c) Second feature triangle for star A.



(d) Third and last triangle built for star A.

**Figure 6:** Visualization for building feature triangles for database.

To illustrate, consider a situation where star A has three neighboring stars B, C, and D within an acceptable distance. In this case, three different triangles can be formed using star A with each of the others and their corresponding angle features, as shown in Figure 6. Once all valid combinations have been generated, they can be tabulated in the database. Database entry for the triangle in the second picture (b) in figure 6 could be, for example, something like: (58.4, 81.8, 39.8), (A,B,C). In this database entry, the feature values appear first, followed by the identifiers of the stars involved. In this example, the identifiers are the letters A,B,C, and D, which can then be used to search the catalog and retrieve the corresponding star coordinates. Such triangle construction must be performed for all stars in the catalog, and when the catalog size is large and the FOV is wide, each star can form dozens of valid combinations. As a result, the database may easily contain tens or even hundreds of thousands of entries.

The size of the database is particularly important. A database containing thousands of features inevitably includes nearly identical entries, often differing only by a few decimal values. On the other hand, a larger database increases the probability of including all possible feature configurations. Database size also directly affects search time and memory requirements. Therefore, finding an appropriate balance is a key consideration during the design phase.

### 2.3.3 Feature Matching

Once the preprocessing stage, as well as the construction of the catalog and database, is complete, the algorithm proceeds to the actual matching strategy. In the first step, all possible triangle (or other polygon) combinations are formed from the detected stars in the image, which are then used for feature extraction. These features can include, for instance, the angles of the triangle, as illustrated in the database example from the previous section. Once a numerical feature value has been computed, it can be directly compared with the precomputed database to search for matches.

There are essentially no strict rules regarding which features may be selected; in principle, any geometric property that can be derived from the relations between the stars can serve as a unique feature. However, features are usually chosen such that they remain invariant to rotation and scale, allowing for unique identification without requiring prior knowledge of the camera's FOV or orientation. For example, using distance in degrees is less favorable, as this requires prior information on the camera's FOV and a catalog constructed with that scale in mind.

Commonly employed features include internal angles of star triangles, areas computed from unit vectors, ratios of triangle side lengths, or any combination of multiple different features. These remain consistent regardless of image rotation or scaling. Feature extraction within the algorithm follows the same logic as that in database construction, with the difference being that, in this stage, the true celestial coordinates of stars are unknown and only their in-frame pixel coordinates are available.

As an illustrative example, detection yields three detected stars with known  $(x,y)$  coordinates. To compute their internal angles, vectors corresponding to the triangle's sides are first formed between the coordinate points:

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}. \quad (3)$$

The angle between two adjacent sides can then be computed using Equation 3, where the dot product  $(\vec{a} \cdot \vec{b})$  of the vectors  $\vec{a}$  and  $\vec{b}$  is divided by the product of their magnitudes  $(|\vec{a}||\vec{b}|)$ , yielding the cosine of the angle  $\theta$ . This procedure is repeated for all three vertices of the triangle, and a uniquely chosen feature is obtained. Now, these angles can be formed for a database search, similar to what was done in the database example,  $(\theta_1, \theta_2, \theta_3), ((x_1, y_1), (x_2, y_2), (x_3, y_3))$ . If multiple triangles are formed, feature extraction is performed for each triangle individually. The database matching step is then carried out for all of these triangles, which often results in multiple candidate matches being associated with the same set of stars.

In many algorithms, a tolerance margin is applied to the comparison, meaning that the feature does not need to be perfectly identical to its counterpart in the database. Instead, a small deviation is allowed depending on the camera's resolution, preprocessing accuracy, and the level of positional noise [5]. Naturally, the larger the allowed tolerance, the higher the number of potential matches for a given star pattern. It is also possible that even with a small tolerance, multiple matches may still exist, especially when the database is large.

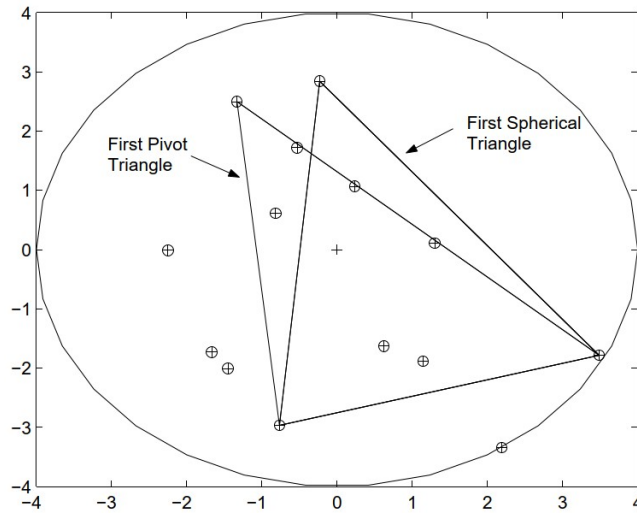
For this reason, the candidate matches obtained in this stage must undergo an additional filtering process to ensure that the correct match is found. This step will be discussed in the next part.

### 2.3.4 Filtering and pivoting

Since the number of matches per image feature pattern can range from dozens to even hundreds, it becomes essential to distinguish the correct matches from the false ones. To achieve correct detection, various techniques can be used, including pivoting and different filtering methods.

The idea behind filtering is to eliminate incorrect matches based on additional information about the stars and their geometric relationships. One common approach is to extract new feature parameters from the stars that were not used during the initial matching phase, as it is statistically unlikely that false matches would share identical values for these additional features.

Several filtering strategies were tested in the simulations described in chapter 3. A significant method is the comparison of distances between stars in the feature pattern. Using the celestial coordinates from the database, the angular separation between two stars can be computed. If the image scale is known, pixel distances can be converted to degrees, allowing direct comparison between the image and the database. Scale



**Figure 7:** First triangle and pivot triangle with two anchor stars. [8]

can be calculated if the image's resolution and FOV is known. This helps distinguish patterns that appear geometrically similar but differ in scale, thereby eliminating false matches.

Another filtering method is magnitude filtering [38], which involves comparing the stars' magnitudes between the image and the matched entries in the database. If the observed stars' magnitudes are consistent within a defined error margin, the match is more likely to be correct. However, this approach requires reliable magnitude estimation during the preprocessing stage.

A third approach is to analyze the spatial distribution of the matched coordinates. If multiple feature patterns in the same image each yield sets of candidate matches, the estimated celestial coordinates can be compared to identify clustering in a specific sky region. If multiple matches converge in the same area, it strengthens confidence in that region as the correct solution. However, this method becomes less reliable when the number of matches is large, as random coincidences can produce misleading clusters, particularly because stars are not uniformly distributed across the celestial sphere. These filtering methods will be discussed in more detail in Section 3.2.2.

Pivoting differs slightly from filtering in that it is performed during the matching stage, particularly when the initial round of matching produces more than one possible solution. In the pivoting process, when examining a single feature pattern, one of the stars is replaced with another star from the image, while the remaining stars are kept fixed, as shown for the triangle in figure 7. As a result, two feature patterns are formed that share common stars but differ by one. The corresponding matches from both triangles can then be compared to determine whether the same predicted celestial coordinates or index numbers appear for the fixed stars in both cases. If multiple potential solutions still remain after this comparison, the pivot star is again replaced with another candidate from the image. This iterative process continues until either no suitable replacement stars remain or a single unambiguous solution is found [8].

### 2.3.5 Verification and Estimation

In the final stage of the algorithm, the verification and estimation of the camera's pointing are performed. A typical verification method is to check whether the estimated coordinates are consistent with the observed image. Once several feature patterns have been identified and matched, and their estimated celestial coordinates are known, these coordinates can be projected back onto the image to assess their alignment. By comparing the logical consistency of one pattern's projected coordinates with those of others, the algorithm can evaluate whether the solution is coherent. If the coordinates are mutually consistent, the estimated attitude and pointing can be assumed to be correct [4].

The final step in the estimation phase can be accomplished by solving Wahba's problem [39], which aims to determine the rotation matrix that best aligns the observed star vectors in the camera frame with their corresponding reference vectors in the celestial coordinate frame. Several analytical and numerical methods exist to solve Wahba's problem efficiently. Among the most widely used are the QUEST algorithm [40] (Quaternion Estimator) and the Davenport q-method [41]. The resulting rotation matrix directly represents the estimated spacecraft attitude relative to the celestial reference frame.

## 2.4 Challenges in Star Recognition

Star recognition also faces various challenges. The performance of the algorithms is influenced by both functional and external factors. Commonly cited challenges include FOV limitations [2], false and missing stars [8][42], database feature ambiguity [9], image noise, and centroiding errors [4]. These are among the most frequently discussed issues in the literature, and they represent the main focus of algorithm design. Consequently, these same challenges are also emphasized in this chapter and throughout this thesis.

Of course, these are not the only factors affecting the performance of star sensors and their algorithms. Environmental and operational conditions can also introduce difficulties. Space is a harsh environment where temperature variations [43] and radiation [44] can degrade optical components and sensors. The motion of the spacecraft may additionally cause motion blur [45], especially when the camera's exposure time is relatively long, making it more difficult to accurately identify star positions in the image. However, such environmental effects are not considered central to the algorithmic design process and are therefore only briefly mentioned here.

Arguably, one of the most critical parameters affecting a star recognition algorithm is the FOV, which determines how many stars are visible in a single image. As discussed in Section 2.3.2, the size of the FOV is directly related to the required size of the star catalog. When the FOV is small, the number of stars included in the catalog must be increased to ensure that enough stars are visible in each image for reliable identification.

Reducing the FOV can also lead to extreme cases where only a few stars are visible, making it impossible to execute the algorithm. With only a limited number of visible stars, the number of possible feature patterns that can be formed from the image is also restricted, resulting in decreased matching confidence [2]. Additionally, a larger catalog implies a larger database, which, in turn, leads to more stored feature patterns. This increases feature ambiguity and extends the algorithm's runtime. In other words, with a small FOV, there are fewer available features from the query image; however, there are too many potential matches in the database, increasing both processing time and the number of incorrect identifications.

Conversely, a very large FOV can also introduce problems due to optical distortions [46]. The sky can be considered the inner surface of a sphere, and when the FOV becomes large, projecting this curved celestial sphere onto a two-dimensional image causes distortions, especially near the image edges. These distortions alter the geometric relationships between stars, making the formation of accurate feature patterns more difficult.

Another common challenge in star recognition is the presence of missing or false stars. Missing stars typically occur when the camera sensor or preprocessing stage fails to detect certain stars [42]. This may result from partial occlusion by the Earth, the Moon, or cloud cover, depending on the operational environment. While missing stars generally reduce the effective FOV and the number of usable reference points, they are usually not critical to the algorithm's overall performance.

False stars, on the other hand, represent a more problematic case. These are bright objects incorrectly identified as stars during preprocessing, often caused by other celestial bodies or artificial satellites. Common examples include planets or passing satellites, which appear as bright point sources similar to stars but differ in their motion characteristics. Unlike stars, such objects move relative to the celestial background and, therefore, cannot be used for reliable attitude determination [8].

The main challenge arises when false stars are used in the formation of feature patterns. If corrupted patterns containing false stars are matched against the database, they inevitably lead to incorrect identifications and increased ambiguity. One strategy to mitigate this issue is to design algorithms that do not attempt to identify every star in the image; instead, they accept that a fraction of the detected objects may be invalid. This approach, however, requires a sufficient number of true stars to remain visible for a successful solution. Another, less common approach is to take images at short intervals and identify moving objects by comparing consecutive frames, allowing non-stellar detections to be discarded [47].

Image noise and centroiding errors are among the most common challenges in real star sensors. Magnitude and positional noise can cause stars to be inaccurately localized in the image, leading to small deviations in their measured positions and brightness. These errors alter the relative distances and angular separations between stars, which, in turn, degrade the reliability of the extracted geometric features [5].

Centroiding errors, in particular, arise when the preprocessing stage fails to estimate the star's precise position within the image frame. Even small deviations

in centroid estimation can distort the calculated geometric relationships between stars, reducing the consistency between the observed features and those stored in the database. Similarly, errors in magnitude estimation can introduce problems if the algorithm uses brightness information for filtering or verification by comparing image magnitudes to database entries.

To mitigate these effects, most algorithms incorporate tolerance thresholds, allowing a small margin of error ( $\pm\Delta$ ) between the observed and reference values [5]. However, increasing the tolerance range comes at the cost of reduced selectivity, meaning that larger tolerances lead to a higher likelihood of multiple database entries matching a single observed feature, thereby increasing ambiguity and the potential for false matches.

The final major challenge is database ambiguity, which was also discussed earlier in Section 2.3.2. In this context, database ambiguity refers to the fact that in large catalogs and databases, the sheer number of stored feature patterns inevitably leads to some of them being nearly identical. This problem becomes more pronounced when other algorithmic challenges, such as noise and limited FOV, are present. A limited FOV requires a larger catalog and database to ensure sufficient star visibility, while noise forces the algorithm to accept feature patterns that deviate slightly from their ideal values. Both effects increase the number of potentially overlapping feature entries, making it more difficult to achieve unique matches during identification.

However, this challenge can be mitigated through filtering techniques, as discussed in Section 2.3.3. By applying additional geometric or photometric constraints, such as magnitude comparison or distance ratio filtering, many incorrect or redundant matches can be eliminated before the final verification step. Another effective approach is to construct more complex feature patterns that incorporate a greater number of stars when forming each feature. By using larger star groups, the likelihood of identical or near-identical features appearing in the database is significantly reduced. However, this approach also increases the requirement for visible stars in each image, making it less suitable for situations where only a few stars are detectable, along with an increased search time [9].

## 2.5 Algorithms in this Thesis

In the final part of this theory chapter, two algorithms that are implemented in this thesis are examined in greater detail. For both algorithms, the following sections aim to clarify how each method utilizes feature patterns, how their databases are constructed, and what key design considerations were taken into account during their development. These algorithms are Liebe's Triangle Algorithm [5][6] and the Planar Triangle Algorithm developed by Crassidis [8], both of which are implemented and simulated in this thesis. Section 2.5.1 introduces Liebe's triangle algorithm, followed by Section 2.5.2, which discusses the Planar Triangle Algorithm and highlights their key differences.

### 2.5.1 Liebe's Triangle Algorithm

As mentioned earlier in section 2.2.1, Carl Christian Liebe published the first star recognition algorithm capable of operating without prior attitude information in 1992, in his paper “*Pattern Recognition of Star Constellations for Spacecraft Applications*” [5]. The algorithm was designed to determine a spacecraft's attitude using data from a star tracker. Its operation closely follows the general structure of classical geometry-based methods: first, the stars are detected in the camera frame, and then geometric feature patterns are formed between them. These features are subsequently compared to an onboard database to identify the corresponding celestial coordinates of the observed stars.

The algorithm is based on the properties of a triangle formed by three stars. This subconstellation with specific features is called a star triplet, and its parameters are referred to as Triangular Features (TF). To form these TFs, three measurements are extracted from each star triplet:

1. The angular distance to the first neighboring star.
2. The angular distance to the second neighboring star.
3. The angle between the two neighboring stars.

These features are illustrated in Figure 4 in Chapter 2.2.1. And for building the database, the same TF method is naturally applied and repeated for all stars in the catalog. However, the database entries are quantized to minimize memory requirements and reduce processor load. Therefore, the measured angular distances are divided by  $0.1^\circ$  (6 arcminutes), and the internal angle between the two neighboring stars is quantized in  $5^\circ$  intervals by dividing the angle by  $5^\circ$ . For example, angular distances of  $5.596^\circ$  and  $8.191^\circ$ , and an internal angle of  $164.204^\circ$ , would be represented as 55, 81, and 32, which can be written as the vector [55, 81, 32].

Due to positional uncertainty, a single identifier is not sufficient for one star. Therefore, the distances must be transformed into class numbers. For instance,  $5.596^\circ$  would correspond to the class numbers 54, 55, and 56, while  $8.191^\circ$  would correspond to the class numbers 80, 81, and 82.

Uncertainty in the angular separation between stars must also be considered, and it is calculated using an angular uncertainty of  $\pm 1.724^\circ$ . The resulting angle range is:  $[164.204^\circ - 1.724^\circ; 164.204^\circ + 1.724^\circ]$ , which corresponds to the class numbers 32 and 33. As a result of these positional uncertainties, each star must have all possible combinations of these class numbers in the database. The list of required entries in this example can be seen in Figure 8.

When building the algorithm's database for stars, it is important to account for uncertainties related to star detection. Due to positional or magnitude noise, the camera may fail to detect certain stars, or the preprocessing stage may incorrectly identify them. For this reason, the stars are classified into three categories: non-detectable stars, detectable stars, and possibly detectable stars. Non-detectable stars are those that

54	, 80	, 32
54	, 80	, 33
54	, 81	, 32
54	, 81	, 33
54	, 82	, 32
54	, 82	, 33
55	, 80	, 32
55	, 80	, 33
55	, 81	, 32
55	, 81	, 33
55	, 82	, 32
55	, 82	, 33
56	, 80	, 32
56	, 80	, 33
56	, 81	, 32
56	, 81	, 33
56	, 82	, 32
56	, 82	, 33

**Figure 8:** List of different combinations required due to positional uncertainty for a triplet star. First column indicates quantized angular distance to first star, second column is the quantized angular distance to second star and last column indicates quantized angle between two stars. [5]

cannot be observed by the camera under any circumstances, even in good conditions, due to their physical limitations; therefore, they are excluded from the database. Detectable and possibly detectable stars, however, are those that may appear in the image. Detectable stars will almost certainly be visible, while possibly detectable stars have magnitudes close to the camera's detection limit and may or may not be observed depending on conditions.

As a result, when the algorithm searches for neighboring stars, a neighbor may belong to either of the two selected categories, and it may not always be the same star. Therefore, the algorithm must create several variations for each catalog star, where the neighboring stars vary according to their uncertainties. This leads to a situation where a single catalog star can have nearly 200 entries in the database. Through this process, the database becomes robust to error conditions in which neighboring stars in the image are incorrectly identified. Consequently, a potential match is more likely to be found.

According to the original publication, the accuracy of Liebe's algorithm is affected by the camera FOV, the number of visible stars, and the size of the constructed catalog and database. These variables are also interdependent as mentioned earlier, the FOV and the number of visible stars are linked, as is the number of catalog stars. When the FOV decreases, the number of visible stars decreases; and when the FOV increases, more stars become visible. The algorithm was designed so that, on average, about 20 stars are visible in each simulated image; although a larger number of stars does not negatively affect performance. The accuracy is also influenced by various anomalies, such as cases where the camera fails to detect a star or mistakenly classifies a non-star

object as a star. The algorithm does not provide a direct solution to these problems. For example, if a triangle feature is formed between two actual stars and a non-star object misidentified as a star, this feature will still be matched against database entries as usual. If the number of stars in the image is too small, such cases may result in an incorrect attitude estimation. A larger FOV and a greater number of visible stars help mitigate the issue, allowing for more analyzable feature triangles.

## 2.5.2 Planar Triangle Algorithm

The second algorithm examined in more detail in this thesis is the Planar Triangle algorithm, published in the article "Fast Star-Pattern Recognition Using Planar Triangles" by Cole and Crassidis [8]. Like Liebe's algorithm, the Planar Triangle algorithm also aims to solve the Lost-in-Space problem by utilizing triangular feature patterns. However, instead of relying on distances and angles in a star triplet, as Liebe's algorithm does, this method utilizes the area of planar triangles and their polar moment. Two features are used because two different planar triangles may have the same area; thus, the additional feature provides a way to distinguish between them. The algorithm is designed to operate with a relatively small FOV ( $8^\circ \times 8^\circ$ ), and its database contains 662,779 planar triangles. The exact number of stars in the catalog is not specified, but based on the database size, it can be assumed to be very large.

When building the database, the planar triangle area is calculated first. The area can be mathematically determined using unit vectors  $\vec{b}_1$ ,  $\vec{b}_2$ , and  $\vec{b}_3$ , where each vector points toward one of the three stars in the planar triangle. The area  $A$  is computed using Heron's formula 4

$$A = \sqrt{s(s-a)(s-b)(s-c)}, \quad (4)$$

where

$$s = \frac{1}{2}(a + b + c), \quad (5)$$

$$a = \|\vec{b}_1 - \vec{b}_2\|, \quad (6)$$

$$b = \|\vec{b}_2 - \vec{b}_3\|, \quad (7)$$

$$c = \|\vec{b}_1 - \vec{b}_3\|. \quad (8)$$

After solving the Planar Triangle area  $A$ , we can use it to calculate the polar moment  $J$ , which is given by Equation 9:

$$J = A(a^2 + b^2 + c^2)/36. \quad (9)$$

The matching process of the Planar Triangle algorithm follows a classical approach and operates in a manner similar to the Liebe algorithm. A star is first selected from the image, after which its two nearest neighboring stars are chosen to form a triangle. The area and polar moment of this triangle are calculated according to Equations 4 and 9. To improve the results, the standard deviation is applied to refine the calculated

polar moment and area. Once these values have been computed, they are compared to the database, and matching entries are retrieved. However, in many cases, the area and polar moment values are similar across multiple database entries, resulting in several possible matches for a single image triangle. In these situations, the Planar Triangle algorithm employs pivoting to eliminate false matches. Pivoting means forming a new triangle by keeping two of the original stars fixed and selecting a new third star from the image. The area and polar moment of this new triangle are calculated in the same way, followed by another database comparison to obtain new matches. By comparing the match lists of the original and pivoted triangles, the algorithm checks whether the same two stars appear in both sets of matches. If multiple candidates still remain, the pivoting process continues until only one valid solution is left or until no new stars are available for further pivots.

Structurally, Liebe's algorithm and the Planar Triangle algorithm are quite similar. Both algorithms utilize three stars to form feature patterns, and the matching process is performed using a catalog and a corresponding database. However, the feature patterns themselves are considerably different, as are the filtering and pivoting procedures following the matching stage. In the following chapter, the focus shifts to the implementation and simulation setup, while Chapter 4 examines how the algorithmic design choices perform under various conditions, such as the presence of noise, and compares the effectiveness of the filtering and pivoting strategies.

## 3 Algorithm Implementation and Simulations

In this chapter, we move from the general overview of star recognition algorithms to a more detailed examination of the implementation of selected algorithms in Python, as well as the use of the simulation environment. First, in Section 3.1, we examine the simulation environment, the chosen method for generating star sensor images, and the simulation settings used for algorithm testing. Then, in Section 3.2, we focus on the implementation of the algorithms and discuss the tools and design choices applied during their development. Finally, Section 3.3 presents the data and performance metrics obtained from running the algorithms, as well as how these results are collected within the simulation environment. Results and data will be discussed in more detail in Chapter 4.

### 3.1 Simulation Environment

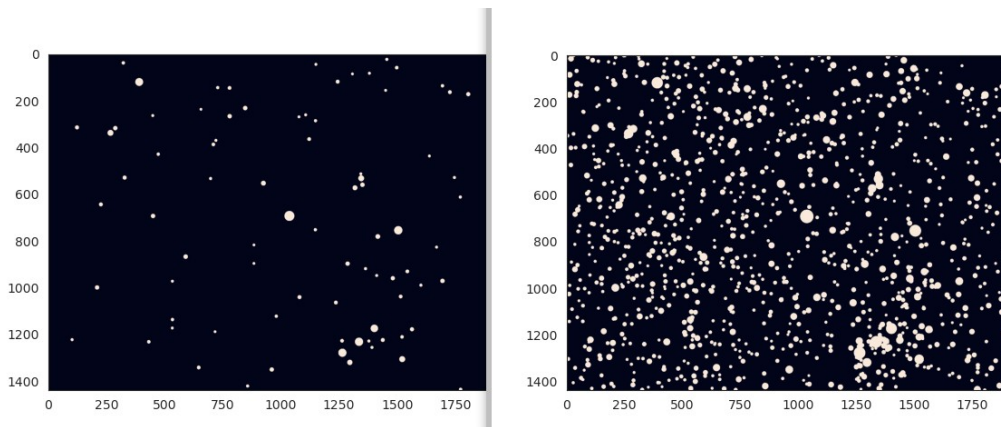
To analyze the performance of star recognition algorithms, the algorithms must first be provided with an image to process. While there are countless images of the night sky and stars available, such images are not necessarily suitable for algorithm development or comparison. The exact pointing or location of the image may be unknown, and precise coordinate information for each star may be unavailable. In addition, the image specifications, such as FOV or resolution, are fixed and cannot be adjusted. Controlling other factors, such as the number of stars (both real and false detections), is also difficult, which limits the ability to test the algorithms under various conditions. For these reasons, in a comparative study aiming for a comprehensive evaluation of different algorithms, the star sensor images used have been generated through simulation.

More details on the simulated images are presented in Section 3.1.1, followed by Sections 3.1.2 and 3.1.3, which describe the simulation scenarios used for algorithm comparison.

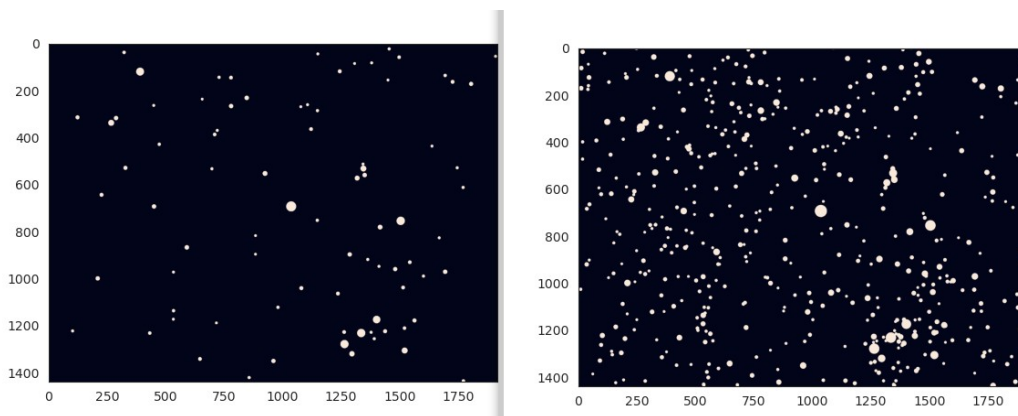
#### 3.1.1 Simulated Star Sensor Images

The software selected for simulating star sensor images is an open-source Python program originally developed for the European Space Agency (ESA) Kelvin competition [48]. The tool was designed as part of a challenge organized by ESA, in which participants were invited to submit their own star recognition algorithms. During the development phase, this simulator was provided as a support tool and testing environment for participants, making it a highly suitable choice for algorithm comparison due to its comprehensive feature set.

The simulator aims to accurately reproduce the optical behavior of a camera, allowing the user to define parameters such as lens properties, FOV, image resolution, and the visible magnitude range. It also enables the adjustment of both the number of real and false stars, as well as the inclusion of positional and magnitude noise. The query image is then produced according to these specifications.



**Figure 9:** Simulated star images generated using the Kelvin software for two exposure times: 0.2 s (left) and 2 s (right).



**Figure 10:** Simulated star images generated using the Kelvin software for two aperture sizes: 15 mm (left) and 30 mm (right).

The image simulation process is based on retrieving stars from a catalog and placing them within the simulated camera frame according to the defined simulation parameters. In essence, the procedure can be viewed as the inverse of what a star recognition algorithm performs. The simulation begins by selecting either a random or a predefined celestial coordinate as the image center, after which it retrieves all stars within the corresponding region from the Hipparcos catalog. The stars are rendered in the image as circular points, whose apparent size and position are determined by the simulated camera characteristics. For example, in addition to the star's magnitude, the optical properties of the camera influence how large the star appears in the image. Once the stars have been placed, the resulting image can be passed to the algorithm for identification. In addition, the simulator provides a corresponding list containing the stars' pixel coordinates  $(x, y)$  and their Hipparcos IDs, which can be used to verify the true catalog coordinates of each star.

The simulation includes a large number of adjustable parameters, of which the most important ones are discussed below. The key parameters are also summarized in Table 2, which lists the parameters along with a short description of each parameter’s role in the simulation. The first essential selection concerns the camera projection model, which defines how the three-dimensional celestial sphere is projected onto the two-dimensional image plane. The available projection models are Rectilinear, Equidistant, Equisolid-Angle, Stereographic, and Orthographic.

**Table 2:** Key parameters used in the star sensor image simulation.

<b>Parameter</b>	<b>Description</b>
Camera Mode	Different projection models, including: Rectilinear, Equidistant, Equisolid-Angle, Stereographic, and Orthographic
Resolution	Defines the simulated image size in pixels. Both x and y dimensions can be adjusted.
Exposure time	Duration for which the camera sensor is exposed to light. Affects detected brightness and limiting magnitude.
Aperture	Diameter of the lens opening that controls how much light reaches the sensor.
Normalized focal length	Distance between the camera lens and sensor; directly corresponds to the field of view (FOV).
Gaussian noise	Positional noise added to the star centroids in radians to simulate sensor or optical uncertainty.
Magnitude noise	Random noise added to stellar magnitudes to simulate photometric uncertainty.
min_true	Minimum number of real (catalog) stars required in the simulated image.
max_true	Maximum number of real (catalog) stars allowed in the simulated image.
min_false	Minimum number of false or noise stars added to the image.
max_false	Maximum number of false or noise stars added to the image.

The Rectilinear projection, also known as the perspective projection, maps straight lines in space as straight lines in the image. It provides high geometric accuracy and is, therefore, widely used with small FOV systems. The Equidistant and Equisolid-Angle projections are both types of fisheye projections. The Equidistant model maintains a linear relationship between the viewing angle and the image radius, although its nonlinear scaling can distort shapes. The Equisolid-Angle model, on the other hand, preserves equal solid angles, resulting in realistic brightness distribution but distorted angular distances, which makes it less ideal for geometric feature matching. The Stereographic projection is a conformal projection that preserves local angles, making it well suited for analyzing star pattern features near the image center.

However, it introduces increasing area distortion toward the image edges. Finally, the Orthographic projection maps all rays perpendicularly onto the image plane. While useful for theoretical analysis and geometric testing, it does not correspond to physically realizable optics [49] [50].

Other camera settings include parameters such as resolution, exposure time, aperture, and normalized focal length. Exposure time and aperture directly affect which magnitudes the camera is able to detect, while the normalized focal length is proportional to the camera's FOV. The effect of exposure time and aperture is visualized in images 9 and 10. The focal length refers to the distance between the camera lens and the sensor, a shorter focal length results in a wider FOV; while a larger focal length produces a narrower FOV. The camera settings also include gaussian noise models for both positional and magnitude errors, which can be increased when additional noise is desired in the simulated image. Several other parameters are related to image and magnitude modeling, such as adjustable pixel aspect ratio, image blurriness, and the photoelectron generation rate per pixel area.

The number of stars in the image is controlled by four variables: `min_true`, `max_true`, `min_false`, and `max_false`. These define the acceptable range for the number of visible stars in the simulated image. The image is only accepted if the total number of stars falls within these specified limits; otherwise, a new simulated image is chosen.

### 3.1.2 Different Simulations for Data Gathering

To enable a fair comparison of the algorithms under different environmental conditions, the simulations were performed using several distinct configuration setups. In total, eight separate simulation configurations were executed, and each configuration generated 200 random sky images. Consequently, both algorithms were run 1,600 times in total. The simulations can be divided into two main categories: normal tests and robustness tests. These tests are collected in the table 3, where the first three tests are normal and the last five are robustness tests.

In the normal test cases, the camera simulation parameters remained constant and optimal. FOV was set to  $30^\circ$ ; no noise was added, and no false stars were present. The purpose of these baseline conditions was to evaluate and compare the algorithms' nominal performance in ideal scenarios. Three different normal test runs were conducted: one with both algorithms in their standard configurations, one with a modified Liebe algorithm, and one with a modified Planar Triangle algorithm. These modified runs were designed to study how internal algorithmic adjustments affect overall accuracy and robustness. In the Liebe algorithm, the modification focused on altering the filtering strategy, while in the Planar Triangle algorithm, it involved changing the number of selected feature triangles.

In the robustness test scenarios, the algorithms themselves remained identical to the base configuration, while only the camera simulation parameters were modified to create more challenging and non-ideal conditions. A total of five robustness test cases were performed. In the first case, the camera FOV was reduced by half to  $15^\circ$ , making fewer stars visible and thus increasing the difficulty of star identification. In

the following configuration, the FOV remained small, but five false stars were added to each image to introduce false-positive recognition challenges. The next robustness test case increased the number of false stars significantly, adding 20 false detections per image. Afterward, the FOV was restored to its normal value of  $30^\circ$ , and all false stars were removed; however, a small positional noise of  $0.01^\circ$  was introduced to simulate minor camera inaccuracies. Finally, the same setup was repeated with the positional noise increased to a high level of  $0.5^\circ$ , creating a highly distorted imaging environment to test the algorithms' robustness under extreme uncertainty.

**Table 3:** Overview of simulation configurations used for algorithm comparison.

Test Name	FOV	False Stars	Noise	Other Changes
Normal	$30^\circ$	0	0	-
Altered Liebe	$30^\circ$	0	0	Different filtering
Altered Planar	$30^\circ$	0	0	Adjusted number of feature triangles.
Small FOV	$15^\circ$	0	0	-
S.FOV + Few False	$15^\circ$	5	0	-
S.FOV + Many False	$15^\circ$	20	0	-
Noise (Low)	$30^\circ$	0	$0.01^\circ$	-
Noise (High)	$30^\circ$	0	$0.5^\circ$	-

As mentioned in Section 3.1.1, the simulation also involves a number of additional parameters, such as the camera mode and aperture. These were selected as follows: the camera mode is stereographic, the resolution is  $1920 \times 1440$ , the aperture is 15 mm, and the exposure time is 0.2 s. These parameters remain constant across all simulation test scenarios.

## 3.2 Algorithm Implementation in Python

In this chapter, we move from image simulation to the actual construction of the algorithms. As mentioned earlier, the implemented algorithms require both a star catalog and a star database to function properly. The creation and operation of these components are described in Section 3.2.1. After that, Sections 3.2.2 and 3.2.3 present the functional implementation of the Liebe and Planar Triangle algorithms, respectively. Finally, other tools and utilities used in the algorithm development process are discussed.

### 3.2.1 Catalog and Database Construction

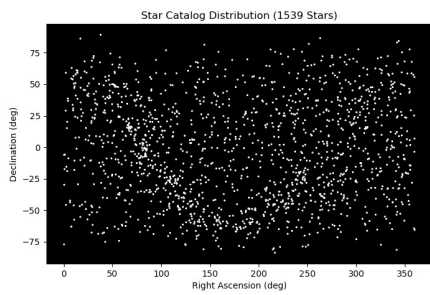
As previously discussed, the catalog and database play an important role in the construction of classical star recognition algorithms. The catalog is used to retrieve star information and serves as the basis for building the database. The database, in turn, enables the features extracted from an image to be matched with the corresponding entries in the catalog.

The completeness and size of both the catalog and the database are highly significant for the algorithm's performance. A catalog that is too small may lead to cases where stars cannot be identified simply because they are not included in the dataset. On the other hand, an excessively large catalog can also cause problems, as it results in an overwhelming number of possible feature patterns, making the matching stage ambiguous when too many stars fit the criteria.

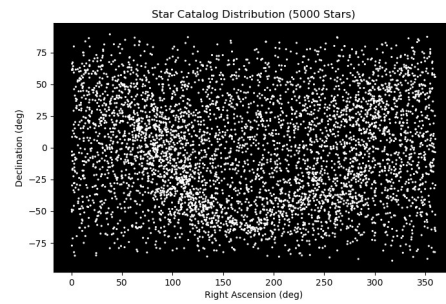
Let us begin by examining the selection and construction of the catalogs used for both algorithms. The catalogs for both implementations were built using the Hipparcos star catalog. The primary reason for this choice was that Hipparcos is already used in the image simulation phase, making it both convenient and logical to use the same catalog for algorithm development. Beyond this, Hipparcos is also a very suitable choice: it contains more than 118,000 stars, which is more than sufficient for star recognition purposes. Additionally, the Hipparcos catalog includes a built-in identification number, HIP, which greatly simplifies cross-referencing between the catalog and the database. This identifier is also highly useful for analyzing algorithm performance. Since the simulated images are generated using the same catalog, each simulated image is accompanied by a list of true star IDs. By comparing these IDs with the recognition results, the correctness of the algorithm's identifications can be easily verified. In addition to the HIP identification number, the Hipparcos catalog contains a large amount of additional information for each listed star. The most important parameters are the star's celestial coordinates: Right Ascension (RA) and Declination (DEC), as well as its magnitude. The listings also include numerous other values, such as standard errors and correlation numbers, which are not a must have for the operation and comparison of the algorithms.

When constructing catalogs for the algorithms, only the essential data were selected for the algorithm-specific catalogs: the RA and DEC coordinate values, HIP number, and magnitude. This simplifies catalog inspection and reduces file size. In addition to reducing the number of parameters per star, the total number of stars was also limited. A catalog containing more than 100,000 stars would be far too large, and creating a database from it would result in an excessively large dataset. Moreover, the typical camera used in this study would not be capable of detecting many of the dimmer stars, so the catalog was reduced to better match the algorithms' requirements.

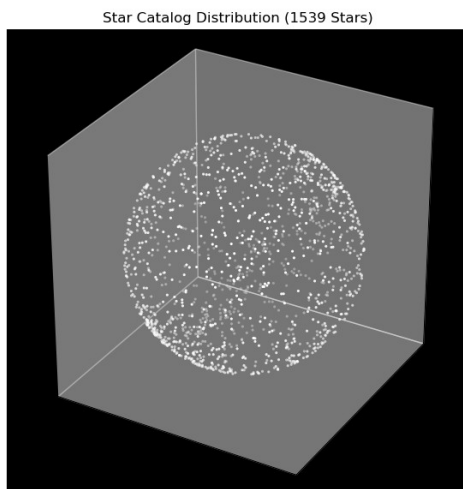
For Liebe's algorithm, the catalog size was kept the same as in the original publication, containing the 1,539 brightest stars. However, for comparison, an additional catalog was created containing the 5,000 brightest stars. The smaller catalog includes stars with a brightness of magnitude 5.049 or brighter. For the Planar Triangle algorithm, a catalog was constructed that includes all stars with a magnitude brighter than or equal to 6.0. This resulted in a catalog of 5,045 stars, making it slightly larger than the second Liebe catalog. The catalog size difference and star distribution is visualized in figure 11. From the figure, it is evident that stars are not evenly distributed on the sphere, but there are no obvious "blind spots" where zero catalog stars are visible if the FOV is large enough. But if the FOV is quite small, there can be problems with smaller catalogs.



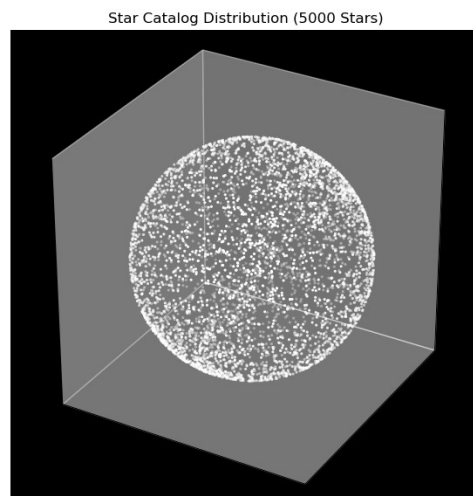
(a) Distribution of 1539 brightest stars.



(b) Distribution of 5000 brightest stars.



(c) 1539 Brightest stars in 3D Space.



(d) 5000 Brightest stars in 3D Space.

**Figure 11:** Examples of different numbers of stars from the Hipparcos catalog shown in 2D (a, b) and 3D (c, d).

All catalogs share the same structure: the first column contains the star’s HIP identification number, followed by the RA and DEC coordinates, and finally the magnitude. In all catalogs, the stars are sorted by magnitude, and the catalogs are in CSV data format. Now that we have a catalog for both algorithms, we can build the feature databases.

For the Liebe’s database, as mentioned in Section 2.5.1, the database can be built using three parameters: the angular distances to two neighboring stars and the angle between these two neighbors. Each of these values is then transformed into a class number. For example, angular distances of  $5.596^\circ$  and  $8.191^\circ$ , and an internal angle of  $164.204^\circ$ , would be represented as 55, 81, and 32, respectively. In addition, uncertainty was introduced into the database, meaning that the extra entries would also be included. For instance, for the mentioned example, additional entries; 54,81,32 and 56,81,32 would also be included in the database because of the uncertainty principle.

In this implementation of the algorithm, the same logic was followed closely according to the principles presented in the original publication. Each star in the catalog was processed individually, and its five closest neighboring stars were identified. Using these neighboring stars, FT combinations were then generated. The quantization values used were the same as in the publication: 0.1 for distance and  $5^\circ$  for the angle. After applying the uncertainty bias, all possible combinations were listed. As a result, each star had approximately 180 different entries in the database. With 1539 stars in the catalog, this resulted in a total database size of 277,020 entries. The same procedure was repeated for the catalog containing the 5000 brightest stars, which resulted in a database size of 900,000 entries. The database is stored in CSV format, where the first three columns correspond to the quantized values QD1, QD2, and QAngle, followed by the star identification numbers: CenterID, Neighbor1ID, and Neighbor2ID.

The construction of the Planar Triangle database is not described as precisely as in the case of the Liebe algorithm; thus, some implementation freedom was applied. As discussed in Section 2.5.2, the Planar Triangle algorithm uses as its feature the area and polar moment of triangles formed by three stars, both calculated using their unit vectors.

The database construction begins similarly to the Liebe algorithm: each star in the catalog is processed individually. The number of neighboring stars is not limited numerically; instead, a maximum angular distance of  $8^\circ$  between stars is defined. This  $8^\circ$  threshold was chosen based on the FOV used in the original publication, which was  $8^\circ \times 8^\circ$ . Even though the simulated camera used in this thesis may have a larger FOV, this does not prevent correct identification. If the maximum distance were significantly increased, geometric distortions caused by the projection could complicate the recognition process. Conversely, this limit does not restrict images with smaller FOVs, since no lower boundary is imposed. For each valid triangle combination, the area and polar moment are computed according to Equations 4 and 9. Each database entry is then added in the form  $[A, J, HIP_1, HIP_2, HIP_3]$ , where  $A$  is the triangle area,  $J$  is the polar moment, and  $HIP_i$  is the identification number of the three stars. The resulting database size with this method is 498,226 entries, which is considerably larger than the Liebe database. This is especially noteworthy

considering that, unlike the Liebe implementation, this database does not include uncertainty-based duplicate entries, meaning all entries are unique. But still, it is less than in the original work, where the number of planar triangle entries in the database was 662,779. Now that the catalogs and databases are ready, we can move on to examining how algorithms were implemented.

### 3.2.2 Liebe Algorithm Implementation

The operation of the Liebe algorithm follows a very typical classical star identification structure, where a feature triangle is first identified from the image and then matched with a corresponding entry in the database, through which the star coordinates can be determined. The implementation in this thesis aims to follow this original concept as closely as possible. However, due to certain ambiguities in the original description, several independent design choices and decisions have also been incorporated into the algorithm.

The algorithm first receives the list of detected stars as its input, meaning that the typical pre-processing phase (such as centroid detection or thresholding) is completely bypassed. This simplification was made because the pre-processing does not affect the comparison between the two algorithms, and the focus of this thesis is solely on the algorithmic recognition performance itself. The input list provided to the algorithm includes each star's  $x$  and  $y$  coordinates in the simulated image coordinate frame, the star's approximate magnitude, and its HIP identification number from the Hipparcos catalog (used only for verification). In a real-world scenario, the HIP ID would, of course, be unknown, but it is included here to simplify result validation. The algorithm also assumes known camera specifications, such as FOV and image resolution, which are used to estimate the image's angular scale.

Once these initial parameters are known, the algorithm begins forming feature triangles in the same manner as during the database construction phase. However, in the case of the image stars, it is possible to limit how many neighboring stars are considered for each central star. In the current implementation, this is limited to 14 nearest neighbors, which is more than in database construction, where five closest neighbors are used. Thus, for each star, the 14 closest neighboring stars are selected, and all possible feature triangle combinations are formed using them. This process is repeated for every star in the image. The triangle features are then calculated and quantized as previously described.

When all features have been generated, the algorithm proceeds to compare them individually against the database. Every database match found is added to a list associated with the corresponding image triangle feature. Due to the relatively coarse quantization, at this stage, each triangle feature typically has multiple matches; therefore, attitude estimation cannot yet be directly performed.

More detailed results of the algorithms are examined in Chapter 5; however, due to the importance of filtering, one example run of the algorithm is reviewed here. The algorithm received a simulated image containing 39 visible stars. Feature triangles

were formed in the usual manner, and the matching phase was started. In total, over 4000 matches were found, of which only 10 were correct. This, of course, means it is practically impossible to determine which of the matches are actually correct. The original publication does not mention any methods for solving this problem or how to proceed when multiple matches are found. Therefore, the filtering methods presented here are developed and adapted specifically for this implementation.

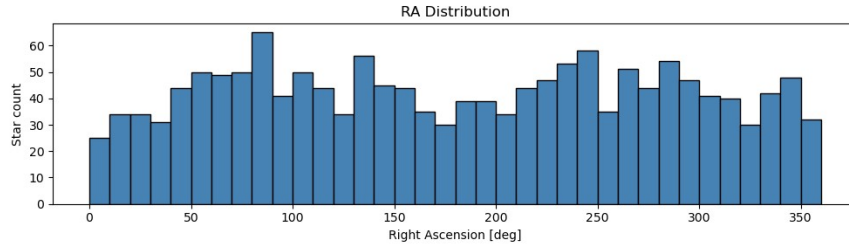
The first filtering method applies to the image scale. For each feature triangle formed from the image, the distances between all three stars can be calculated. This means that instead of using only two sides, a third side is also considered. The same can be done for each database match, allowing for the comparison of all three sides instead of just two. These side lengths can also be quantized, but in this implementation, distances in degrees and a fixed tolerance for side length differences are used instead. This filtering method has proven to be relatively effective, and the inclusion of the third side as a feature yields good results. For example, in the earlier case, the total of 4000 matches was reduced to 334, which is more than a tenfold reduction. However, even 300+ matches were still too many for reliable attitude determination.

The next idea was to analyze the matched stars' RA and DEC coordinates and attempt to group them to see if any clustering could be observed. However, it quickly became apparent that when the match coordinates were divided into moving bins, the results were incorrect; some false regions of the sky tended to dominate. This led to a closer examination of the coordinate distribution of stars in the catalog. As can be seen from Figure 12, although the stars in the smaller catalog are fairly evenly distributed over the celestial sphere, they are not numerically uniform. Next, an attempt was made to normalize the catalog coordinates and use them for comparison, but the results remained poor, even though a correct match was occasionally found.

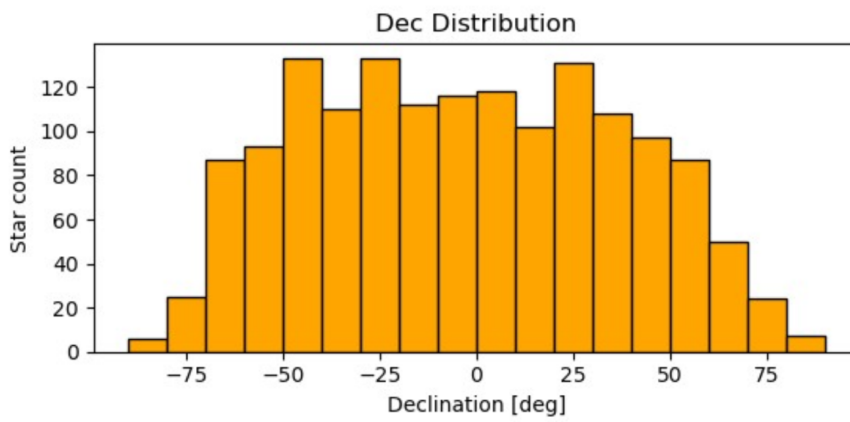
With an appropriate approach and a well-formulated mathematical model, a coordinate-based clustering method has the potential to be effective for both large and small data sets. However, during the exploration of various filtering techniques, no relevant research papers directly addressing this specific topic were identified. As a result, alternative filtering approaches were examined, and the focus was eventually placed on using the magnitude as the primary criterion.

Magnitude has also rarely been used as a filtering parameter; at least, it has not been frequently mentioned in the literature. However, since the pre-processing phase can determine magnitudes with high accuracy and since this information is always available in both catalogs and optionally in databases, it has proven to be a very convenient filtering method.

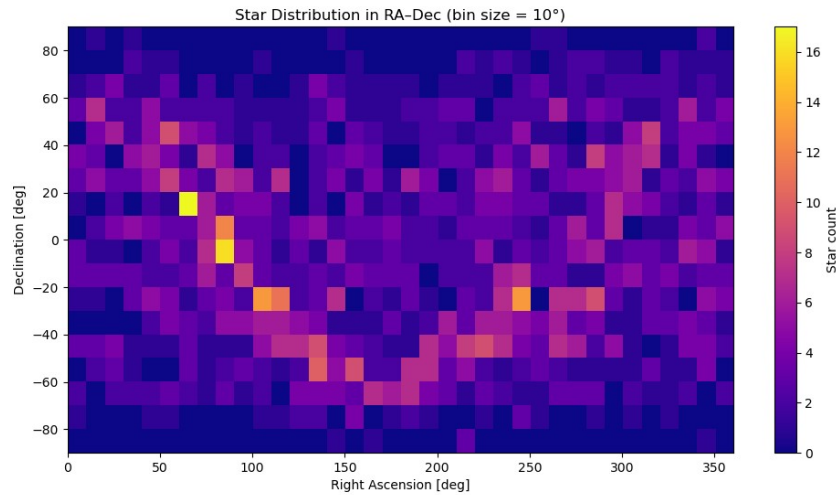
There are several possible ways to use magnitude. One could, for instance, directly compare the absolute brightness values between the image and the database matches. In this implementation, however, the comparison is based on the relative magnitude order within each feature triangle. That is, for a chosen anchor star, its magnitude is compared to that of the other two stars in the triangle, and the stars are sorted by brightness. For example, if one of the neighboring stars is the brightest and the anchor star is the dimmest, the brightness order can be represented as [2, 3, 1], from brightest to faintest. This same brightness ordering is computed for all database triangles as



(a) Right Ascension (RA) Distribution.



(b) Declination distribution.



(c) RA-DEC distribution.

**Figure 12:** Distribution of the 1,539 brightest stars from the Hipparcos catalog used in this work, shown with a 10° bin size.

well, and if the order does not match, the corresponding match is discarded. With this filtering technique, the number of matches was reduced from 334 to 12 triangles. In other words, only two incorrect triangles remained, meaning that over 75% of the remaining stars were already correctly identified.

Now that filtering has increased the number of correct matches relative to incorrect ones, a final verification can be performed, and a small set of stars selected for attitude estimation. The final verification and fitting are carried out using Wahba's problem. The idea is to form unit vectors for the image stars and the corresponding unit vectors from the database, and then find the rotation that aligns the two vector sets. For each remaining triangle, different combinations are formed, and all combinations are tested to determine which feature triangles fit best. By comparing the relationships between different triangle features, the last incorrect matches can be filtered out. From the resulting rotation matrix, one can then derive the image pointing and perform attitude estimation. Next, we turn to the implementation of the Planar Triangle algorithm.

### **3.2.3 Planar Triangle Algorithm Implementation**

The Planar Triangle algorithm was also implemented to follow the original publication as closely as possible. However, due to certain ambiguities in the paper, some custom design choices were necessary. In terms of functionality, the Planar Triangle algorithm operates quite similarly to the Liebe algorithm. First, the image is processed to extract the (x, y) coordinates of the detected stars, which are then used to perform feature triangle extraction. As mentioned previously, the features of the Planar Triangle are its area and polar moment. After this, a filtering step may be applied, or the process can proceed directly to the pivot function, which reduces the number of false matches. Finally, as in the previous algorithm, Wahba's problem is used to determine the rotation matrix and thereby achieve attitude estimation.

In this implementation, the pre-processing stage was completely omitted, following the same approach as in the Liebe algorithm. The star coordinates are initially obtained directly from the camera simulation as a list, which also includes the estimated magnitude and, for verification purposes, the HIP ID.

The algorithm begins by constructing all possible triangles from the stars detected in the image. However, the maximum side length is limited to  $8^\circ$ , identical to the constraint used during database construction. This ensures that the triangles formed in the image are theoretically present in the database. Since the simulated camera model uses a stereographic projection, the triangle areas are not expected to remain highly accurate near the edges of the image due to the wide field of view. Therefore, to maintain the feature area and polar moment as reliable and comparable quantities, it is preferable to prioritize triangles located near the image center. The detected triangles are thus ranked according to their quality so that poor triangles can be ignored. Each triangle is scored based on the following criteria:

1. Centroid proximity to the image center
2. Side length similarity to 3°
3. Avoidance of overly narrow (skinny) triangles
4. Pivot star availability

Among these criteria, the centroid proximity is given the highest weight, with a coefficient of 5. The side length similarity uses a weight of 2, and the shape criterion a weight of 3. If there are no pivot stars available, the triangle is completely discarded. This scoring process is repeated for all triangles in the image, producing an approximate quality ranking of the triangles. More precisely, the score is obtained using the Equation 10:

$$score = (5d_{center} + 2|l_{mean} - l_{target}| - 3(\frac{A}{l_{mean}^2})). \quad (10)$$

Here  $d_{center}$  is the distance to the center of the image from the triangle centroid,  $l_{mean}$  is the mean triangle side length,  $l_{target}$  is the preferred side length for the triangle, and  $A$  is the area of the triangle.

Using the computed score, the triangles in the image can now be ranked by quality, and the algorithm can proceed to the next stage: calculating the triangle features: the area and polar moment, using the unit vectors of the stars. Once these triangle features have been computed, the matching phase can begin.

The matching phase can either be performed for all triangles or, to speed up the simulation, only a limited number of top-ranked triangles can be selected based on the previously computed scores. In this implementation, only the five best triangles are passed to the matching phase. During this stage, the calculated triangle areas and polar moments are compared against the entries in the database within predefined tolerance limits, and all possible matches are collected and linked to the corresponding image triangle.

Due to the differences between the image-derived area and the polar moment, as well as those stored in the database, the tolerance must be kept relatively wide. This results in multiple incorrect database entries being associated with a single image triangle. To eliminate these false matches, a filtering step is again applied, similar to the one used in the Liebe algorithm, as well as the pivoting method described in the original Planar Triangle paper. Initially, the goal was to rely solely on the pivoting approach, but the number of matches per triangle proved too large; therefore, an initial filtering stage was necessary.

The filtering is performed by comparing the side lengths of each image triangle with those of the corresponding database entries. If the side lengths do not match within the tolerance limits, the entry is discarded. This process significantly reduces the number of matches before the pivoting stage.

As described in Section 2.5.2, the pivoting process operates by removing one star from the currently evaluated triangle and replacing it with a new nearby star selected

from the image. In this implementation, the nearest available star is chosen. Once the new star is selected and a new triangle is formed, its triangle features (area and polar moment) are recalculated, and matching entries are again searched for in the database. When the matches for two triangles are available, these two match lists can be compared. Since the two triangles share two common stars, the goal is to find triangles in both lists that also share these two stars. After this comparison, the number of matches is evaluated. If more than one valid match remains, the pivoting process is repeated, resulting in three triangles, each sharing two identical stars, while the third one is unique.

This pivoting is repeated three times in this implementation to reduce the number of matches. Finally, the Wahba problem is solved to compute the rotation matrix and determine the attitude estimation. It is worth noting that this final step does not strictly require the matches to be reduced to a single triangle; the Wahba solution can still be performed if a few potential matches remain after the pivoting stage.

In this implementation, since the initial number of matches before filtering is extremely large, the pivoting process does not always significantly reduce the total number of candidates. When the number of pivoting iterations is limited to three, each newly formed pivot triangle typically receives around 1,000 to 10,000 potential matches. In many of these cases, several stars are found to be common with those from previous pivoting iterations.

For example, in one simulation case, the initially selected image triangle yielded over 3,000 matches, which were reduced to only 61 after applying the side-length filtering. Pivoting was then used: after the first pivot, the number of matches dropped to 32; after the second iteration, it decreased further to 12; and after the third pivot, no additional reduction was achieved. However, in this situation, the Wahba fitting process was still able to successfully determine the correct attitude solution, even though 11 false triangles existed.

### **3.3 Simulation Structure and Data Gathering**

Now that the implementation of the individual components of the simulation has been examined in detail, we can move on to reviewing the overall simulation, the comparison process, and the data collection. The simulation is executed in a separate main file, `main.py`, which calls the different simulation modules and gathers the resulting data. An overview of the simulation structure in the flowchart is shown in Figure 13, which illustrates the order of different functions, such as the filtering functions and image generation.

The simulation's algorithm and camera settings can primarily be adjusted only within their corresponding files, while `main.py` controls the number of simulation runs and the location where the collected data is stored. First, the simulation calls the camera simulation module, which generates a star image based on the defined settings, as explained in 3.1.1. The simulated image can be displayed if desired; however, when simulating large quantities of scenarios, plotting is disabled. In such cases, the

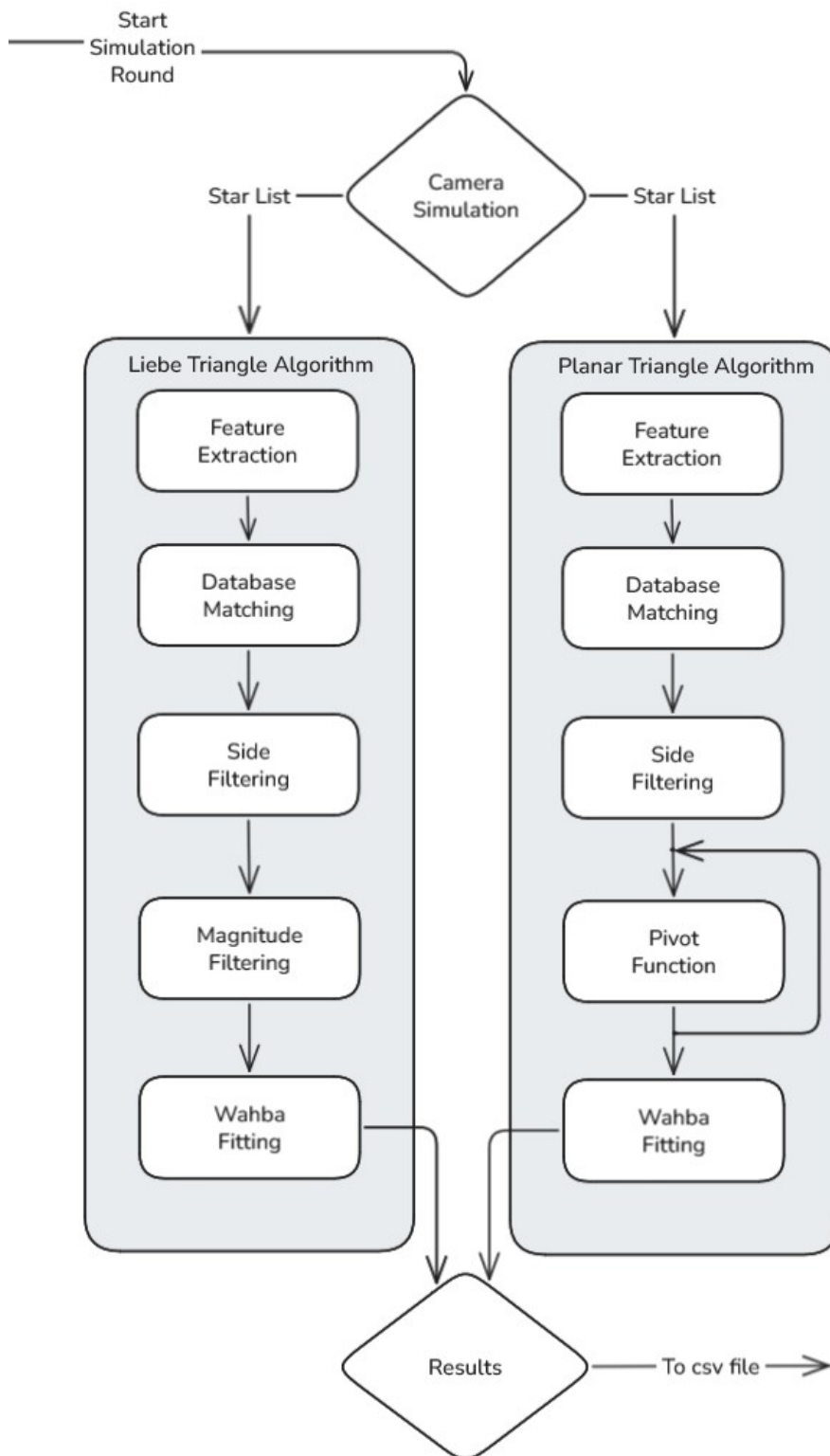
camera module only returns a list of simulated stars and their corresponding in-frame coordinates. This list is then passed unchanged to both algorithms, ensuring that the initial input data for each algorithm is identical. After this step, the process moves to the algorithm side, where star recognition is performed and data collection takes place. The algorithms return a solution that is a list of matched stars between the image and the database. Using this list, it is possible to verify whether the matched database stars are correct, as the true star identities are also available from the camera simulation output.

In addition to the verified solution, various types of data are collected to analyze the performance of different stages within the algorithms. The collected data for both algorithms are as follows. First, we have the initial amount of matches: this records the number of star matches obtained during the first matching round, which indicates how many database feature triangles were initially paired with image feature triangles before any filtering or selection. This metric helps evaluate the effectiveness of the filtering stages and reveals how broad the initial matching tolerances are.

Next, both algorithms record the number of matched feature triangles after the filtering stages. Both algorithms collect the number of matches after the first scale based filtering step. The Liebe's algorithm additionally logs the number of matches remaining after the magnitude filtering, while the Planar Triangle algorithm records the number of matches left after pivoting. These filtering related metrics are essential for assessing how well each filtering stage performs.

Afterward, both algorithms collect the number of visible stars and the runtime in seconds. The number of visible stars is affected by the database's lowest designed magnitude; therefore, the number of visible stars is expected to be higher for the planar algorithm. These values make it possible to analyze how the number of stars affects algorithm performance and filtering efficiency. Runtime is also a crucial indicator of algorithmic efficiency under different test conditions.

Finally, for debugging purposes, the system records whether the algorithm completed the simulation successfully and logs any error messages that may occur, allowing for later inspection of potential code or logic failures. All the aforementioned variables are collected from every simulation run and stored in CSV files for result analysis. Data from different simulation configurations are saved in separate CSV files to ensure clarity and prevent misinterpretation. In the next section, we proceed to examine the obtained simulation results in Chapter 4.



**Figure 13:** Flowchart of the simulation process used in this study.

## 4 Results and Discussion

This chapter examines the simulation results obtained from eight different simulation setups, as summarized previously in Table 3. In each configuration, both implemented algorithms were executed 200 times. The results are analyzed in terms of efficiency, robustness, and computational performance across the different setups. First, the chapter provides a general overview of the algorithms' solution capabilities and performance under nominal conditions. Then, in Section 4.2, the functionality of the algorithms under varying conditions is examined; for example, how FOV or noise affects their accuracy and speed.

Section 4.3 focuses on failure cases, identifying the reasons why the algorithms fail to perform correctly under certain scenarios. Section 4.4 presents a comparative analysis between the algorithms, highlighting which one performs better and under what circumstances each demonstrates its strengths. Finally, Section 4.5 provides a critical reflection on the quality of the simulations and implementations, comparing the results to the performance evaluations presented in the original publications.

### 4.1 Overall accuracy and Success Rates

The general success rate refers to the algorithm's ability to determine the positions of stars in the celestial coordinate system from a given image. If the algorithm is able to correctly solve both the position and identification numbers of the stars in the image, the simulation is considered successful. If the stars are incorrectly identified, or if the simulation fails to complete, the result is marked as unsuccessful. Using these simulation outcomes, a success rate percentage can be calculated for each algorithm under each configuration. These results are presented briefly in Table 4, which also includes the mean values of other collected simulation parameters, such as runtime and the number of detected stars.

The first column of the table presents the simulation configuration, the details of which were previously described in Section 3.1.2. The next column shows the algorithm identifier, which consists of the letters L or P to indicate whether the result corresponds to Liebe's algorithm or the Planar Triangle algorithm, respectively. Following that, the Success Rate is presented as a percentage. The subsequent column shows the Mean Runtime, which was calculated using only the successful simulation runs; cases in which the algorithm failed to solve the star identifications were excluded from the average. This same principle applies to the mean values of the parameters  $N_{stars}$ ,  $N_m$ ,  $N_{f1}$ , and  $N_{f2}$ , which represent, respectively, the total number of stars in the image, the number of matched stars after the first matching round, the number of stars remaining after the first filtering stage, and the number of stars remaining after the second filtering (or pivoting) stage.

Let us first examine the initial six rows of the table, corresponding to the three base simulation configurations, which represent the baseline conditions for the algorithms. In the Altered Liebe's algorithm, the first filtering stage has been removed, which is also indicated in the table as N/A. In the Altered Planar Triangle algorithm, instead of

**Table 4:** Overall success rates and average performance metrics for each simulation and algorithm used in this study. The best values are highlighted with a gray background for clarity.

Simulation Name	Algorithm	Suc. [%]	$t_\mu$ [s]	$N_{stars}$	$N_m$	$N_{f1}$	$N_{f2}$
Normal	L	96.0	8.689	55.90	988.82	91.72	6.54
Normal	P	94.5	9.303	57.17	6176.62	52.14	16.56
Alt. Liebe	L	99.0	8.596	57.94	1121.51	N/A	26.15
Alt. Liebe	P	95.0	8.823	58.23	5415.55	51.65	18.71
Alt. Planar	L	96.0	7.360	55.05	934.04	85.14	6.37
Alt. Planar	P	77.0	7.406	57.14	5725.73	52.57	18.04
Small FOV	L	98.5	8.504	43.78	2452.41	177.83	8.33
Small FOV	P	64.5	25.889	44.12	7492.08	49.56	31.61
S. FOV + few false	L	99.5	9.028	45.42	4207.75	157.64	7.81
S. FOV + few false	P	52.0	23.758	45.52	6375.69	49.70	30.59
S. FOV + many false	L	79.0	16.255	43.77	10751.00	69.39	4.16
S. FOV + many false	P	27.0	31.490	46.80	6997.15	48.90	29.63
Noise (low)	L	95.0	5.513	60.58	627.33	59.45	5.24
Noise (low)	P	95.0	8.216	61.95	6261.44	51.74	16.01
Noise (high)	L	40.0	8.947	37.94	974.44	85.69	1.83
Noise (high)	P	1.5	26.887	60.24	5102.39	50.33	16.53

using the five best triangles, the algorithm attempts to solve the image using only the single best triangle. The Normal configuration, in turn, represents the standard setup without any modifications.

By looking at the success rate, it can be observed that the Liebe’ algorithm’s solution rate does not change significantly when the filtering is modified, remaining consistently around 96–99%. The Planar algorithm, however, shows a much stronger sensitivity to this change in the altered planar triangle configuration. When only one triangle is used instead of five, the success rate drops from around 95% to 77%.

When examining the runtimes, we see that in the altered Planar triangle case, the recognition occurs roughly one second faster. However, this effect is also visible for Liebe’s algorithm under the same configuration, which suggests that the 200 test images in this batch may have been slightly easier to solve, rather than the reduced number of triangles providing a real computational advantage. Overall, the runtimes of the algorithms do not vary much, staying within about one second of each other. Neither implementation appears particularly fast, but this is largely due to the extensive use of slow CSV-based data structures. For instance, the catalog and database are iterated over several times, and with lengths reaching up to 100,000 entries, this inevitably introduces some computational slowness.

Moving on to the matching phase, the Liebe algorithm yields, on average, about 1,000 matches after the first feature triangle matching round, which is significantly fewer than the 5,400–6,200 matches produced by the Planar Triangle algorithm matching phase. This likely results from differences in the catalog and database

structures used by the algorithms. As mentioned earlier, a larger number of database entries increases the probability of encountering a higher number of false matches. Another factor may be the looseness of the matching tolerances. While looser thresholds may allow for more false matches, they can also help in the presence of positional noise, provided that filtering is effective enough. And in this case, the filtering indeed appears to be effective.

In the Normal configuration, the first filtering stage (scale filtering) removes about 90% of feature-triangle matches for Liebe's algorithm, showing that including the third side length in the comparison significantly narrows down the results. For the Planar Triangle algorithm, the scale filtering is even more selective, eliminating about 99% of false results and leaving only around 52 candidate solutions, which is still quite a large number considering that this corresponds to a single triangle out of the five selected. However, the combination of side length, area, and polar moment seems to be a rather good combination to use in feature triangles. In the second stage, the Liebe algorithm's magnitude filtering again removes roughly 90% of the remaining matches, leaving, on average, 6.54 feature triangles. This is noticeably better than the Planar Triangle algorithm's pivoting, which only removes about two-thirds of the false matches, leaving around 16 candidate triangles.

In the altered Planar Triangle algorithm, the results remain largely similar, as the filtering logic has not been modified. However, the Altered Liebe algorithm produces particularly interesting results: since the scale filtering was completely removed (hence  $N_{f1} = -1$ ), only the magnitude filtering remains active. This single filtering stage still managed to remove the majority of incorrect matches, and the success rate remained high, even though the number of remaining triangles was roughly four times higher compared to the normal configuration.

Let's take a brief look at the success rates under stress conditions. For the Liebe algorithm, it can be observed that the success rate remains exceptionally high (around 99%) when the FOV is reduced, even in cases where a small amount of positional noise or a few false stars are added to simulations. However, even for the Liebe algorithm, the success rate drops when a larger number of false stars is introduced, though it still remains relatively strong at about 70%. The weakest performance occurs when a high level of positional noise is added, at which point the success rate falls to around 40%.

For the Planar Triangle algorithm, good success rates are maintained only under conditions of low noise, where the performance is comparable to that of Liebe's algorithm. However, the success rate decreases significantly as the FOV becomes smaller and continues to drop further when false stars are added to the small FOV scenario. In the high-noise scenario, the success rate drops to only 1.5%, meaning that in practice, the algorithm can no longer be considered to provide successful solutions.

Next, in Section 4.2, these stress test conditions and their simulation results are examined in more detail.

## 4.2 Performance under Different Conditions

This section takes a closer look at the effects of different variables on the operation of algorithms. First, the effect of the reduction in the FOV is examined in Section 4.2.1, followed by an analysis of the impact of false stars in Section 4.2.2. Then, Section 4.2.3 focuses on the results related to positional noise and its influence on algorithm performance. After that, Section 4.2.4 examines the effect of the number of visible stars.

### 4.2.1 Effect of FOV

Let us take a closer look at the effect of the FOV on the performance of the algorithms. Here, we focus specifically on the results of the Normal and Small FOV simulation configurations, excluding cases that include false stars. This allows us to isolate and analytically assess the impact of FOV reduction. As mentioned earlier, and as shown in Table 4, Liebe' algorithm is barely affected by the smaller FOV. In fact, the success rate even increases by about 2.5 percentage points; though this is most likely due to random variation. The test set of 200 simulated images may have included slightly easier cases to solve. For the Planar Triangle algorithm, however, the change is significant: the success rate drops from 94.5% to 64.5%. A decrease of 30 percentage points clearly indicates a notable degradation in the algorithm's robustness as the FOV decreases.

The other values for the Liebe algorithm remain largely similar to those in the normal configuration. The number of visible stars is slightly smaller than in the larger FOV case, which is expected since a smaller image area naturally contains fewer stars. However, the number of matches after the first round has increased, and consequently, the filtering stages also produce slightly higher intermediate counts. Still, after the magnitude filtering, only about 8.33 matches remain on average, which is close to the original mean of 6.54 matches. The increase in initial matches is likely explained by the size of the feature triangles: in larger-FOV images, triangles tend to form between stars that are farther apart, while in smaller FOV images, the triangles are smaller and such configurations are more densely represented in the database.

For the Planar Triangle algorithm, the remaining metrics also deviate slightly from the normal case, especially the runtime, which is more than twice as long. This is most likely because the algorithm spends excessive time attempting to find a valid solution before giving up. The number of feature triangle matches during the matching phase is roughly higher than in the normal case. However, the first filtering round produces about the same number of candidates as before, while the pivoting stage performs notably worse: after pivoting, 32 matched triangles remain, which is roughly double the number in the normal condition. This is likely too many candidates for the Wahba solver to handle effectively, which explains the resulting drop in the overall success rate.

The Liebe algorithm performs excellently and is capable of forming a sufficient number of feature triangles even in cases with a smaller FOV. The database also

appears to contain enough of these smaller triangle features to enable reliable matching and successful solutions. The Planar Triangle algorithm, however, does not perform nearly as well, so let us take a closer look at the reasons behind its poor success rate. The simulation results are divided into successful and failed runs, as seen in Table 5.

**Table 5:** Mean performance metrics for the Planar Triangle method with reduced FOV, separated between successful and failed runs, as obtained in this study.

Planar Algorithm	$t_{\mu}$ [s]	$N_{stars}$	$N_m$	$N_{f1}$	$N_{f2}$
Successful Run	17.795	43.359	6518.771	48.366	29.878
Failed Run	40.393	45.458	9298.375	51.764	34.889

The average runtime for failed simulations was over 40 seconds, while for successful runs, it was only 17 seconds. The number of matches after the first matching round was also significantly higher in the failed cases, with about 9,200 matches compared to 6,500 for successful runs, representing roughly a 50% increase. This excessive number of matches is also reflected in the final number of matches after the pivoting stage: successful simulations ended with an average of 29 triangles, whereas failed ones still had around 35 triangles remaining. So, it is clear that in failed simulations, the performance metrics are worse than those in successful simulations.

#### 4.2.2 Effect of False Stars

False stars were added in two separate simulation stages: first, a small number of five false stars per image, and then a second simulation with a larger number of twenty false stars. In these false star simulations, the FOV was also reduced, so the results should be analyzed primarily in relation to the Small FOV simulation discussed in the previous section, rather than the normal baseline conditions.

From Table 4 at the beginning of Chapter 4, we can again find the results for these two false-star simulations. For Liebe’s algorithm, it is observed that the success rate in the smaller false star test is 99.5%, which is, in fact, the highest success rate among all simulations, even surpassing the normal configuration. However, the performance of Liebe’s algorithm decreases slightly when 20 false stars are added, which causes the success rate to drop to 79%, representing a reduction of about 20% points.

For the Planar Triangle algorithm, false stars pose a more significant challenge. With five false stars, the success rate is only 52%, and when the number of false stars is increased to twenty, the algorithm’s performance deteriorates further by about 25 percentage points, reaching a 27% success rate.

When it comes to the number of matches and filtering, compared to the small FOV case, the number of matches in the first round increases for the Liebe algorithm when false stars are added: instead of around 2,500 matches, the number rises to 4,200 and 10,700. Both values are several times higher than those in the small FOV configuration and far greater than in the normal baseline case.

However, the filtering appears to function flawlessly. The scale filtering stage reduces the number of matches to 157, which corresponds well to the small FOV situation. When more false stars are added, the scale filter performs even better, reducing the number of matches to 69. The magnitude filtering also successfully narrows down the matches to approximately eight and four final matches, respectively. These results indicate that Liebe's algorithm's filtering steps are highly effective at removing matches formed from false stars, leaving primarily the correct matches through to the final stage and leading to a high success rate.

For the Planar Triangle algorithm, the number of matches in the first round is slightly smaller than in the small FOV case, at around 6,300–7,000. The scale filtering yields nearly identical averages compared to the small FOV scenario, as does the pivoting stage, which ultimately leaves about 30 filtered matches. However, the success rate remains significantly lower than in the small FOV case. This likely stems from the algorithm's operational principle, where only the five best triangles are processed instead of all possible triangles. When false stars are introduced, it becomes increasingly probable that some of these top five triangles include false star vertices, leading to incorrect solutions.

Run times for Liebe's algorithm were aligned with the success rate. When only a few false stars were present, the runtime was approximately the same with a small FOV configuration; however, when more were added, the runtime almost doubled to over 16 seconds. For the planar triangle algorithm, the average runtime was aligned with the average of the small FOV simulation.

### **4.2.3 Effect of Noise**

The final two simulation configurations involve positional noise. Two different setups were implemented for this case: one with a low amount of noise, representing realistic operating conditions of star sensors, and another with a high amount of noise, which might correspond to a faulty or degraded sensor. In these simulations, the 30° FOV was used, allowing the results to be compared directly to the normal baseline configuration.

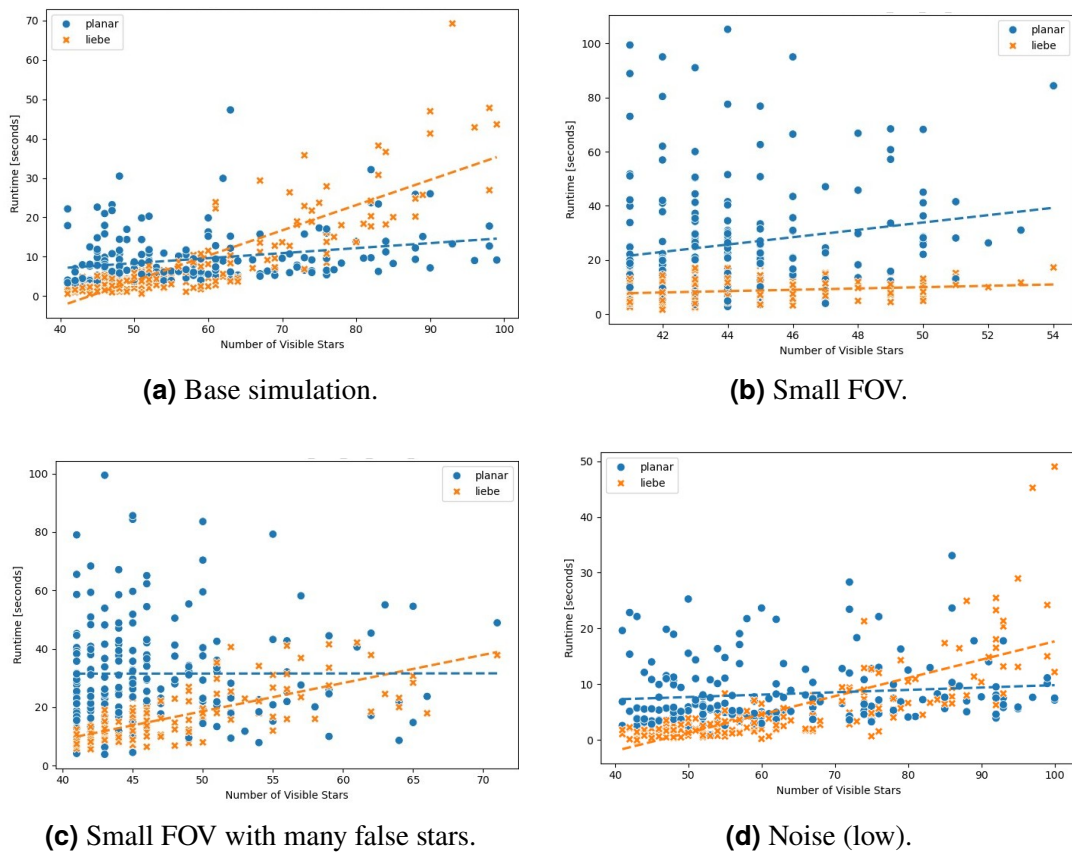
Both algorithms performed very well in the low-noise simulation, each achieving an identical 95% success rate. All other metrics also appear nominal compared to the normal condition. Filtering works reliably, and the runtime remains consistent. The only notable difference is that, for Liebe's algorithm, the number of matches after filtering is slightly lower than in the normal case.

In the high-noise scenario, however, both algorithms' performance drops significantly. Liebe's algorithm reaches its lowest success rate of 40%, while the Planar Triangle algorithm barely solves any cases at all, with only a 1.5% success rate (succeeding in just 3 out of 200 simulations). Despite this, the mean values of the matching and filtering stages for both algorithms remain close to those of the normal condition. This suggests that, due to positional noise, the stars' locations are sufficiently distorted such that the algorithms begin to match triangles with incorrect database entries, particularly in the case of the Planar algorithm. For the Liebe algorithm, however, the

bin size used in feature generation appears to be large enough to tolerate substantial positional noise, allowing it to still correctly identify about 40% of the images, even under extreme conditions.

#### 4.2.4 Effect of Amount of Visible Stars

No noticeable effect of the number of visible stars was observed on the success rates. For example, in Section 4.2.1 in Table 5, the differences between successful and failed simulations of the planar triangle algorithm were examined, and the number of visible stars did not show a significant influence. This is completely logical since the structural functioning of the algorithms does not strongly depend on the number of stars, as long as the minimum required amount is met. However, the number of visible stars does affect the algorithm runtime, so in this section, the correlation between runtime and the number of visible stars is examined.



**Figure 14:** Run times (in seconds) as a function of the number of visible stars for the Planar Triangle algorithm (blue dots) and Liebe (orange dots) methods analyzed in this study.

From the data of four different simulation configurations, plots were generated and are presented in Figure 14. The configurations examined are the base simulation,

small FOV, small FOV with false stars, and low noise. In all plots, data from Liebe's algorithm is shown in orange, and the data from the planar triangle algorithm is shown in blue.

In the base simulation, shown in Figure 14a, the number of stars varies between 40 and 100; however, in most simulations, the number of visible stars remains below 70. For the Liebe algorithm, the runtime increases as the number of stars grows, not perfectly linearly, but quite clearly when considering the average. With fewer than 55 visible stars, the runtime never exceeds 10 seconds; whereas, after 60–70 stars, this becomes the typical minimum time for the algorithm to complete. The maximum runtime observed was approximately 50 seconds, with one exceptional case reaching 70 seconds.

For the planar triangle algorithm, no clear correlation is visible; perhaps only a few seconds increase at most. The runtime appears very consistent in both the 50–70 and 70–100 visible star ranges. Generally, the runtime is between 7 and 10 seconds, though a considerable number of outliers between 15 and 30 seconds exist, regardless of the number of visible stars.

Figure 14b presents the small FOV simulation. Here, the first observation is that the number of visible stars is lower than in the normal case, with a maximum of 54 visible stars due to the smaller FOV. The curve for Liebe's algorithm appears very stable and similar to the base simulation if only the range 40–54 is considered. Thus, reducing the FOV did not significantly affect the performance of the algorithm.

However, for the planar algorithm, the runtime increases drastically compared to the base simulation, with the curve starting at about 20 seconds and reaching around 40 seconds at the maximum star count. A notable number of simulation runs take over 30 seconds, which was very rare in the base simulation. There are, of course, a few runs where the simulation completes in just a few seconds, but these are very uncommon.

Figure 14c shows the simulation, including the smaller FOV with false stars. Liebe's algorithm shows a similar pattern to the base simulation, where runtime increases with the number of stars; however, in this case, it increases much more sharply. The average starts at about 10 seconds and reaches 30–40 seconds at around 60 stars. Thus, the presence of false stars clearly causes longer runtimes for Liebe's algorithm, especially compared to the small FOV case without false stars, where no significant effect was observed.

For the planar triangle algorithm, there was a large variation regardless of the number of stars, which has already been noted to be typical for this algorithm. In some cases, it took only a few seconds, while in others, the runtime reached up to 100 seconds; however, most runs fell between 20 and 40 seconds. With few visible stars, it was noticeably slower than Liebe's algorithm, but as the number of stars increased and Liebe's performance degraded, the runtimes became roughly similar.

Finally, Figure 14d shows the results of the low positional noise simulation. The plot closely resembles the base case. The runtime of the Liebe algorithm is nearly

identical to that of the base simulation for cases with fewer than 60 visible stars; after which, between 60 and 100 stars, the runtime increases. The spread of runtimes is almost the same as in the base simulations.

This also applies to the planar triangle algorithm, whose curve and average runtime resemble those of the base simulation. With fewer stars, the solution time is about 10 seconds, and as the number of stars increases, the curve remains at roughly the same runtime level. The spread is smaller than in the small FOV cases, and the maximum runtime observed is only about 50 seconds.

In general, Liebe's algorithm is much more affected by the number of visible stars, and this directly impacts its runtime. The algorithm is extremely fast when the number of stars is around 40–50, often taking only a few seconds. However, it slows down significantly when the number of stars exceeds 70. Its speed would likely also decrease if the number of stars were reduced further, although that might lead to a drop in the success rate.

The Planar Triangle algorithm is generally slower than Liebe's algorithm. No clear correlation was found between the number of visible stars and their runtime, which varied rather evenly regardless of the star count. It may be a few seconds slower when more stars are present. This is likely due to the algorithm's structure, where only the five best triangles are selected for evaluation. The formation of triangles does not seem to depend much on the number of stars, since database searches are not required. Overall, the slower performance compared to Liebe's algorithm is probably due to the number of pivoting rounds and the many list comparisons. The Planar Triangle algorithm also had a much larger database than Liebe's, which may further slow it down, especially since no optimized data structures were used.

### **4.3 Failure Analyses**

Liebe's algorithm performs extremely well under all normal and relatively typical conditions. Its success rate consistently exceeded 95%, except in very challenging and unrealistic stress test configurations, where a significant number of false stars were added or where the positional noise was extremely high. Even in those cases, the success rate remained decent. Few failed solution attempts were observed. In normal conditions, these failed runs always ended with a "division by zero" Python error. Upon closer inspection, this was caused by all matches being filtered out during the filtering stage, which led to division by zero in a helper function. In the normal baseline simulation, however, Liebe's algorithm never guessed the wrong pointing. Every failed run was due to the algorithm terminating early because no matches remained. The same error message appeared in all other simulation configurations as well, and there were no cases where the algorithm produced an incorrect guess. Thus, the filtering was extremely effective, perhaps even too strict in some cases.

The Planar Triangle algorithm performed excellently under normal conditions and when a small amount of positional noise was present. However, its success rate dropped dramatically when the FOV was reduced or when false stars were added to

the image. In normal conditions, the algorithm always completes its execution, unlike Liebe' algorithm. This means that in failed runs, it did not encounter an error but instead guessed the stars incorrectly. The data does not reveal whether the guess was still relatively close to correct (for example, if two out of three stars were identified correctly), since even small errors were classified as failures. By examining the data from failed runs, no clear reason for the algorithm's failures could be identified; for example, the number of matches after filtering varied evenly from a few to several dozen, meaning no direct correlation could be established from the available data. It is likely that the cause lies in the structural design choices of the algorithm: the limited number of generated triangles, a small number of pivot iterations, and possibly overly broad tolerance limits may all contribute to these issues.

It is quite possible that examining only five triangles is not sufficient to find the correct solution, and a larger number of triangles may be needed, especially when false stars are present in the image. As mentioned previously, the simulations were performed for two versions of the planar algorithm: first, the normal configuration, where five triangles were used for matching, and then a modified planar simulation, where only one triangle was used. As shown in the Table 4, the success rate of the planar algorithm increased from 77% to 94.5%, meaning it is entirely possible that increasing the number of triangles would further improve the success rate.

When it comes to false stars, it is very likely that when only five triangles are selected, each of them contains at least one false star, which makes solving them theoretically impossible. With a greater number of triangles, the probability of having a triangle without false stars increases. The number of pivoting iterations should also be increased, as this would reduce the likelihood of false stars being present in all pivot triangles.

The number of triangles examined and the number of pivot iterations were limited to their current values due to the excessively long runtime. As seen from the data in the previous section, the Planar Triangle algorithm is not particularly fast even now; therefore, doubling or tripling the number of triangles and adding more pivot iterations would greatly slow down the algorithm, which would be undesirable. The database size, algorithm structure, and programming design would likely need to be reconsidered if the number of processed triangles and pivot iterations were increased while keeping the runtime acceptable.

The effect of the tolerance parameter is difficult to assess, but it may be a possible cause of issues. If the tolerance is too small, the correct database entries may fail to match the image triangle when the triangle is slightly distorted by camera optics. Conversely, if the tolerance is too large, too many matches may be formed because multiple database triangles are incorrectly matched to the image triangle. In this implementation, a very large number of triangles are matched, often around 6000. Even after filtering and pivoting, the number of matched triangles remains close to twenty, which may cause the fitting function to accidentally find another closely matching but incorrect solution. This might also be caused by an overly large database and not by too loose matching tolerance.

Of course, it is also possible that the area and polar moment simply do not work as well for star recognition, especially when false stars or noise are present in the image. The area and polar moment are not sufficiently unique features, which lead to ambiguity and false matches between the image triangles and the database entries.

#### 4.4 Algorithm Comparison

Now that the performance of both algorithms has been examined from several perspectives and across multiple simulation scenarios, a general comparison can be made to determine which algorithm performs better. Before proceeding, we briefly summarize the overall performance of each algorithm.

Liebe's algorithm performed very well. Under normal conditions, the success rate was around 96%, and the results were highly accurate, even though the filtering of the third triangle side was omitted. The algorithm also performed excellently in various challenging scenarios, such as reduced FOV cases and when false stars were present in the image. The success rate of the algorithm dropped below 95% only in situations where the number of false stars was excessive (20 stars) or when the level of positional noise was extremely high.

The planar triangle algorithm also performed very well under normal conditions, achieving a success rate similar to that of Liebe's algorithm at around 95%. However, its performance deteriorated when subjected to more challenging conditions. The reduced FOV case lowered the recognition rate to 64.5%, and the success rate continued to drop as false stars were added. The algorithm still performed reasonably well under small positional noise, but when the noise level increased, its ability to recognize stars dropped almost to zero.

Among the two algorithms, Liebe's method yields the most favorable results. In every simulation, Liebe's algorithm achieved a higher success rate than the planar triangle algorithm, except in the low-noise simulation, where both reached an identical 95% success rate. Under normal conditions, both algorithms perform quite similarly, with only a few percentage points of difference in success rates and very similar runtimes. This means that either algorithm could be used interchangeably in such cases.

However, as conditions become more difficult, such as when the FOV decreases, false stars appear in the image, or positional noise is added, the performance of the planar triangle algorithm drops significantly compared to Liebe's algorithm. The difference in success rates between the algorithms can be as large as 30% to 40% points, which is substantial. The computational efficiency also diverges notably under these conditions: Liebe's algorithm maintains a runtime of below 10 seconds in small FOV and false star scenarios, while the planar algorithm's runtime increases to around 25 seconds. Therefore, among the implementations developed in this work, Liebe's algorithm is, by far, the superior star recognition algorithm in all aspects when compared to the planar triangle algorithm.

## 4.5 Critical Reflection

When examining the results, it is important to critically assess the credibility of the outcomes and the quality of the implementation. One must consider which results can be trusted and what could have been done differently.

Overall, the implementation of Liebe’s algorithm can be considered successful based on the results obtained. The success rate closely matches the values reported in the original publications. However, in this implementation, several modifications were introduced, the most significant of which are the additional filtering techniques. The use of an extra side (the so-called scale filtering) and magnitude filtering are new extensions added to the algorithm, enabling the removal of large numbers of matches and the elimination of poor candidates. With these functions present, the algorithm is not a fully authentic version of the original Liebe, but rather an improved variant, as the core operating principle remains the same for both the database and the feature triangles.

There is uncertainty regarding the quality of the Planar Triangle algorithm implementation. Under normal conditions, the algorithm works correctly; however, once the FOV is reduced, the success rate drops significantly, contrary to what the original publication suggests. It is likely that, with further fine-tuning, the algorithm’s performance could be significantly improved, even under the more demanding stress-test scenarios. Adjusting the tolerance limits for accepting feature triangle area and polar moment values, as well as refining the selection of feature triangles and increasing the number of pivot cycles, could improve results. It is also highly likely that the runtime performance of both algorithms could be improved by designing more efficient database structures.

The simulations were successfully designed from the perspective of fair comparison: both algorithms receive identical input data, and the simulated images are always the same for each algorithm. One possible improvement would be to increase the number of simulation runs to obtain more precise results. With the current number of repetitions set to 200, the possible variation in outcomes is approximately  $\pm 1.5\%$ . This can be calculated using the approximate standard error of a proportion, as shown in Equation 11, where  $p$  is the percentage of successful runs and  $N$  is the total number of simulation runs:

$$SE_P = \sqrt{\frac{p(1-p)}{N}}. \quad (11)$$

For instance, increasing the number of simulation iterations to 500 would reduce the variation to below one percent. This would improve the reliability of the results and make the comparison of values that lie close to each other more robust. With the current limited number of iterations, natural variation could even allow the Planar Triangle algorithm to appear superior under normal conditions if the Liebe success rate dropped slightly and the Planar Triangle success rate increased by one percentage point.

## 5 Conclusion

In this final chapter, the content of the thesis is reviewed once more. Section 5.1 provides a brief overview of the topics addressed in this work. Section 5.2 summarizes the key findings of the comparison. Finally, Section 5.3 discusses possible future research directions that could be pursued on this topic.

### 5.1 Summary of Work

This thesis examined the operating principles of two well-known star identification algorithms: Liebe's algorithm and the Planar Triangle algorithm. Both algorithms were implemented in a shared simulation environment using Python, with some minor modifications where necessary. For each algorithm, a star catalog, feature database, feature extraction method, and matching function were constructed as closely as possible following published descriptions.

In the simulation environment, images of the sky were generated using controlled and adjustable settings, which were provided identically to both implemented algorithms for the star identification task. Several different simulation configurations were created to investigate various aspects and weaknesses of the algorithms. These simulations were used, for example, to compare algorithm performance with a reduced field of view, under positional noise, and in scenarios where false stars were added to the image. In total, eight different simulation configurations were tested, and each configuration was run 200 times for both algorithms to ensure reliable results. Data were collected from multiple stages of the algorithms, including the number of initial matches, runtime, filtering efficiency, and final simulation success rates.

### 5.2 Main findings

The simulation results show that, of the two algorithms, the Liebe algorithm performs best in every tested scenario. In the baseline simulation, the success rates of the algorithms were very similar, differing only by a few percentage points, with the Liebe algorithm performing slightly better and operating faster by approximately 0.6 seconds.

The Liebe algorithm also demonstrated strong robustness in the stress test simulations. Its success rate remained close to 100% even when the field of view was reduced and a few false stars were added to the simulated image. Introducing a small amount of positional noise had virtually no effect on the algorithm's performance. The success rate began to decline only when an excessive and unrealistic number of twenty false stars was added to the image; yet, even in this case, the algorithm performed very well, achieving a 79% success rate. The weakest performance occurred under high positional noise, where the success rate dropped to 40%. Even this result is notably strong when compared with the Planar Triangle algorithm, which achieved a success rate of only 1.5% in the same configuration.

For the Planar Triangle algorithm, the stress test scenarios proved to be considerably more challenging. A reduction in the field of view alone lowered its success rate to 64.5%, and performance deteriorated further when false stars and high positional noise were introduced. The only scenario in which the Planar Triangle algorithm matched the performance of the Liebe algorithm was the low positional noise configuration, where both achieved a 95% success rate. The runtime of the Planar Triangle algorithm was also frequently much worse than that of the Liebe algorithm. The Planar Triangle implementation often reached runtimes of around 25 seconds, whereas the Liebe algorithm typically remained below 10 seconds in most scenarios.

Based on these results, the Liebe algorithm is significantly more accurate, reliable, robust, and efficient in the tested simulations.

### **5.3 Future Work**

Several possible directions for future work can be identified. First, the simulation environment could be further developed to create a more realistic testing framework for star identification. Analyzing real star sensor imagery would also be valuable and could provide more accurate insight into how the algorithms perform outside a simulated setting.

Second, the algorithms themselves could be improved and refined. As shown in the results, the Liebe algorithm already performs at a very high level; however, new or alternative filtering methods could be explored and implemented. For example, the clustering approach mentioned in Section 3.2.2 appears to be a promising method for filtering large numbers of outlier feature matches. For the Planar Triangle algorithm, structural modifications to the algorithmic workflow could be considered. In its current form, the algorithm is computationally heavy and time consuming, even with limits placed on the number of triangles and pivot iterations. Expanding these limits, combined with refined tolerance adjustments, may provide a pathway to achieving higher success rates.

Finally, an additional direction is to testing new algorithms in the simulation environment. This thesis has demonstrated that the simulation setup provides a functional platform for algorithm evaluation and comparison; however, there is still room for increased realism. For instance, implementing and assessing neural network based star identification methods, briefly mentioned in Section 2.2.2, could be an interesting direction, allowing for a more detailed comparison against classical approaches.

## References

- [1] David Rijlaarsdam, Hamza Yous, Jonathan Byrne, Davide Oddenino, Gianluca Furano and David Moloney. *A survey of Lost-in-Space Star identification Algorithms Since 2009*. Sensors, 1 May 2020
- [2] Fuqiang Zhou and Tao Ye. *Lost-in-Space Star Identification Using Planar Triangle Principle Component Analysis Algorithm*. Mathematical Problems in Engineering, 2015:1–11, 2015.
- [3] Guangjun Zhang. *Star Identification*. National Defence Industry Press: Berlin, Germany, 2017
- [4] Dustin Lang, David W. Hogg, Keir Mierle, Michael Blanton, and Sam Roweis. *Astrometry.net: Blind astrometric calibration of arbitrary astronomical images*. The Astronomical Journal, 139(5):1782–1800, Mar 2010.
- [5] Carl Christian Liebe. *Pattern recognition of star constellations for spacecraft applications*. IEEE Aerospace and Electronic Systems Magazine, 7(6):34–41, Jun 1992.
- [6] Carl Christian Liebe. *Star Tracker for attitude determination*. IEEE Aerospace and Electronic Systems Magazine, 10(6):10–16, Jun 1995.
- [7] Daniele Mortari, Malak A. Samaan, Christian Bruccoleri, and John L. Junkins. *The pyramid star identification technique*. Navigation, 51(3):171–183, Sep 2004.
- [8] Craig L. Cole and John L. Crassid. *Fast star-pattern recognition using planar triangles*. Journal of Guidance, Control, and Dynamics, 29(1):64–71, Jan 2006.
- [9] Ali Toloiei., Reza Ghasemi. *A comparative analysis of star identification algorithms*. Astrophysics and Space Science, April 2020
- [10] Ali Toloiei., Morteza Shayan Arani, Manouchehr Abaszadeh. *A new composite algorithm for identifying the stars in the star tracker*. Int. J. Comput.Appl. 102(2), 28–33 (2014)
- [11] F. Schwarz, T. Falk, G. Falbel and P. Spangenberg. *System Design and Performance of Earth/Lunar Horizon Sensor, BEC Project 3744* Barnes Engineering Co, October 21, (1966)
- [12] Kara M. Huffman, Raymond J. Sedwick, James Stafford, James Peverill and William Seng. *Designing Star Trackers to Meet Micro-Satellite Requirements*. SpaceOps 2006 Conference, Rome, Italy (2006)
- [13] Andrea Antonello, Lorenzo Olivieri. and Alessandro Francesconi. *Development of a low-cost sun sensor for nanosatellites*. Astra Astronautica, Volume 144: 429:436, March 2018

- [14] Kristian Svartveit. *Attitude Determination of the NCUBE Satellite*. Master Thesis, Department of Engineering Cybernetics, Norges Teknisk-Naturvitenskapelige Universitet (NTNU), Trondheim, Ålesund, Gjøvik, Norway (2003)
- [15] Guangjun Zhang, Xinguo Wei and Jie Jiang. *Full-sky autonomous star identification based on radial and cyclic features of star pattern*. *Image Vis. Comput.* 26(7), 891–897 (2008)
- [16] Benjamin B. Spratling and Daniele Mortari. *A survey on Star Identification Algorithms*. *Algorithms* 2009, 2, 93-107
- [17] David Rijlaarsdam, Hamza Yous, Jonathan Byrne, Davide Oddenino, Gianluca Furano and David Moloney. *Efficient Star Identification Using a Neural Network*. *Sensors*, 20, 3684, (2020)
- [18] Mikaël Marin and Hyochoong Bang. *Design and Simulation of a High-Speed Star Tracker for Direct Optical Feedback Control in ADCS*. *Sensors*, 20, 2388, (2020).
- [19] Paul H. Salomon. *A microprocessor controlled ccd star tracker*. In *AIAA 14th Aerospace Sciences Meeting*; AIAA: Washington, DC, USA, 1976.
- [20] Thomas E. Strikwerda and John L. Junkins. *Star Pattern Recognition and Spacecraft Attitude Determination*. *Astronaut. Sci.* 25, 251–270. (1977)
- [21] Curtis Padgett and Kenneth Kreutz-Delgado. *A grid algorithm for autonomous star identification*. *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, no. 1, pp. 202-213, Jan. 1997
- [22] Erik Høg, Claus Fabricius., Valeri V. Makarov, Stephen E. Urban, Thomas E. Corbin, George L. Wycoff, Ulrich Bastian, Peter Schwekendiek and Andreas Wicenec. *The Tycho-2 Catalogue of the 2.5 million brightest stars*. *Astronomy & Astrophysics*. 355, 27-30. (2000)
- [23] Michael A. C. Perryman, Lennart Lindegren, Jean Kovalevsky, Erik Høg, Ulrich Bastian, Pier Luigi Bernacca, Franco Donati and Michel Grenon. *The Hipparcos Catalogue*. *Astron. Astrophys.* 500, 501–504. 1977.
- [24] Dorrit Hoffleit and Wayne H. Warren, Jr. *The Bright Star Catalog*. *Astron. Data Cent.* 18, 51–64, 1991.
- [25] Benjamin B. Spratling and Daniele Mortari. *The k-vector nd and its application to building a non-dimensional star identification catalog*. *J. Astronaut. Sci.* 58, 261–274. 2011
- [26] Larry Hardesty. *Explained: Neural networks*. <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>, Apr 2017. Accessed: 24-09-2025

- [27] Phillip Alvelda, A. Miguel San Martin, C. Bell, Jacob Barhen. *Spacecraft attitude determination using neural star pattern recognition*. Neural Networks, 1:421, Jan 1988.
- [28] Jian Hong and Julie A. Dickerson. *Neural-network-based autonomous star identification algorithm*. Journal of Guidance, Control, and Dynamics, 23(4):728–735, Jul 2000.
- [29] Marco Mastrofini, Ivan Agostinelli, Francesco Latorre and Fabio Curti. *A convolutional neural network approach to star sensors image processing algorithms*. AAS 21-589, 2021.
- [30] Hongrui Zhao, Michael F. Lembeck, Adrian Zhuang, Riya Shah, and Jesse Wei. *Real-time convolutional neural network-based star detection and centroiding method for cubesat star tracker*. IEEE Transactions on Aerospace and Electronic Systems, page 1–13, 2025.
- [31] TERMA. *T3 Star Tracker*. <https://www.satnow.com/products/star-trackers/terma/37-1195-t3-star-tracker>, Accessed: 21.11.2025
- [32] Jena Optronik GmbH. *ASTRO CL Star Tracker*. <https://www.satnow.com/products/star-trackers/jena-optronik-gmbh/37-1188-astro-cl>, Accessed: 21.11.2025
- [33] Hao Wang, Zhi-yuan Wang, Ben-dong Wang, Zhuo-qun Yu, Zhong-he Jin, and John L. Crassidis. *An artificial intelligence enhanced star identification algorithm*. Frontiers of Information Technology & Electronic Engineering, 21(11):1661–1670, Aug 2020.
- [34] Márcio A. A. Fialho. *Evaluation of centroiding algorithms for an autonomous star tracker*. The European Physical Journal Special Topics. <https://doi.org/10.1140/epjs/s11734-025-01917-0>
- [35] Xiaowei Wan, Gangyi Wang, Xinguo Wei, Jian Li and Guangjun Zhang. *Star centroiding based on fast gaussian fitting for star sensors*. Sensors, 18(9), 2836. 2018.
- [36] Ronald C. Stone. *A comparison of digital centering algorithms*. Astron. J. 1989;97:1227–1237.
- [37] Hyun-Wook Lee, Hyoung-Jun Park and June-Ho Lee. *Accuracy improvement in peak positioning of spectrally distorted fiber Bragg grating sensors by Gaussian curve fitting*. Appl. Opt. 2007;46:2205–2207.
- [38] Márcio A. A. Fialho. *Improved star identification algorithms and techniques for monochrome and color star trackers*. Ph.D. dissertation, Instituto Nacional de Pesquisas Espaciais. Brazil. 2017

- [39] Grace Wahba. *A Least Squares Estimate of Satellite Attitude*. SIAM Review, 1965, 7(3), 409
- [40] Landis Markley and Daniele Mortari. *Quaternion Attitude Estimation Using Vector Observations*. Journal of the Astronautical Sciences, 2000, 48(2):359–380
- [41] Landis Markley. *Attitude Determination using Vector Observations and the Singular Value Decomposition*. Journal of the Astronautical Sciences, 1988, 38:245–258
- [42] Liyan Luo, Luping Xu and Hua Zhang. *An Autonomous Star Identification Algorithm Based on One-Dimensional Vector Pattern for Star Sensors*. Sensors 2015, 15, 16412-16429
- [43] Hongyu Guan et al. *Thermal Deformation Analysis of a Star Camera to Ensure Its High Attitude Measurement Accuracy in Orbit*. Remote Sens. 2024, 16(23), 4567
- [44] Micheal W. Hubbard and Dennis L. Murata . *Radiation-hard breadboard star tracker. Final report*. Sep. 1985.
- [45] Liheng Ma, Dongkai Dai and Yuanman Ni. *How to improve the attitude accuracy of the star sensor under dynamic conditions: A review*. Acta Astronautica, Volume 233, August 2025, Pages 42-54
- [46] Frédéric Devernay & Olivier Faugeras. *Straight Lines Have to Be Straight Automatic Calibration and Removal of Distortion from Scenes of Structured Environments*. Mach Vis Appl. 13. 2001
- [47] Erdem Onur Ozyurt and Alim Rustem Aslan. *A Morphological Approach Of False Star Removal And Camera Motion Estimation For Star Sensors*. 11th Nano-Satellite Symposium, 1-5
- [48] European Space Agency. *ESA Kelvins, Star Trackers: First contact*. <https://kelvins.esa.int/star-trackers-first-contact/> Accessed 21.11.2025
- [49] C. Hughes. *Accuracy of Fish-Eye Lens Models*. Applied Optics, Vol. 49, No. 10, 2010
- [50] Jian Xu, De-Wei Han, Kang Li, Jun-Jie Li and Zhao-Yuan Ma. *A Comprehensive Overview of Fish-Eye Camera Distortion Correction Methods*. arXiv:2401.00442, Decemerm (2023)