MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

CONSTELLATION DETECTION USING TRIANGLE SIMILARITY

LICENSE THESIS

Graduate: **Horatiu-Stefan SPANIOL**

Supervisor: **Assoc.Prof.Dr.Eng Ion-Augustin GIOSAN**

2021

MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,	HEAD OF DEPARTMENT,
Prof. dr. eng. Liviu MICLEA	**Prof. dr. eng. Rodica POTOLEA**

Graduate: **Horatiu-Stefan Spaniol**

LICENSE THESIS TITLE

1. **Project proposal:** *Constellation detection using triangle similarity*

2. **Project contents:** *(enumerate the main component parts) Presentation page, advisor's evaluation, title of chapter 1, title of chapter 2, …, title of chapter n, bibliography, appendices.*

3. **Place of documentation**: *Example*: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**:

5. **Date of issue of the proposal:** November 1, 2020

6. **Date of delivery:** July 8, 2021 (date when document is submitted)

Graduate: ______________________________

Supervisor: ______________________________

MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind
autenticitatea lucrării de licenţă

Subsemnatul(a) Horatiu-Stefan Spaniol, legitimat(ă) cu CI seria CJ nr. 483084 CNP 1980427124604, autorul lucrării ____ ____ ____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică şi Calculatoare, Specializarea ________________________________ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _________________ a anului universitar 2020-2021, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data

____

Nume, Prenume

Horatiu-Stefan Spaniol

Semnătura

De citit înainte (această pagină se va elimina din versiunea finală):

1. Cele trei pagini anterioare (foaie de capăt, foaie sumar, declaraţie) se vor lista pe foi separate (nu faţă-verso), fiind incluse în lucrarea listată. Foaia de sumar (a doua) necesită semnătura absolventului, respectiv a coordonatorului. Pe declaraţie se trece data când se predă lucrarea la secretarii de comisie.
2. Pe foaia de capăt, se va trece corect titulatura cadrului didactic îndrumător, în engleză (consultaţi pagina de unde aţi descărcat acest document pentru lista cadrelor didactice cu titulaturile lor).
3. Documentul curent a fost creat în **MS Office 2007.** Dacă folosiţi alte versiuni e posibil sa fie mici diferenţe de formatare, care se corectează (textul conţine descrieri privind fonturi, dimensiuni etc.).
4. **Cuprinsul** începe pe pagina nouă, impară (dacă se face listare faţă-verso), prima pagina din capitolul **Introducere** tot aşa, fiind numerotată cu 1. Pentru actualizarea cuprinsului, click dreapta pe cuprins (zona cuprinsului va apare cu gri), Update field->Update entire table.
5. Vizualizaţi (recomandabil şi în timpul editării) acest document după ce activaţi vizualizarea simbolurilor ascunse de formatare (apăsaţi simbolul **π** din *Home/Paragraph*).
6. Fiecare capitol începe pe pagină nouă, datorită simbolului ascuns Section Break (Next Page) care este deja introdus la capitolul precedent. Dacă ştergeţi din greşeală simbolul, se reintroduce (*Page Layout* -> *Breaks*).
7. Folosiţi stilurile predefinite (Headings, Figure, Table, Normal, etc.)
8. Marginile la pagini nu se modifică (Office 2003 default).
9. Respectaţi restul instrucţiunilor din fiecare capitol.

Table of Contents

Chapter 1. Introduction

1.1. Project context

The night sky has always been fascinating to humans world-wide. Even early humans must have been looking curiously at the sky, wondering about the lights that can be seen all across the landscape. At some point, humans let their imagination define some patterns in the stars observed, and have associated them to animals, objects, mythological creatures and characters. They also realized that some patterns can be seen in only a specific time period throughout the year and from certain locations on the globe, which allowed them to use these patterns even in navigation.

These patterns are called constellations. Nowadays, the International Astronomical Union recognizes 88 different constellations[1][2] on the celestial sphere. Constellations provide an easy entry in the field for people that take a liking towards astronomy because they can be seen with the naked eye and getting to know their backstories and the meaning behind their names is fun. This information can be refreshed every time while person recognizes one of these constellations while star gazing.

An astronomical image represents an image of the night sky. This image could be taken with the camera in a phone, although a higher quality would definitely be obtained by taking a photo with an actual camera directly or through a telescope. An example of an astronomical image can be seen in Figure 1.1.

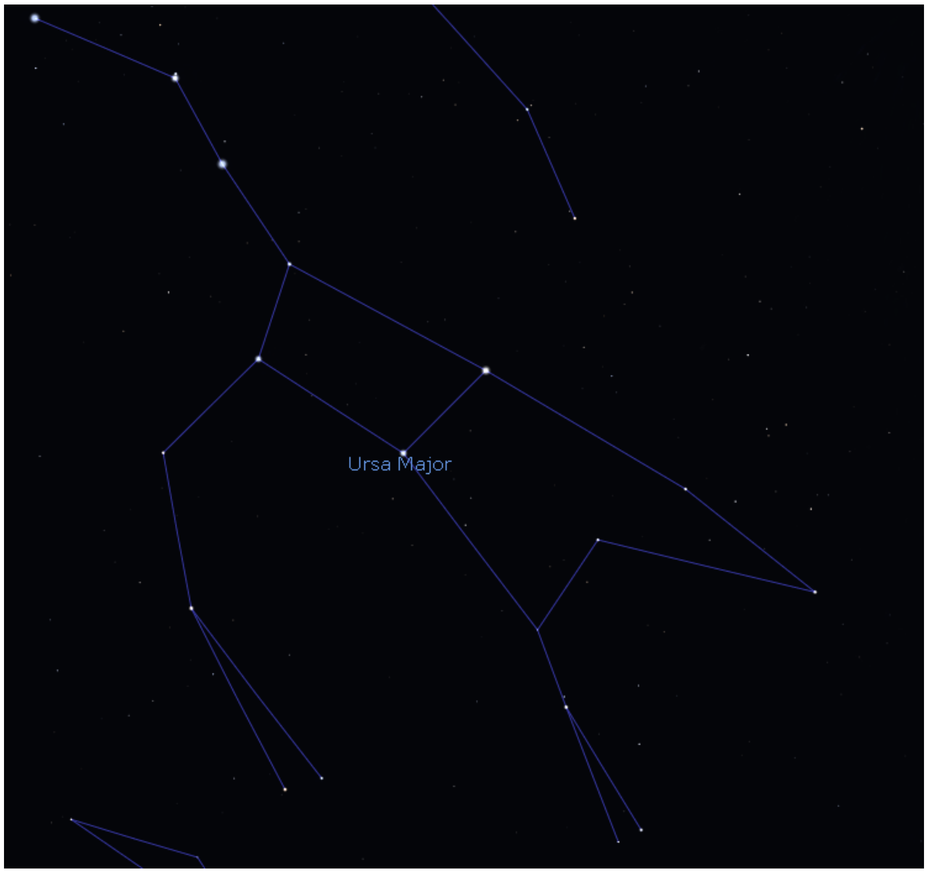


Figure 1.1 Ursa Major constellation as seen in Stellarium[21]

Ursa Major is perhaps one of the most famous constellations, due to being able to be seen throughout the whole year in most of the northern hemisphere.

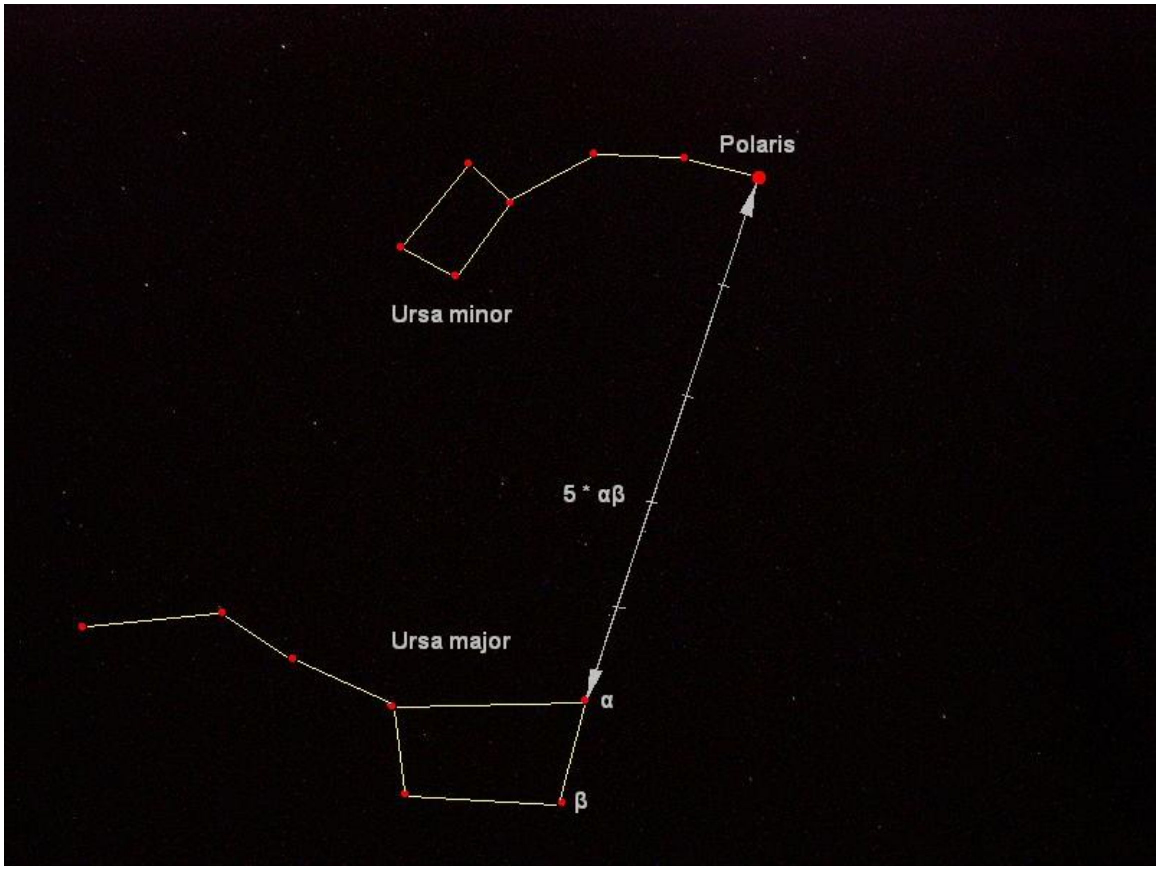


Figure 1.2 Relation between „Big Dipper" asterism and Ursa Minor constellation[22]

It also contains the very visible "Big Dipper" asterism, which mimics the pattern of the "Little Dipper", or the Ursa Minor constellation. The latter has also served its purpose in navigation due to its brightest star, Polaris, being the closest star to the north pole, hence named the north pole star.

Constellations are not easy to spot if not for an untrained eye. Even so, the "manual" detection can be difficult due to the fact that the position of the observer on the globe and the time of the year influence the representation that a constellation might form on the celestial sphere. However, what is important to note is that the overall structure remains the same; the positions of the stars stay relatively the same to each other. This statement is not entirely true as stars are constantly moving but the human lifespan is much too short for that movement to become apparent. Such is why I considered that the overall structure remains constant.

Without having an expectation or information about the way the constellation should appear on the night sky, a certain constellation might be present but the observer might not perceive it as such due to the constellation appearing warped in one way or another. It could be in another location that the observer expects, or the observer might misjudge the angle that the constellation would be rotated at the current time of the day, or the observer might misjudge the scale of the constellation.

Chapter 2. Project Objectives

2.1. Objectives

My project's purpose is to allow amateur astronomers to recognize constellations in their photographs of the night sky. Astronomy enthusiasts might not even realize that certain constellations are present in their astrophotographs, and this application would automatically bring this aspect into attention. The project does not aim to identify each star in the image as it does not use a star catalogue, but to identify the constellation as a whole.

The application should not make any assumptions about the image such as longitude and latitude on the globe from which the photograph was taken, the precise moment in time when the photograph was taken nor should it receive any additional information about the image. Other information that is not required about the image includes details about the field of view, or lens, focal length of scope or the eyepiece, if the image would be taken through a telescope. These data would only decrease the usability of the application, which is not something I want. However, due to the large number of possible combinations between the aspects mentioned previously, the extremely large number of stars that can appear in an astrophotograph and the fact that the only proposed measure of matching is by geometry in nature, it is expected to obtain false positives.

2.2. Specifications

The specifications of the whole project will be divided in either functional or nonfunctional requirements.

2.2.1. Functional Requirements

The application should be a desktop application that takes as input an image provided by the user. This image could be in several formats, such as Windows bitmaps (.bmp, .dib), JPEG files(.jpeg, .jpg, .jpe), Portable Network Graphics (.png) or other widely known image formats. The image could also be of an arbitrary size. The system should be able to read and interpret any type of images described above. It should also be able to create an image of the same size, altered in such a way that the constellation detected is clearly visible and identified by the user of the application.

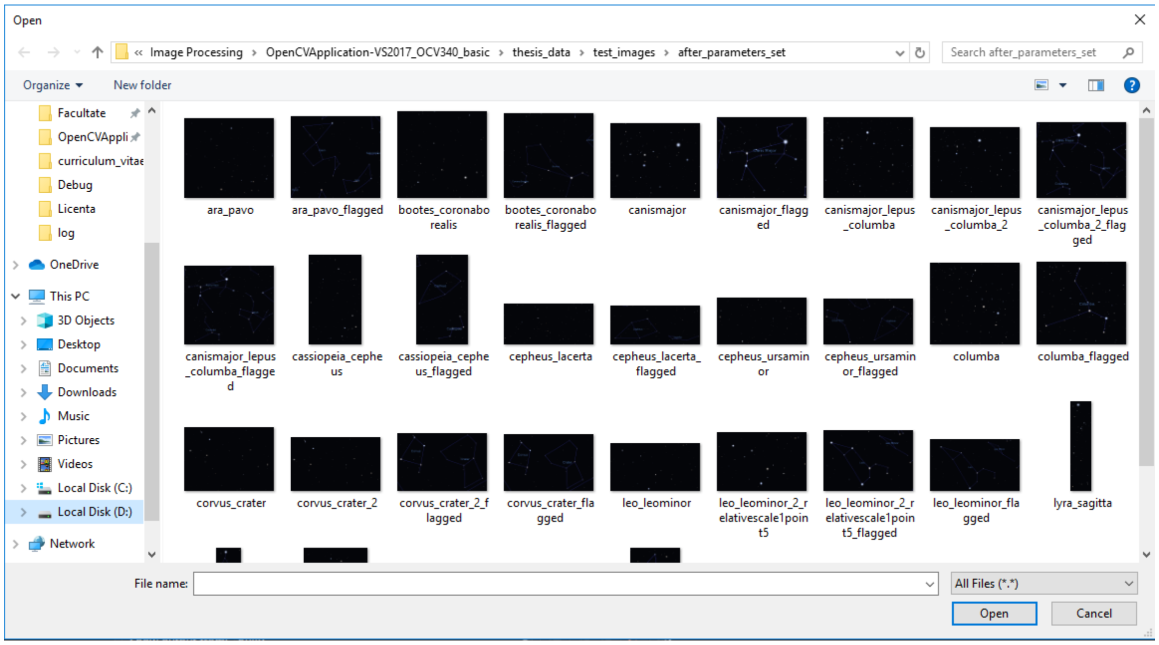


Figure 2.1 Example of a file dialog used to provide the application with an image

The application should be able to convert the image read into a grayscale image, as the primary colors in our domain are black and white.

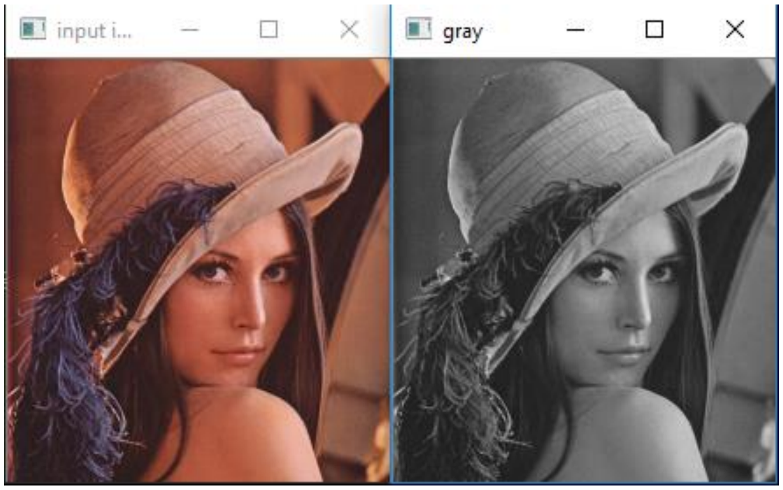


Figure 2.2 Example of image conversion from color to grayscale

Also in the same context, the system should convert the image from an 8-bit grayscale into a 1-bit grayscale, or a binary image, consisting of white points represented by objects, or stars, and black points consisting of the sky which is perceived as background.

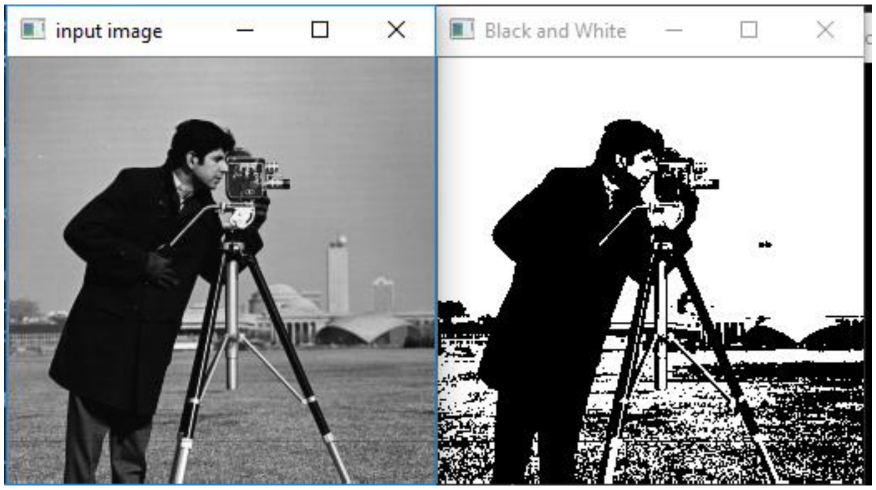


Figure 2.3 Example of image conversion from grayscale to binary

After the process of binarization, the system should be able to begin the process of labeling, which will differentiate stars from each other.

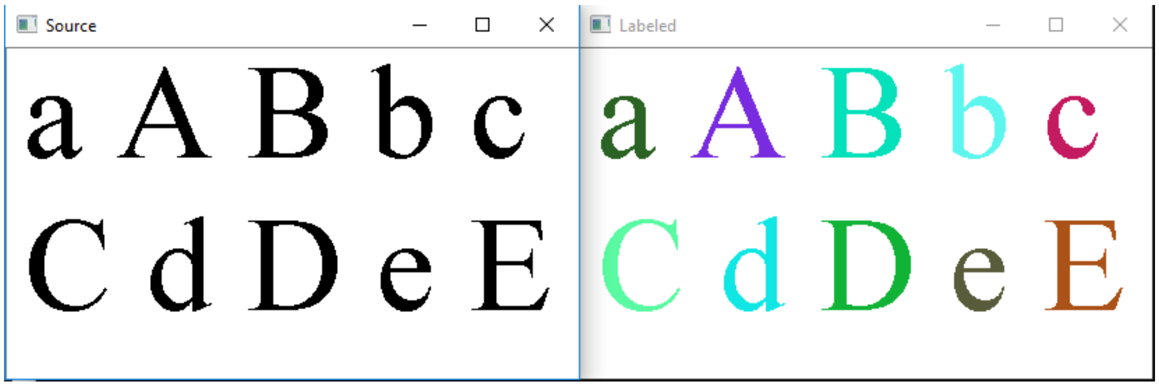
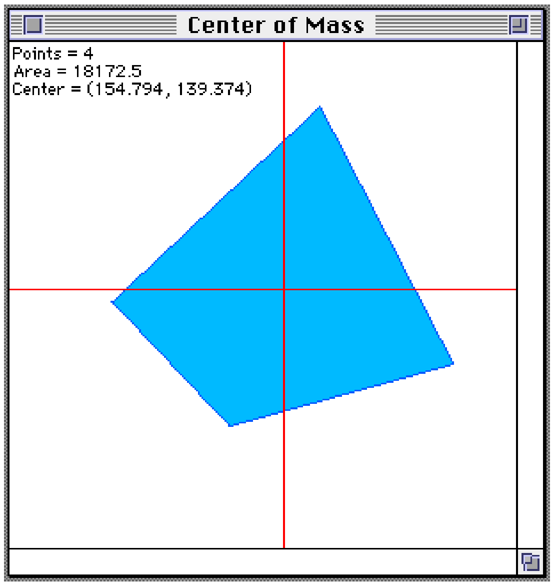


Figure 2.4 Example of binary image labeling

Once the stars can be differentiated, the application should compute the centers of mass and the areas of each star in order to represent them as a single point in the image.



```
Points = 4
Area = 18172.5
Center = (154.794, 139.374)
```

Figure 2.5 Example of area and center of mass of a polygon [23]

When processing the constellation models, the system should be able to generate triangles, compute the lengths of each side of the triangles, normalize the lengths to the smalles one and then store them for later use in detection. When processing images from a user for detection, the system should be able to follow the same process regarding triangle generation and then start the detection process by comparing triangles from constellation models with triangles from the input image and finding similar pairs. Furthermore, the system should be able to match a constellation model in the source image by warping the model based off a pair of similar triangles previously found. The application should be able to filter through detections and provide the most likely detection for said constellation, which should then be drawn on the original source image in order to provide the user the visual aid necessary. Finally, said altered source image should be saved as a separate file with the time stamp of the end of execution.

2.2.2. Nonfunctional Requirements

2.2.2.1. Performance

The application's most important nonfunctional requirement is the performance. Since it is intended for amateur enthusiasts of astrology, this application should be able to produce results in the fastest time possible in order to not lose the interest of the user. However, since the system would be running and performing computations on the computer of the user, the performance is dependant on the actual hardware of the computer. More exactly, the application should see performance increases with a better CPU.

2.2.2.2. Usability

The second most important nonfunctional requirement is the usability. The application should be able to be used with effectiveness and without problems even by a novice computer user. It should provide a familiar way for the user to upload the image and then show the results in a way that is simple to interpret.

2.2.2.3. Availability

The third most important nonfunctional requirement is the availability. Since this application is ran on the computer of the user, and does not rely on the internet whatsoever, the system is always available for use.

2.2.2.4. Scalability

The fourth most important nonfunctional requirement is the scalability. The design of the system should allow for easy manipulation in order to modify the existing pipeline process of image manipulation, whether by adding new intermediary or end processes.

2.2.2.5. Reliability

The fifth most important nonfunctional requirement is the reliability. Ideally, the detection algorithm should be able to produce correct results with little to none false detections. However, due to the nature of the thesis, this is very hard to achieve. The fact that the application does not use any other information regarding constellations other that distances between stars, it is hard to determine when a detection is false. A correct result should be manageable but a highly magnified image of the night sky where a high number of visible stars is present is bound to encumber the performance and reliability of the detection. Due to this high number of stars, coincidences can occur where a pattern of stars appears to look like a certain constellation and the detection algorithm will falsely detect the mentioned constellation. This is possible because the models and the images don't always match one-to-one, and a certain position displacement error should be maintained and allowed for each star.

Chapter 3. Bibliographic Research

3.1. Introduction

In this section, similar systems and papers will be presented in order to provide justifications for the decisions taken in the development of the application.

3.2. Related Work

3.2.1. Evaluation of Star Identification Techniques

In paper [3] the authors present a detailed evaluation regarding star identification using various algorithms. These algorithms make use of a star catalogue and the paper also discusses the difficulties implied when designing such a system. A star catalogue typically represents stars as vertices in an undirected graph and optionally other features, such as brightness or precomputed distances between stars. The detection problem can be more technically described as finding two isomorph graphs: one belonging to the input image, and one belonging to the star catalogue. Two such algorithms described by the paper are a triangle star identification algorithm and a match group star identification algorithm.

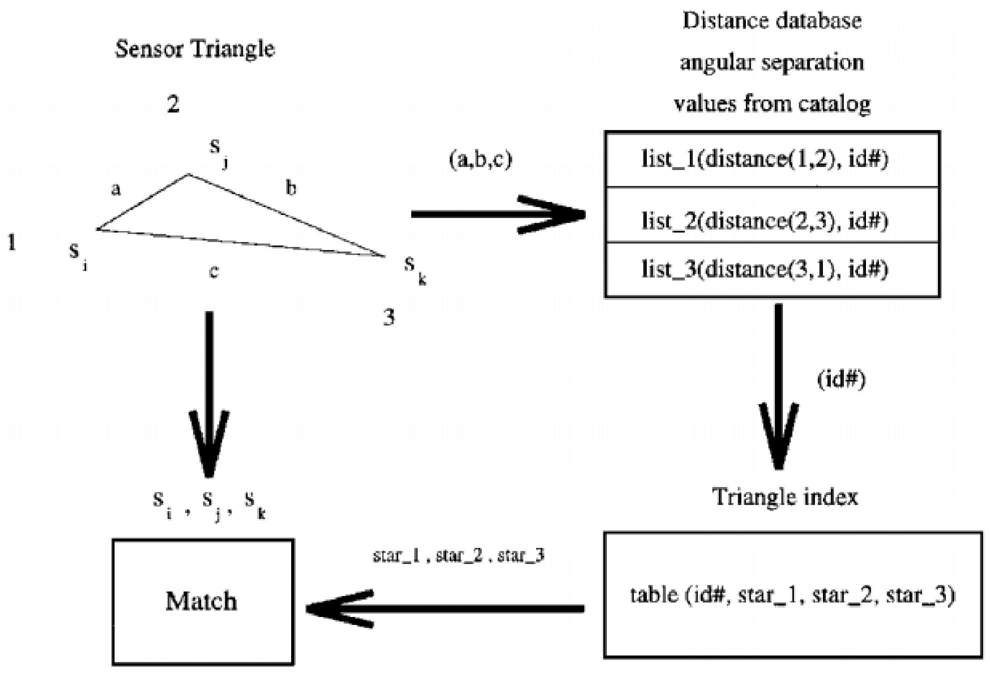


Figure 3.1 Triangle star identification algorithm [3]

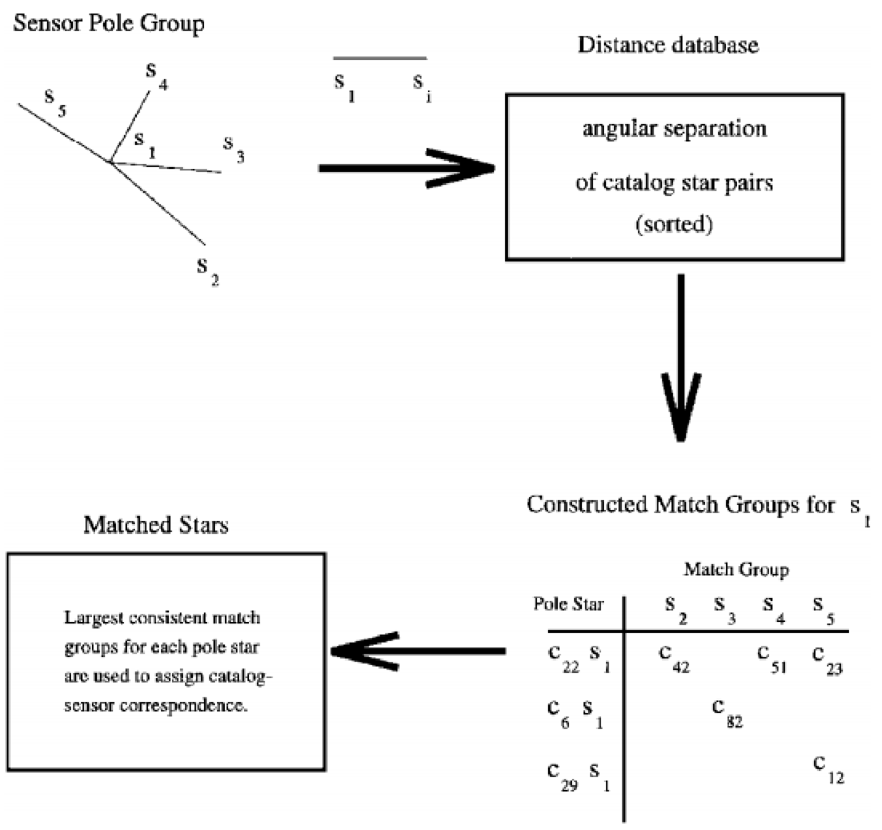

Sensor Pole Group
s4
s5
s1
s3
s2
s1 si
Distance database
angular separation of catalog star pairs (sorted)
Constructed Match Groups for s1
Match Group
Pole Star
s2 s3 s4 s5
c22 s1
c42
c51
c23
c6 s1
c82
c29 s1
c12
Matched Stars
Largest consistent match groups for each pole star are used to assign catalog-sensor correspondence.

Figure 3.2 Pole star or match group algorithm [3]

The triangle algorithm compares the lengths of the sides of triangles in order to prove similarity while the pole star algorithm chooses a star and compares the distances of from the selected star to neighbouring stars with distances from the catalogue.

Since a star catalogue is built in order to be used by a specific system, it should be tailored to it; this means that the precomputed distances between stars and the positions of the stars should leave little to no room for variation and the matches should be very strict. In my case, this is hardly possible as the application is intended to be used by amateur enthusiasts that capture photos of the night sky with various devices. Therefore it is necessary to provide the system with a certain room for error.

3.2.2. Lost-In-Space Star Identification Using Planar Triangle Principal Component Analysis

The same concept of the triangle algorithm is also described in paper [4] for use in devices called star sensors. A star tracker, or sensor, is usually mounted on interstellar objects such as satellites in order to determine their orientation in space.

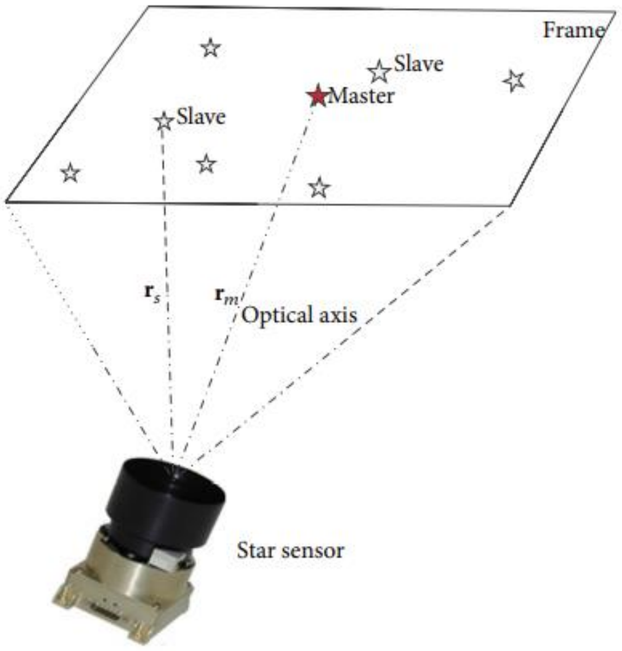


Figure 3.3 Star sensor capturing image [4]

The mentioned article uses a simulated star sensor with a resolution of 1024x1024 in order to form images as can be seen in the figure above. It also is focused on transposing the stars and subsequentially, the triangles, from a sensor frame in order to use principal component analysis for matching.

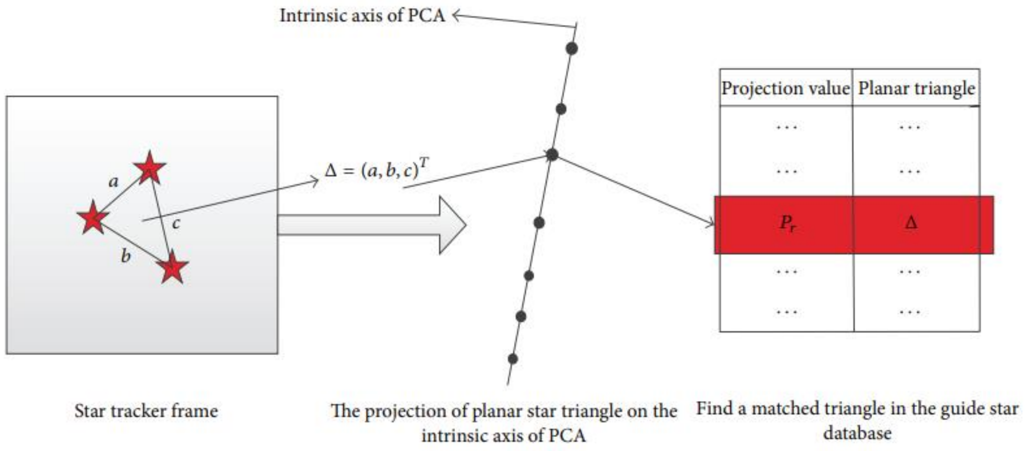

Intrinsic axis of PCA
$\Delta = (a, b, c)^T$
a
b
c
Projection value
Planar triangle
P_r
Δ
Star tracker frame
The projection of planar star triangle on the intrinsic axis of PCA
Find a matched triangle in the guide star database

Figure 3.4 Diagram of projection geometry [4]

The hardest detection is when the system has to rely entirely on the current image data, and is called lost-in-space. This is also the type of detection to be designed an implemented in this thesis, as requiring other information would substantially decrease the usability of the application.

3.2.3. Neural networks

In this section, I shall present a couple articles and papers that use neural networks in order to solve the problem of star identification. While the second article does focus on finding a subset of stars, the first one focuses on single star identification. It is important to note this is not the final goal of my thesis, as I consider single star identification less intriguing to an amateur astronomy enthusiast.

3.2.3.1. Efficient Star Identification Using a Neural Network

The [5] article also tackles the problem of lost-in-space star identification, however in a different way than previously discussed. The method used to solve this problem is a pattern feature extraction method. The authors report that even with a small and fully connected neural network, results can be very promising as long as the feature extraction method is appropriate.

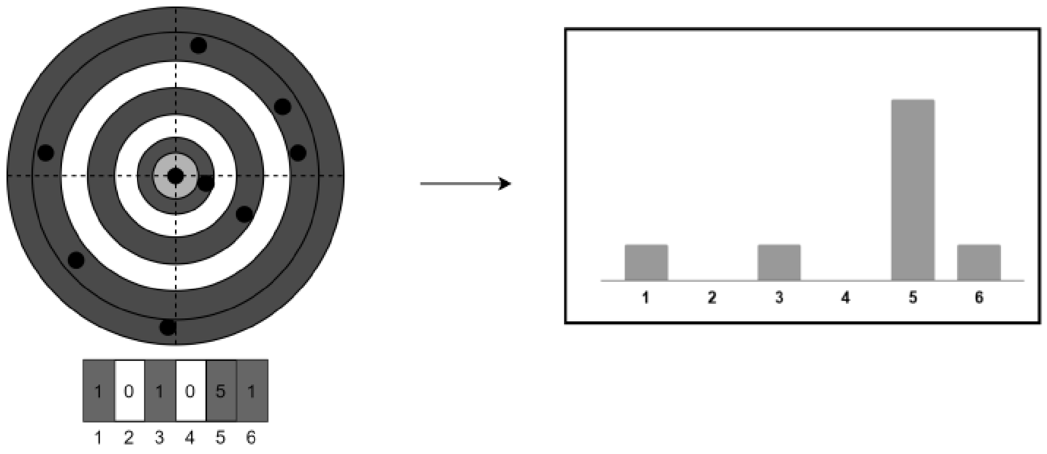


Figure 3.5 Feature extraction method in [5]

The feature extraction presented in this article makes use of a distance distribution of stars from a selected polestar in order to create a histogram. The input of the discussed network is the distribution shown above. In the next figure we can see the proposed network architecture in this paper.

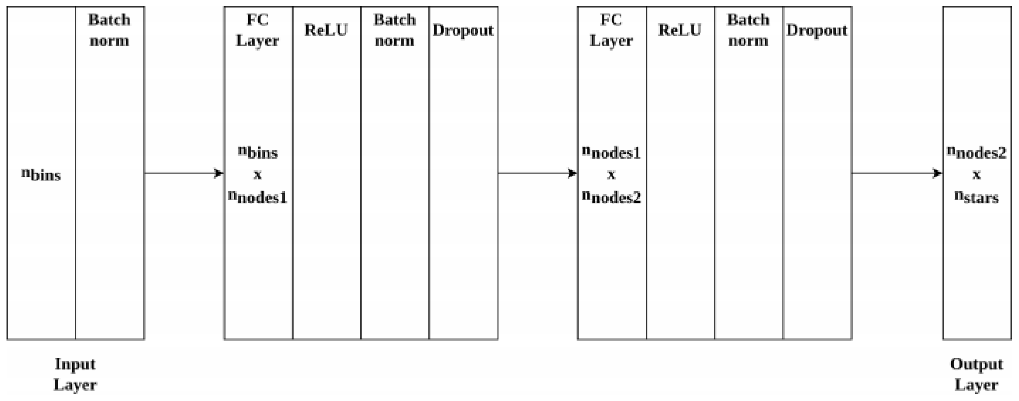


Figure 3.6 Proposed architecture [5]

This is a general architecture that can be modified and tailored to a specific application environments, as the nnodes1 and nnodes2 variables are determined through experiments and the layer sizes can also be dependant on the application enviroment.

3.2.3.2. Neural Network approach to star field recognition

Paper [6] make use of hopfield neural networks for star field recognition. These are a type of recurrent artifical neural network that has the connections across its layers of the same size.

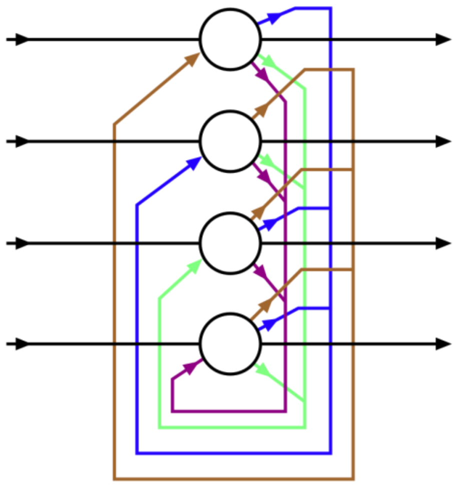


Figure 3.7 Example of a
Hopfield network with four units [7]

As mentioned in the abstract of the article, the detection process uses a compatibility function in order to compute similarity between stars in the field of view and stars present in the catalogue. This function uses information regarding the position and the magnitude of the stars, much like this thesis. Another interesting feature is that the orientation of the field of view is not required, thus this system can also be used for the more challenging lost-in-space star identification problem. The positional information is used in order to compute the angular separation between the stars in the field of view. Then, the star field recognition is based on comparison of angular separations between the stars and the magnitude of each star that belongs to the triangle(see figure below).

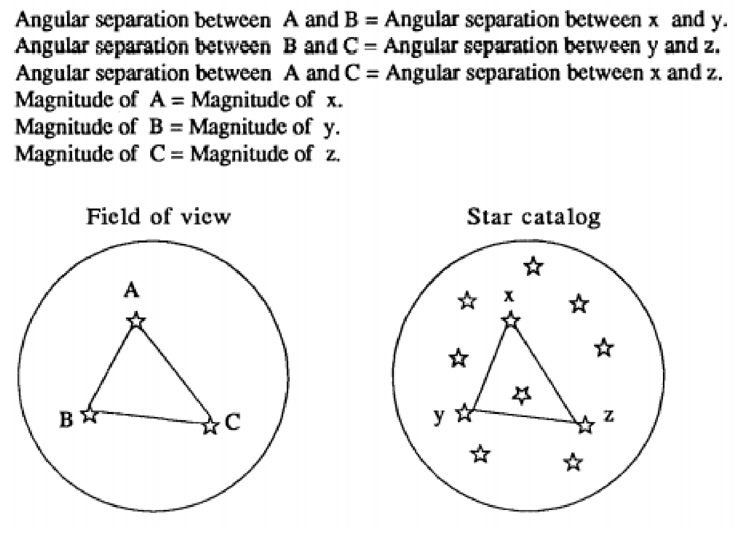

Angular separation between A and B = Angular separation between x and y.
Angular separation between B and C = Angular separation between y and z.
Angular separation between A and C = Angular separation between x and z.
Magnitude of A = Magnitude of x.
Magnitude of B = Magnitude of y.
Magnitude of C = Magnitude of z.

Figure 3.8 Recognition process from [6]

The system creates hypotheses between each star in the input image and each star in its catalogue, computing compatibility of two such hypotheses through a fuzzy function. Then, by using the parameters discussed previously, a function f computes the similarities of the two stars that belong to the field of view from the hypotheses and the stars that belong to the catalogue. The final goal of the network is to find a set of hypotheses that are pair-wise compatible and classify each star in the field of view.

The paper reports 50 correct results out of 50 test images, each image containing 5 stars. Tests were also done after inducing some noise the images, to which the system also responded with the same success. Although the number of stars in the field of view is fairly low, the authors recommend parallelization in order to achieve better results for a larger number of stars.

3.2.4. Constellation Detection

Suyao Ji, Jinzhi Wang and Xiaoge Liu from Stanford University wrote an interesting paper[8] regarding constellation detection. Until now, the examples I provided had somehow of a different scope than [8], in the sense that they tackled the problem of star pattern recognition in general, while [8] is concerned with detecting the 88 constellations recognized by the International Astronomical Union. It’s also worth noting that that a constellation is indeed a star pattern, so there isn’t much difference between these two concepts except for the fact that constellation detection has a known set of patterns.

The authors implemented their own star catalogue, or constellation database as mentioned in the paper. The latter name is more suitable as it the focus is on the detection of the constellation as a whole. An original template of the Gemini constellation can be seen in the next figure:

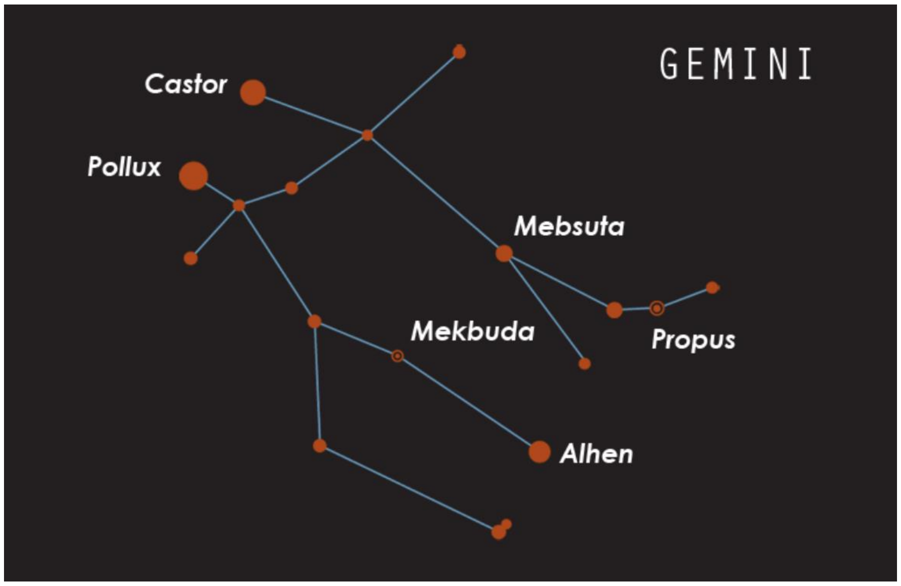


Figure 3.9 Template of Gemini constellation used in [8]

This template goes through various processing steps in order to be able to be used as part of the detection process, such as binarization, filtering of unrelated features and recording of the stars and connections between them. For building the database, each transformation of the template also holds a reference to the brightest two stars and thus will be normalized to the distance between these two stars. One more important aspect is the use of constellations neighbors that are also being stored, which are used in a filter in the detection process.

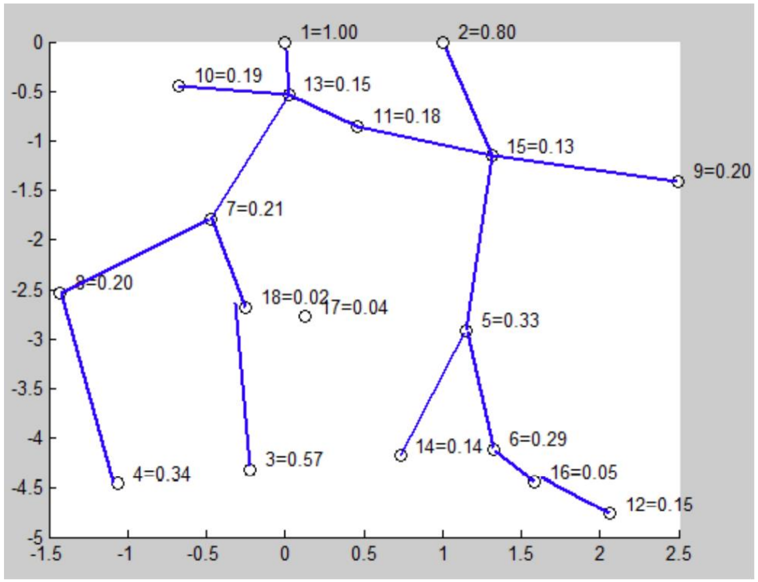


Figure 3.10 Visualized descriptor of Gemini from [8]

The previous figure shows how a visualized descriptor of Gemini would look after the processing of the template. The brightest two stars being placed on the at the (0,0) and (0,1) positions, and the rest of the template is normalized with respect to the distance between these two stars.

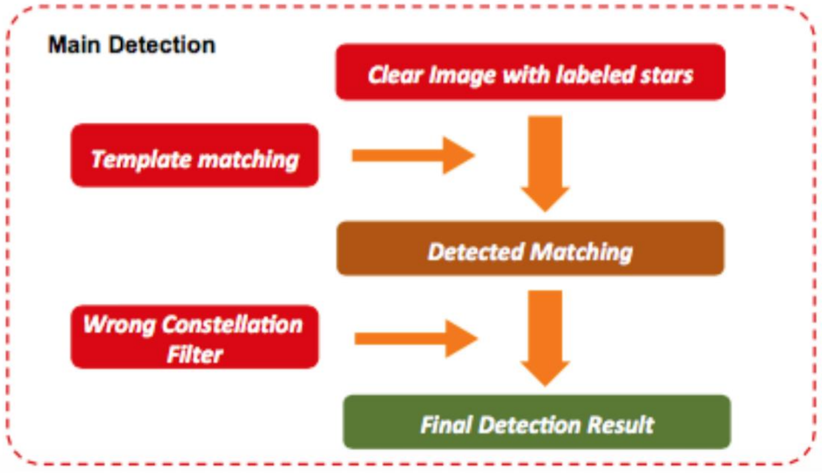


Figure 3.11 Detection process from [8]

The detection process is next discussed. The input image is binarized and then the stars in the image are labeled and sorted by their brightness. The constellation templates are also sorted by their total number of stars. Then, the brightest two stars in the input image are selected in order to compute a scale and rotation for the template. Next, by using the precomputed scale and rotation angle, the input image is checked to see if there are stars where the template indicates there should be, and if there are more than half the number of stars from the constellation present, the match is considered successful. The scale factor is also used to filter out incorrect constellations as the paper points out that multiple constellations in an image should have a close values for the scale factor. The important thing to note here is that this filter assumes the first detection is correct. If the detection would be incorrect, it would disrupt the detection of all the other constellations, which would in turn result in either none or false detections.

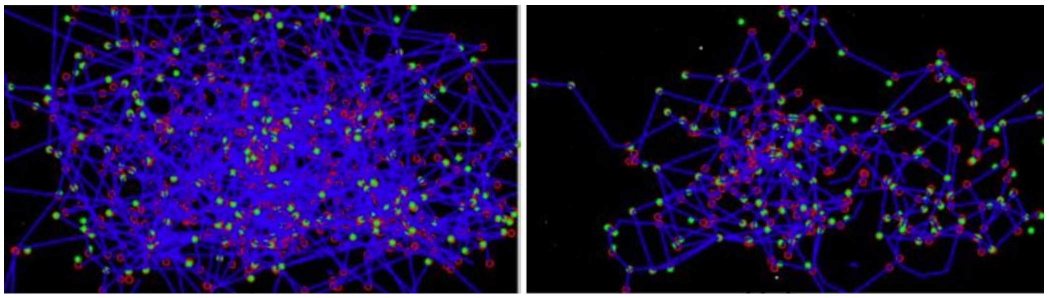


Figure 3.12 Scale filter from [8]

The scale filter does however reduce the number of detections by a large amount, as can be seen in the previous figure. The concepts of constellation neighbors and overlapping stars are used in a final filter before providing the resulting image. The system finds the first constellation that is considered as correct, and then any other detection that shares stars with the first correct detection is eliminated. Then, any other constellation found that is not a neighbor of the correct constellation is eliminated. If another correct constellation is found, meaning it was a neighbor of a previously correct constellation, the process is repeated. A final result is seen in the next figure:

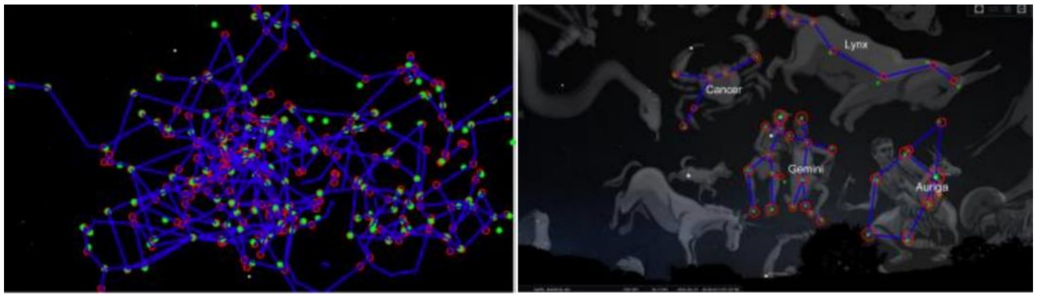
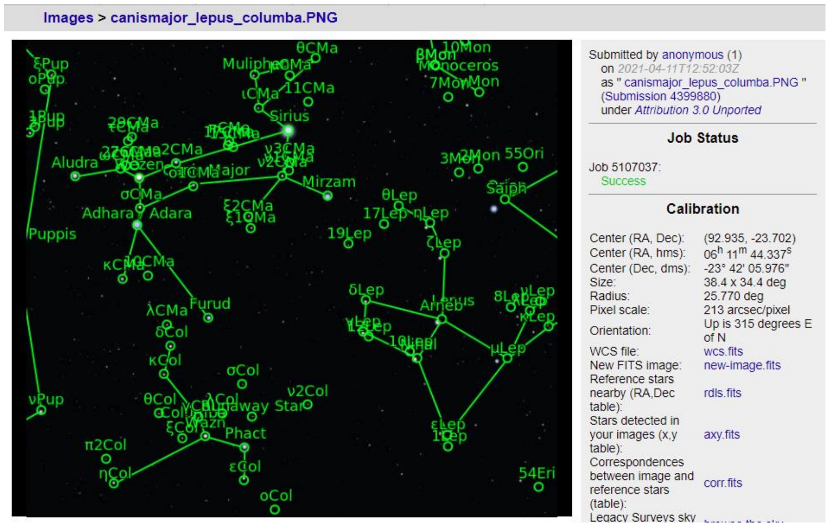


Figure 3.13 After applying the constellation neighbour and overlapping stars filter [8]

The paper reports a 92.8% correct constellation detection percentage, with a 71.4% correct test image detection percentage and an average detection speed of 85 seconds/image. While the correct percentages are impressive, I consider the detection speed to be rather large for the purposes of this thesis.

3.2.5. Astrometry.net

Astrometry.net[9] is a project for an "astrometry engine" that can provide astrometric calibration meta-data from any astronomical image. It can also recognize and classify the stars in the input image, as well as other celestial objects such as galaxies, nebulae etc. The project can be used currently in 3 different ways: a beta version web service that allows any user to upload a photo, on the flickr.com web site where a photo can be uploaded and then submitted to the astrometry group to be checked and through a separate software package.



Submitted by anonymous (1)
on 2021-04-11T12:52:03Z
as " canismajor_lepus_columba.PNG "
(Submission 4399880)
under *Attribution 3.0 Unported*

Job Status

Job 5107037:
Success

Calibration

Center (RA, Dec):	(92.935, -23.702)
Center (RA, hms):	$06^h\ 11^m\ 44.337^s$
Center (Dec, dms):	-23° 42' 05.976"
Size:	38.4 x 34.4 deg
Radius:	25.770 deg
Pixel scale:	213 arcsec/pixel
Orientation:	Up is 315 degrees E of N
WCS file:	wcs.fits
New FITS image:	new-image.fits
Reference stars nearby (RA,Dec table):	rdls.fits
Stars detected in your images (x,y table):	axy.fits
Correspondences between image and reference stars (table):	corr.fits
Legacy Surveys sky	

Figure 3.14 Example of astrometry.net web service detection

An example of submitting a photo to the web service can be seen in the figure above, where various calibration results are displayed on the right side. In the picture, we can see that the engine can classify both individual stars and draw the constellations they belong to. For example, in the top-center we can see Canis Major, in bottom left we can see Columba and in the right we can see Lepus. There are also parts of the Puppis, Monoceros and Orion constellations present, as the results indicate.

Astrometry.net reports some incredible preliminary results at the Sloan Digital Sky Survey such that for 336,554 fields science grades, the engine had 0 false positive, 99.84% solving percentage with a total of 530 unsolved, and a 99.27% solving percent while considering the 60 brightest objects in the field of view.

The project solves the problem of searching by using the idea of an inverted index. First of all, a complex sky catalogue was built, using the USNO-B sky catalogue that contains 10^9 stars and galaxies and the TYCHO-2 sky catalogue, that contains 2.5 million stars. The result catalogue has somewhere around 144 million stars. Then, for the matching process, the idea that of an inverted index. Some selections of 4 stars are made from the input image, and such an inverted index can point out where on the sky catalogue these selections can be seen.

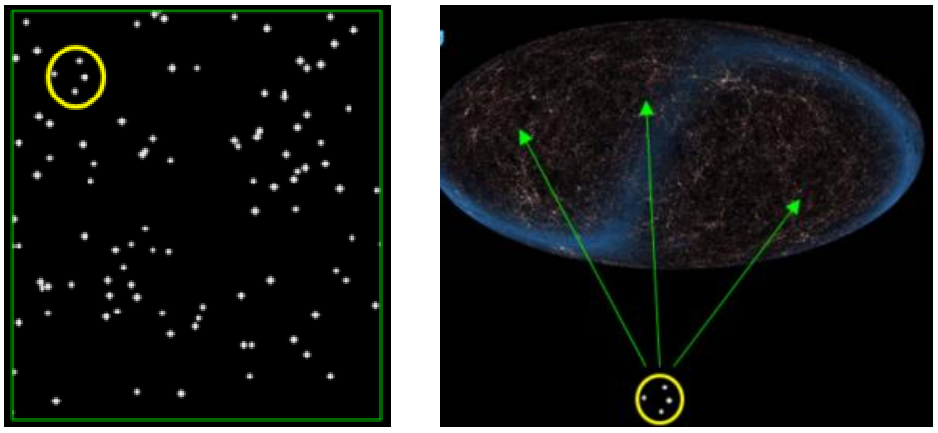


Figure 3.15 Example of selection of 4 stars [9]

By choosing multiple such selections, the engine can determine the most likely part of the catalogue that matches the image by intersection.

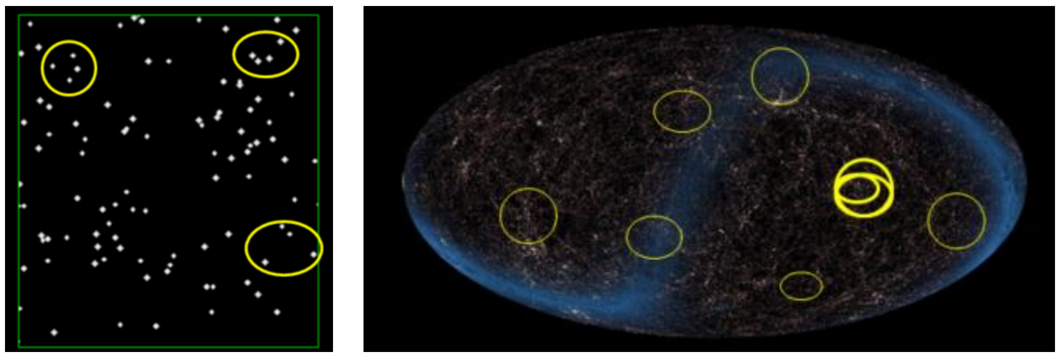


Figure 3.16 Example of multiple selection and intersection of views [9]

The encoding of the four stars is also a very interesting subject of this engine, as each quad defines its own coordinate systems by the most separated pair of points. The positions of the other two points define a 4-dimensional hash code.

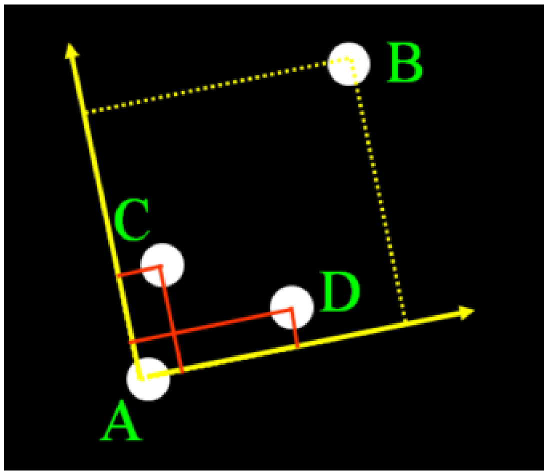


Figure 3.17 Initial coordinate system defined by points A and B [9]

By using the hash code, the quad becomes invariant to scale translation and rotation.

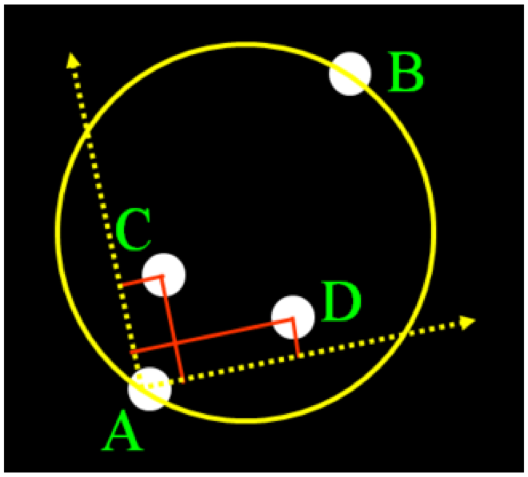


Figure 3.18 Hash code determined by all points [9]

This geometric hashing is part of the reason why this engine is so fast and provides such great results. The other part is the data structures used, which is a fancy implementation of kd-trees that is pointer-free. The tree is built in such a way that by storing the nodes in a linear array, the children of a node at index n can be accessed at index 2n+1 for left child and 2n+2 for right child.

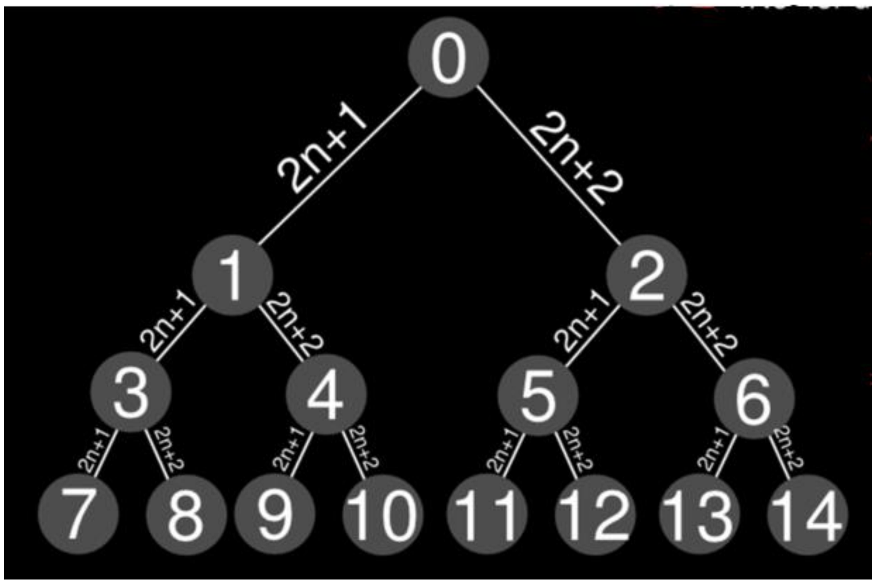


Figure 3.19 Example of the implementation of the kd-tree [9]

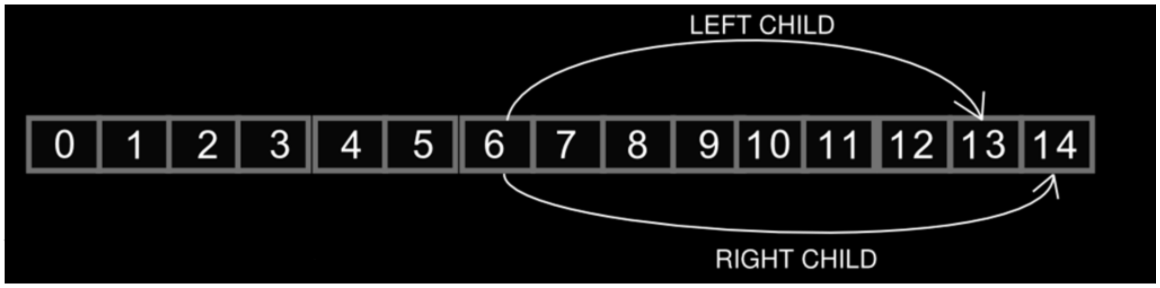


Figure 3.20 Previous kd-tree nodes stored in a linear array, and accessing through the indexing method [9]

Chapter 4. Analysis and Theoretical Foundation

4.1. Technologies and concepts used

4.1.1. Geometric similarity and triangles

Two objects are said to be similar in Euclidean geometry when one object can be obtained by applying operations such as scaling, translation, reflection or rotation to the other object. More simply described, two objects are similar if they have the same shape.

Translation is a transformation that moves a figure in a certain defined direction by a given distance. It can also be represented as adding a vector to each point in the figure. A translation matrix is of the form:

$$T_{\mathbf{v}} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.1 Translation matrix with vector v

Scaling is a transformation that affects the size of the figure. By going through a scaling transformation, the figure can either be enlarged or shrinked by a certain scale factor. Scaling can either be uniform on non-uniform. The first type maintains the scale factor the same for all axis directions, which results in a similar figure to the initial one, while non-uniform scaling has at least one different scale factor, considering all axis directions. For scaling an object with a vector v, a scaling matrix is of the form:

$$S_v = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix}$$

Figure 4.2 Scaling matrix with vector v

Rotation, in Euclidean geometry, represents a transformation of a figure around a point that represents the center of rotation and an angle of rotation. A rotation matrix for a rotation around origin O by an angle θ is of the form:

$$\mathrm{Rot}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Figure 4.3 Rotation matrix with angle θ

Reflection in two dimensions is a transformation of a figure around an axis. The reflections of a figure is the mirror image with respect to the axis. A reflection matrix for a reflection with respect to a line that passes through the origin and forms an angle θ with the x axis is of the form:

$$\mathrm{Ref}(\theta) = \begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix}$$

Figure 4.4 Reflection matrix with angle θ

These four transformations are represented in the figures below:

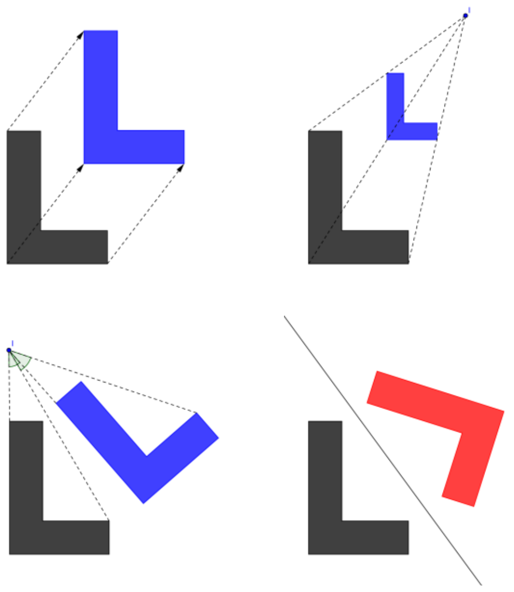


Figure 4.5 Examples (in clockwise order) of translation, scaling, reflection and rotation [10]

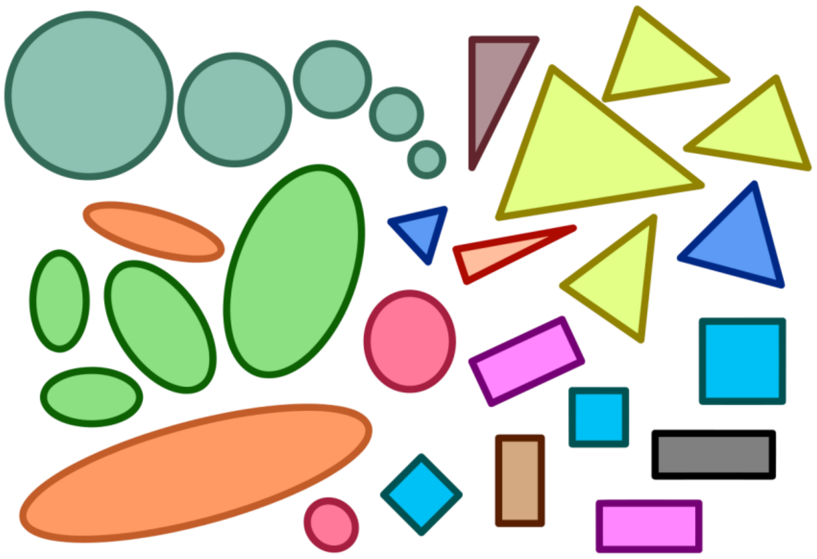


Figure 4.6 Similar figures represented with the same color [10]

Triangle similarity can be proven through numerous statements in Euclidean geometry, each being sufficient by itself to demonstrate the similarity property:

- Two triangles in which all of their angles are congruent are similar. [11]
- Two triangles in which all the corresponding lengths of the sides of the triangles have the same ratio are similar. [11]
- Two triangles in which 2 of the corresponding lengths of the sides of the triangles have the same ratio and the angle between those 2 sides are congruent are similar. [11]

4.1.2. Degeneracy and triangles

A degenerate case in mathemathics represents a special case in a class of objects that differs from the generic cases [13]. Degeneracy cases usually transform the object from the generic dimension of the class to another dimension, which is a clear indication of degeneracy.

An example of a dimensionality reduction would be the degenerate case of a circle, an object of dimension 2, with the radius of 0 turns into a point, which is an object of dimension 0. For triangles, the degenerate case consists of either at least one of the angles having a value of 0 degrees, either at least one of the lengths of a side being 0.

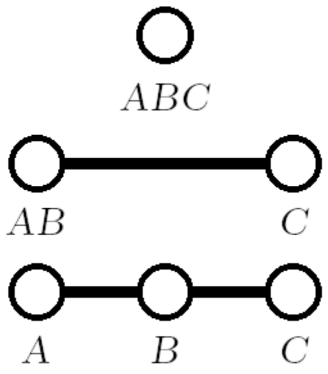


Figure 4.7 Examples of degenerate triangles [13]

4.1.3. Distance

Distance is a metric that can be computed in many different ways in the field of image processing. Due to this, I shall furthermore present three types of distance metrics.

4.1.3.1. Manhattan distance

The Manhattan distance[14], also called Manhattan length, city block distance and many other names defines the taxicab geometry; this makes reference to the city of Manhattan, in which most of the streets are displayed in a grid-like pattern, and the distance geometry refers to the easiest way to navigate the grid in order to get from one point to the other. The Manhattan distance is not unique, as can be observed in the next figure:

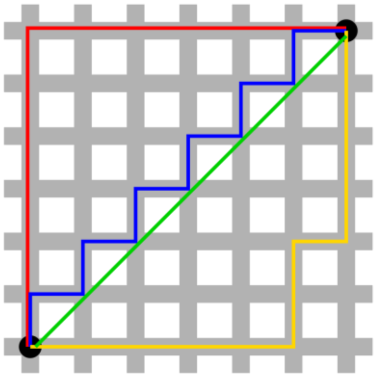


Figure 4.8 Manhattan distance in red, blue and yellow [14]

As mentioned previously, the Manhattan distance between the black dots depicted by Figure 4.8, can be seen colored with red, blue and yellow. All of the options have the same distance of 12, however it can clearly be seen they are very different. Given two n-dimensional vectors p and q, such that p=(p1, p2, ..., pn) and q=(q1, q2, ..., qn), the Manhattan distance can be computed by the total sum of the absolute difference between respective components:

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

Figure 4.9 Manhattan distance formula [14]

4.1.3.2. Chebyshev distance

The Chebyshev distance[15], named after the Russian mathemathician Pafnuty Chebyshev, is a distance metric that can be observed in the movement of the king or queen piece in the game of chess. Given two n dimensional vectors x and y, such that x=(x1, x2, ..., xn) and y=(y1, y2, ..., yn), the Chebyshev distance can be computed by choosing the maximum absolute distance between respective components of the vectors:

$$D_{\mathrm{Chebyshev}}(x, y) := \max_i(|x_i - y_i|).$$

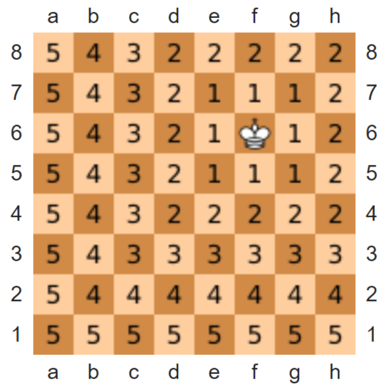
Figure 4.10 Chebyshev distance formula for n dimensional vectors [14]

In two dimensions, this formula becomes:

$$D_{\mathrm{Chebyshev}} = \max\left(|x_2 - x_1|, |y_2 - y_1|\right).$$

Figure 4.11 Chebyshev distance formula for 2 dimensions [14]

A visual representation of the king distance in chess can be seen in the next figure:



	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1	♔	1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

Figure 4.12 Chebyshev distance in chess [15]

The king can move one square horizontally, vertically or diagonally. As we can see, the king is situated on the f6 square and each square that surrounds it has a Chebyshev distance of 1. If we would want to move the king to d7, which has a Chebyshev distance of 2, we can take the e7-d7 route or the e6-d7 route. This distance metric also does not provide a unique solution.

4.1.3.3. Euclidian distance

The Euclidean distance[16] is a metric in Euclidean space, that defines the distance between two points as the length of the segment whose ends are said points. It uses Pythagora's theorem, as the segment can be represented as the hypothenus of the right angled triangle. Given two points p(p1, p2) and q(q1, q2), the distance can be computed by the square roof of the sum of the differences between coordinates that reside on the same axis. This can be seen in the figure below:

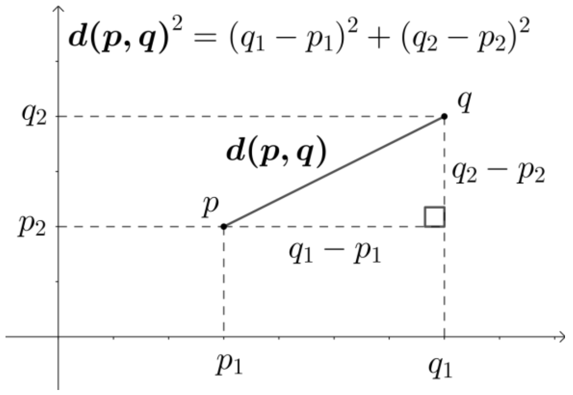


Figure 4.13 Euclidean distance [16]

It is clearly seen that the Euclidean distance between two points is more computational intensive than the Manhattan distance and the Chebyshev distance. However, Euclidean distance provides a unique solution and much more accurate results than the other two metrics.

4.1.4. Area

The two dimensions, the area of an object represents the measure of the total surface covered by a defined object or figure. This is usually expressed in square meters and it is computed with different formulas depending on the type of figure, as this surface would contain an infinite number of points. In image processing however, this can be simplified as there are a finite amount of pixels that compose an image. Therefore, the area of an object can be determined by simply counting the amount of pixels that make up the object, given that those pixels are known and are attributed to the object via a label. This method is not as exact, as it can only provide us with an integer as the value of the area, but it is much more simple to compute.

We can conceive a function I of an object i that takes as parameters a row r and a column that belong to an image. This function returns 1 if the pixel at the mentioned row and column are actually part of object i, and 0 otherwise. Mathemathically, this can be represented as:

$$I_i(r,c) = \begin{cases} 1, & \text{if } I(r,c) \in \text{object labeled 'i'} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where } r \in [0 \ldots Height-1] \text{ and } c \in [0 \ldots Width-1]$$

Figure 4.14 Formula for checking appartenence to object [17]

Using the I function, the area of an object i can be computed by scanning the image and summing the result of the I function for each row and column of the image. The formula is:

$$A_i = \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} I_i(r,c)$$

Figure 4.15 Area in image processing [17]

An example of the area can be seen in Chapter 2, figure 2.5, where the object and its area are colored in blue.

4.1.5. Center of mass

The center of mass is a unique position that can be associated with one or more objects or figures. It can also be considered as the average position with respect to the object. The center of mass is usually also a point that belongs to the figure, but it is not necessary. The point could very well be situated outside the object, such being the case for rings or concave irregular polygons. Usually, the center of mass uses a more complicated definition, but considering we are dealing with objects that have a uniform distribution of mass in two dimensions, we can use the following formula:

$$\bar{r}_i = \frac{1}{A_i} \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} r I_i(r,c)$$

$$\bar{c}_i = \frac{1}{A_i} \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} c I_i(r,c)$$

Figure 4.16 Formula for center of mass in image processing [17]

In image processing, by using the I function defined before at point 4.1.5 along with the area, we can compute the center of mass of an object i in the following manner: The row of the center of mass is computer by scanning the image for pixels that belong to object i, and summing up all row number of such pixels. The sum is then divided by the total area of the object, and the result is the row of the center of mass point. The same procedure is applied for finding the column of the center of mass point.

An example of the center of mass point can be seen in Chapter 2, figure 2.5, at the intersection of the two red lines.

4.1.6. Image warping

Image warping consists of transforming an image in such a way that it appears distorted. There are different types of image warping such as affine warping or perspective warping. As mentioned in point 4.1.2, an affine warping maintains parallelism, which is something that is desired for this thesis. This process is important as it allows us to warp the parts of interest from the constellation model, which are the stars, to some coordinates according to the pair of similar triangles found.

4.1.7. Affine Transformation

An affine transformation [12], in Euclidean geometry, represents a transformation of a figure in such a way that parallelism is maintained. All the transformations mentioned previously at 4.1.1: translation, scaling, rotation, reflection or any combinations between them are examples of an affine transformation.

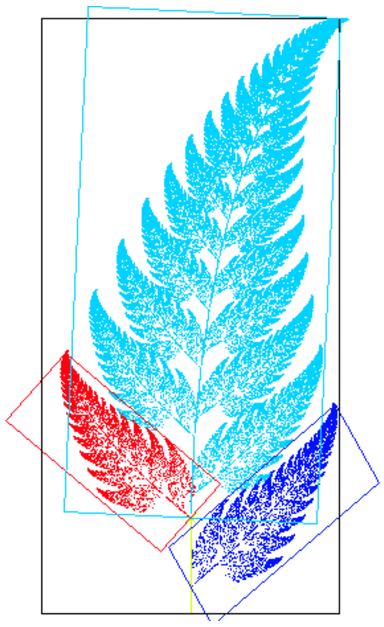


Figure 4.17 Affine transformation exemplified on Barnsley's fern [12]

Figure 4.17 shows a fractal figure, named Barnsley’s fern, that provides an easy way to observe affinity, as any leaf of the fern can be transformed into any other leaf with an affine transformation. For instance, the dark blue leaf could be transformed into the red leaf with a combination of reflection, translation and perhaps a small rotation.

The whole transformation process can be conveyed through a matrix multiplication and a vector addition. In two dimensions, which is what we use for image processing, the matrix A and vector B would look something like this:

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}_{2\times 2} \quad B = \begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix}_{2\times 1}$$

Figure 4.18 2D affine transformation matrix and vector [18]

4.1.8. Algorithms used

Some of the essential steps in the system are binarization and labeling, which can be obtained with different algorithms.

4.1.8.1. Binarization

A very simple and primitive algorithm to transform a grayscale image into a binary image consists of selecting a treshold value and assigning a new value of for each pixel of 0 or 1 based off a comparison of the pixel to the threshold. This threshold is fixed and selected before the actual process of conversion is started.

$$Dst(i,j) = \begin{cases} 0 & (black) \quad , \quad if \quad Src(i,j) < threshold \\ 255 & (white) \quad , \quad if \quad Src(i,j) \geq threshold \end{cases}$$

Figure 4.19 Image processing simple binarization [17]

An example of the fixed threshold binarization can be seen in the next figure, where the threshold value is fixed to 128:

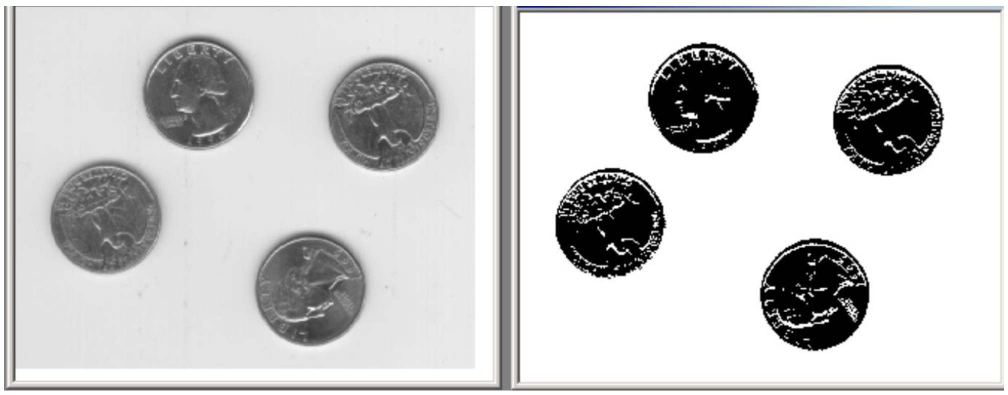


Figure 4.20 Fixed threshold binarization [17]

A more advanced method of grayscale image to binary image conversion would be having an adaptive threshold. A refined concept was introduced by S.D. Yanowitz and A.M. Bruckstein in their 1988 paper named „A New Method for Image Segmentation” [19]. Briefly put, the method uses information about edges and gray-levels in order to produce a threshold surface which then goes through a validation process.

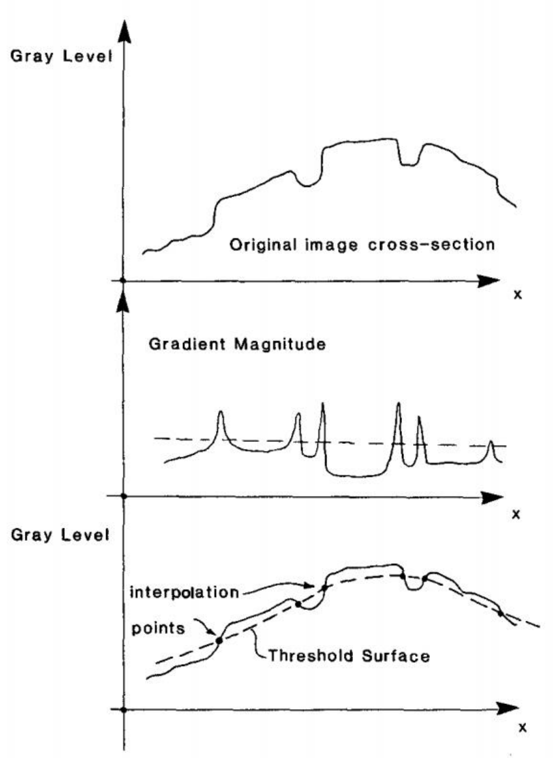


Figure 4.21 Obtaining the threshold surface [19]

As we can see in figure 4.21, the first part shows the original image cross-section. The second part shows the gradient magnitude image cross section and the third part shows the peaks of the gradient magnitude at interpolation values, determining the threshold surface. An example for this thresholding method can be seen in the next figure:

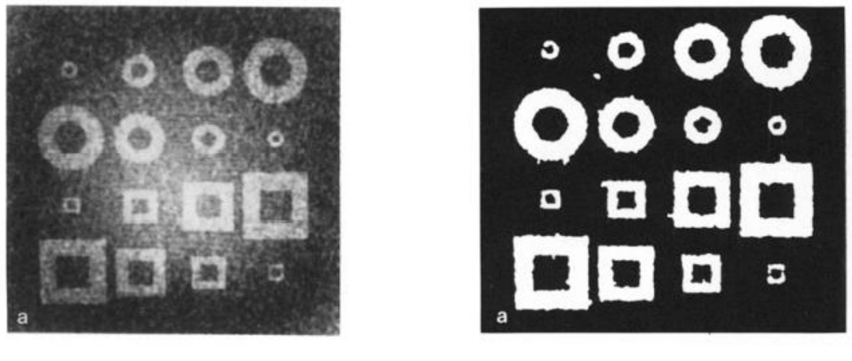


Figure 4.22 Adaptive threshold example [19]

It is important to note that the adaptive thresholding is valuable in cases where the background can be uneven or hard to distinguish due to a weak quality of illumination in the image. This is hardly the case for this thesis, as in astronomical images it can be clearly observed that the background's gray level is very close to 0. Taking this into consideration, and also the fact that the fixed threshold method is much simpler and faster, I considered it the optimal choice for this project. Adaptive thresholding remains however a viable option for more varied input images.

4.1.8.2. Labeling

Labeling refers to assigning a unique number to each object in the scene for identification purposes. This process can be also done through various algorithms, such as breadth first traversal connected component labeling or two-pass connected component labeling. Regardless of the algorithms, the neighbourhood connectivity is an important aspect to consider for this process. As the object is represented as a component made up from object-pixels that are connected between themselves. Neighborhood connectivity decides which neighbors qualify as properly connected to other pixels. Some popular examples in image processing are 4-neighborhood(a), 8-neighborhood(b) and previous neighbors(c):

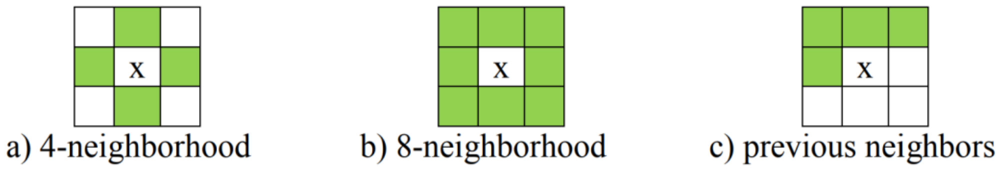


Figure 4.23 Examples of neighbourhoods [17]

The 4-neighborhood checks for neighbors only on adjacent pixels on the horizontal and vertical, the 8-neighborhood adds diagonals on top of the 4-neighborhood and the previous neighbors assumes the image is scanned from top-left to bottom-right, so it checks only for the positions the algorithm has already processed, which are left, top-left, top-middle and top-right.

The first algorithm for labeling relies on the breadth first search algorithm.

```
label = 0
labels = zeros(height, width)   //height x width matrix with 0
for i = 0:height-1
    for j = 0:width-1
          if img(i,j)==0 and labels(i,j)==0
                label++
                Q = queue()
                labels(i,j) = label
                Q.push( (i,j) )
                while Q not empty
                      q = Q.pop()
                      for each neighbor in N8(q)
                            if img(neighbor)==0 and labels(neighbor)==0
                                  labels(neighbor) = label
                                  Q.push( neighbor )
```

Figure 4.24 BFS Labeling pseudocode [17]

The input for the algorithm is a binary image, where objects have the value of 0(black) and the background has the value of 255(white). The label variable is the unique identifier assigned to each object. The lables variable is a matrix with the same size as the initial image, that is initialized with 0, meaning unlabeled. The image is scanned from top-left to bottom-right, pixel by pixel, and whenever one object pixel that is unlabeled according to the labels matrix, this means that a new object has been found. The label variable is incremented as to assure the uniqueness of the id, and then a breath-first search is issued from this new pixel found with the connectivity discussed previously. The breath first neighborhood search propagates from neighbor to neighbor, and all the neighbours are assigned with the same label as the first new pixel belonging to the object found and marked as such in the labels matrix. After the queue is emptied, meaning the object is completely labeled, the scanning process resumes in order to find the next object pixel that is unlabeled. This process repeats until the end of the image is reached, meaning that the labeling process is complete.

The second example for connected component labeling is the two-pass algorithm with equivalence classes. The first pass scans the image and labels object pixels using the connectivity mentioned previously, in such a way that if the current pixel has any neighbors with a lower label number, that label number is assigned to it and a connection between the mentioned labels is added in a graph as to say that the labels are equivalent; otherwise, a new label is attributed to the current object pixel. After this first pass, each object has an initial label. However, the graph built indicated which are the equivalent labels. Processing this graph results in relabeling all the labels equivalent to label X to X; the next unequivalent label to X gets relabeled to X+1, and all of it's equivalent labels are relabeled to X+1. This process repeats until all labels are processed. A second and final pass over the image is needed in order to update the initial labels assigned in the first pass.

4.1.9. Morphological operations

Morphological operations are used in image processing on binary images in order to modify objects in certain ways. The two main morphological operations are dilation and erosion. The operations use a structuring element which resembles the neighborhood connectivity in figure 4.23, though in a slightly different way. The structuring element can have any form and the next figure shows some of the most common:

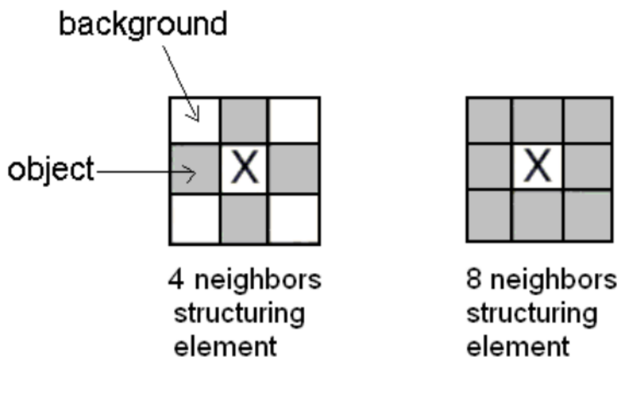

background
object
X
X
4 neighbors structuring element
8 neighbors structuring element

Figure 4.25 Examples of structuring elements[17]

4.1.9.1. Dilation

The dilation process works in the following manner: the image is scanned, typically from top-left to bottom-right, laying the structuring element with the origin on the current scanned pixel. If the pixel belongs to an object, then the neighbors defined by the structuring element also become object-pixels. Thus, the dilation process is used to enlarge objects or to fill objects that contain small holes.

The notation for the dilation process of an image A with a structuring element B is:

$$A \oplus B$$

Figure 4.26 Dilation notation [17]

A visual representation of the dilation process can be seen in the next figure:

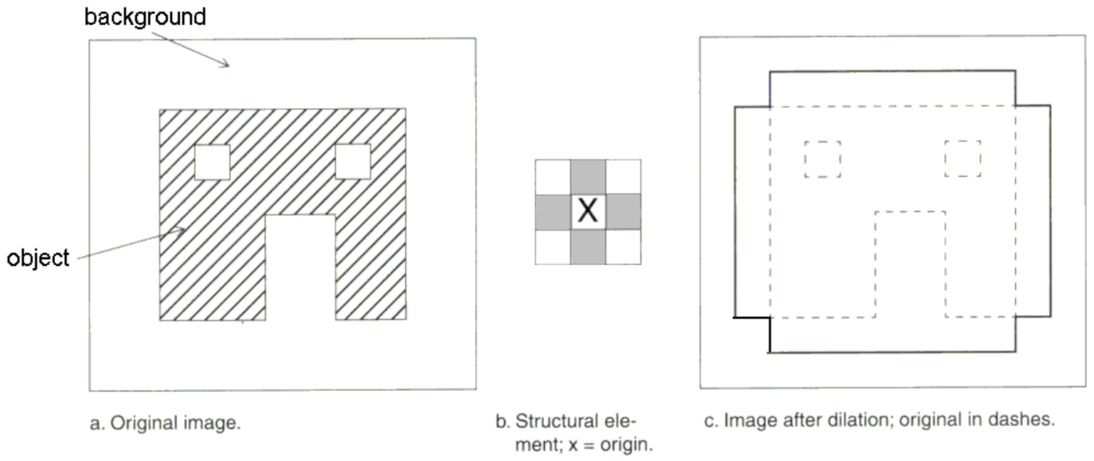

background
object
X
a. Original image.
b. Structural element; x = origin.
c. Image after dilation; original in dashes.

Figure 4.27 Dilation representation [17]

An example of the dilation process can be seen in the next figure, where the left image is the original and the right image is the result after dilation:

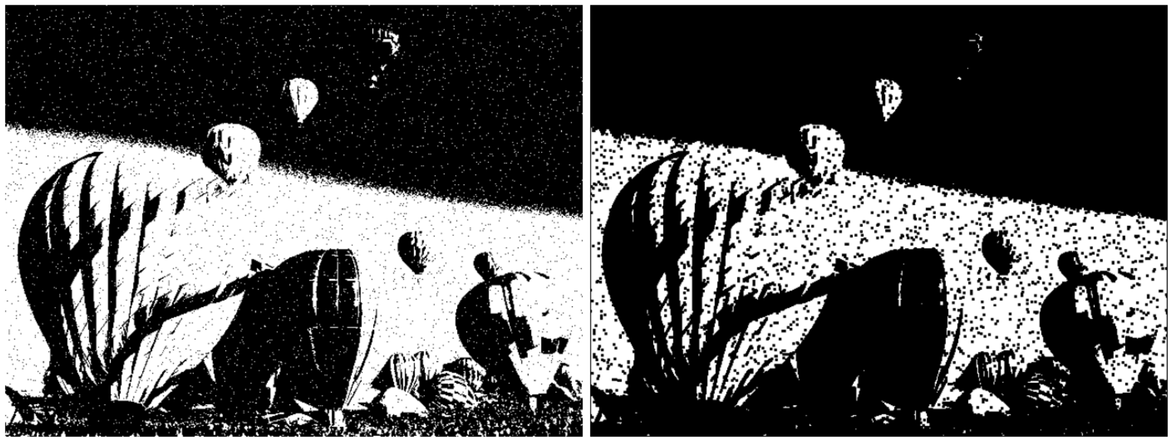


Figure 4.28 Dilation example [17]

4.1.9.2. Erosion

The erosion process is the second main morphological operation. It works in the following way: the image is scanned, typically from top-left to bottom-right, laying the structuring element with the origin on the current scanned pixel. If the current pixel belongs to an object, and at least one neighbor defined by the structuring element is a background-pixel, then the current object-pixel is transformed into an background-pixel. Erosion is used to shrink objects or to separate objects connected by thin lines or structures.

The notation for the erosion process of an image A with the structuring element B is:

$$A \Theta B$$

Figure 4.29 Erosion notation [17]

A visual representation of the erosion process can be seen in the next figure:

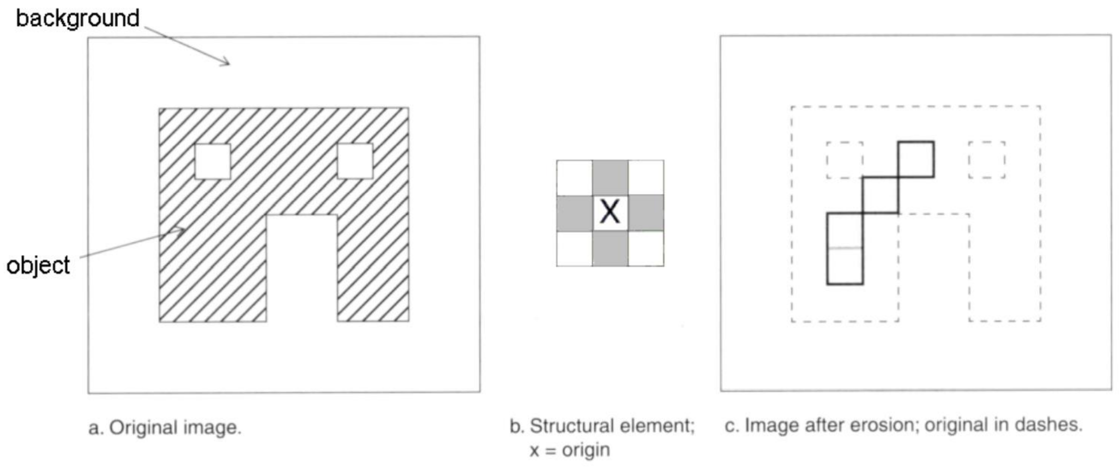


Figure 4.30 Erosion representation [17]

An example of the erosion process can be seen in the next figure, where the left image is the original and the right image is the result after erosion:

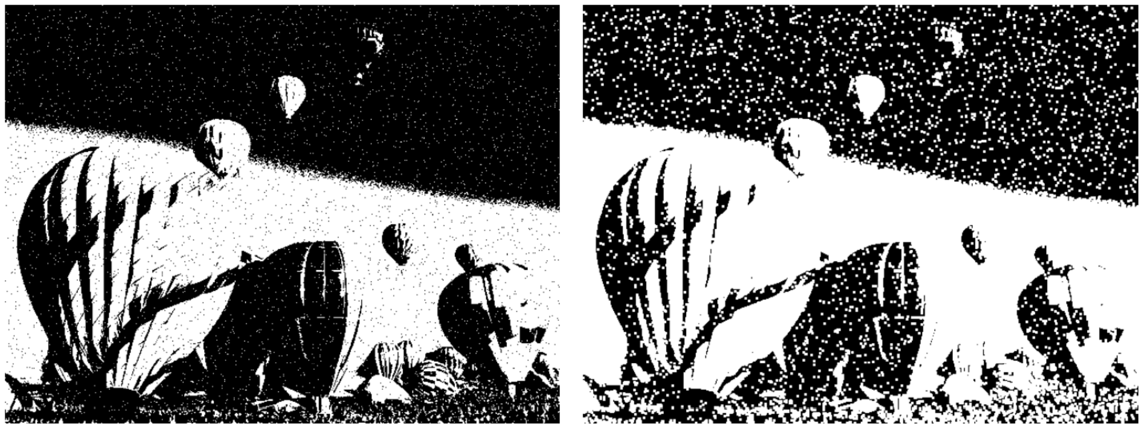


Figure 4.31 Erosion example [17]

4.1.9.3. Opening and Closing

Opening and closing are morphological operations composed from the main operation described previously, such that: the opening process consists of essentially applying dilation operation followed by an erosion operation, while the closing process is the opposite: applying an erosion operation followed by a dilation process.

Opening is therefore used in order to eliminate objects smaller than the structuring element and closing is used for filling gaps that can be covered by the structuring element.

The notation for the opening process of an image A with the structuring element B is:

$$A \circ B = (A \Theta B) \oplus B$$

Figure 4.32 Opening notation [17]

The notation for the closing process of an image A with the structuring element B is:

$$A \bullet B = (A \oplus B) \Theta B$$

Figure 4.33 Closing notation [17]

A visual representation of the opening process can be seen in the next figure:

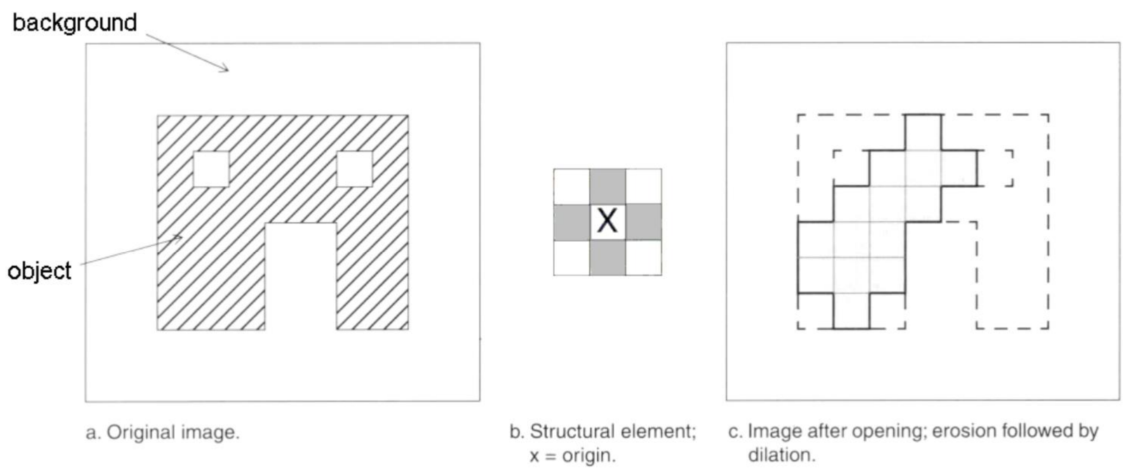

background
object
X
a. Original image.
b. Structural element; x = origin.
c. Image after opening; erosion followed by dilation.

Figure 4.34 Opening representation [17]

A visual representation of the closing process can be seen in the next figure:

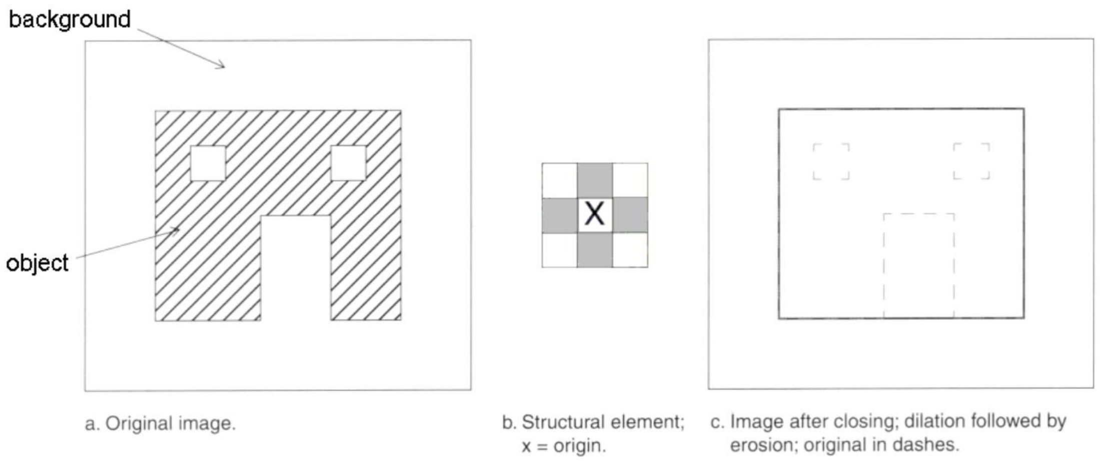

background
object
X
a. Original image.
b. Structural element; x = origin.
c. Image after closing; dilation followed by erosion; original in dashes.

Figure 4.35 Closing representation [17]

Examples for the opening and closing process can be seen in the next figure respectively from left to right. The input image is the left image from figures 4.31 and 4.28.

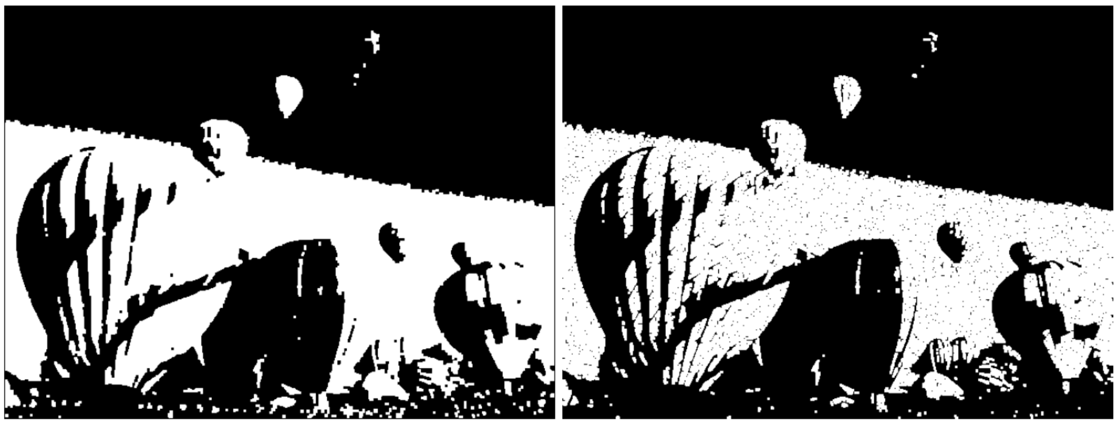


Figure 4.36 Opening(left) and closing(right) example

Chapter 5. Detailed Design and Implementation

The whole project has two main parts that needed to be designed separately and then implemented in order to allow the second part to use the results generated by the first part. These are:

- Constellations database
- Detection system

5.1. Constellations database

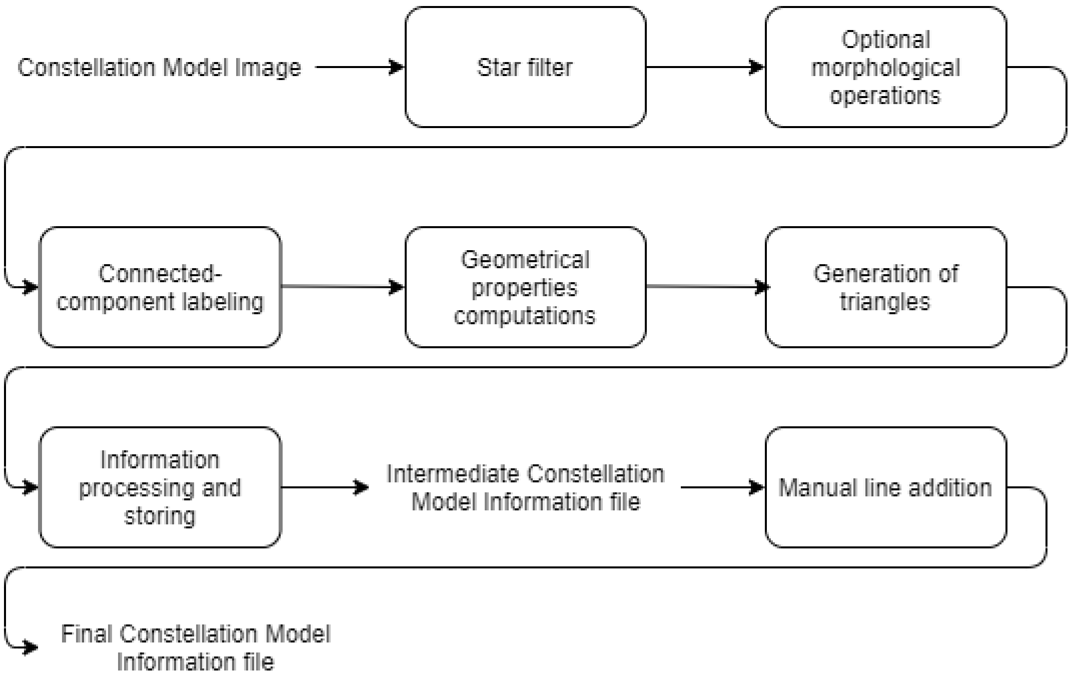

Constellation Model Image
Star filter
Optional morphological operations
Connected-component labeling
Geometrical properties computations
Generation of triangles
Information processing and storing
Intermediate Constellation Model Information file
Manual line addition
Final Constellation Model Information file

Figure 5.1 Diagram of the constellation database building process

As we can see from the diagram above, the database that holds information about each constellation is not implemented as a relation database, but as a text file. Furthermore, each stage will be discussed in detail.

5.1.1. Constellation Model Image

The image models for the constellations were taken from the astronomyonline.org website[20]. I chose these models because they are simple to process due to the distinct color of the stars and the fact that paper [8] also uses the same website for their constellation database building. I considered this fact to be somewhat useful for comparison between my results and the results of [8].

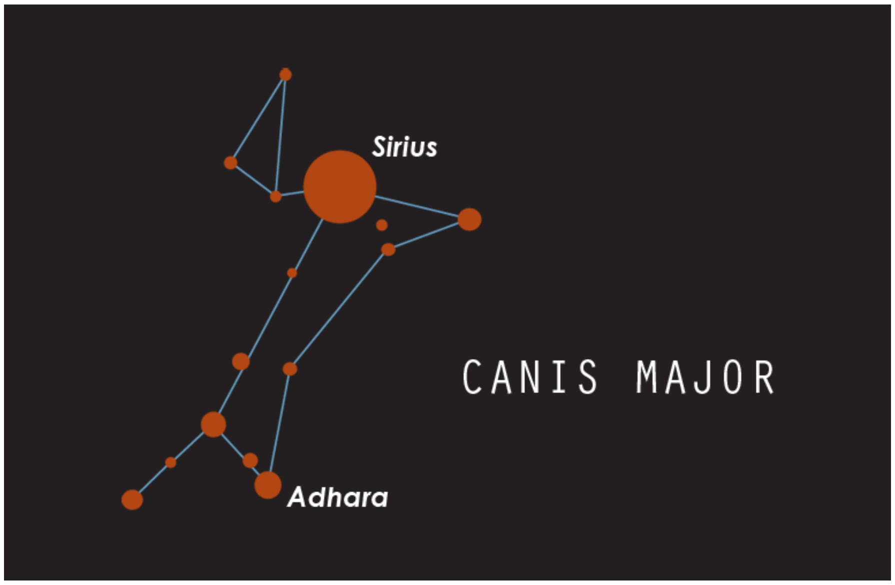
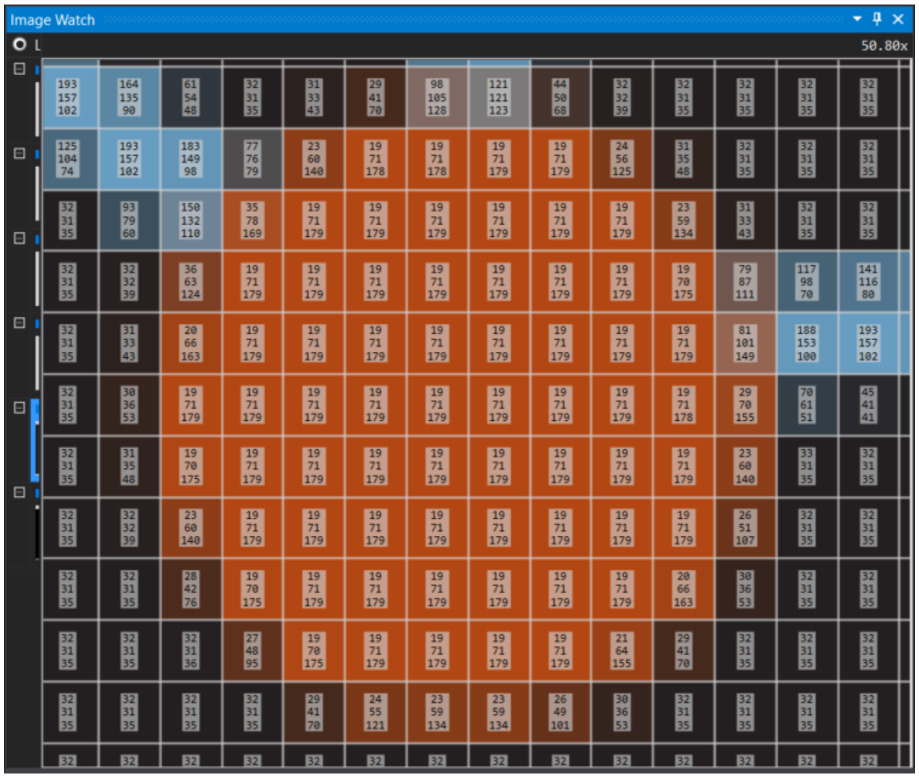


Figure 5.2 Model for constellation Canis Major [20]

5.1.2. *Star filter*

The previous image is then filtered in order to preserve the stars in the image. As mentioned in point 5.1.1, the stars in the image model contain a very distinct RGB color from the rest of the colors in the image.



193 157 102	164 135 90	61 54 48	32 31 35	31 33 43	29 41 70	98 105 128	121 121 123	44 50 68	32 32 39	32 31 35	32 31 35	32 31 35	32 31 35
125 104 74	193 157 102	183 149 98	77 76 79	23 60 140	19 71 178	19 71 178	19 71 179	19 71 179	24 56 125	31 35 48	32 31 35	32 31 35	32 31 35
32 31 35	93 79 60	150 132 110	35 78 169	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	23 59 134	31 33 43	32 31 35	32 31 35
32 31 35	32 32 39	36 63 124	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 70 175	79 87 111	117 98 70	141 116 80
32 31 35	31 33 43	20 66 163	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	81 101 149	188 153 100	193 157 102
32 31 35	30 36 53	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 178	29 70 155	70 61 51	45 41 41
32 31 35	31 35 48	19 70 175	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	23 60 140	33 31 35	32 31 35
32 31 35	32 32 39	23 60 140	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	26 51 107	32 31 35	32 31 35
32 31 35	32 31 35	28 42 76	19 70 175	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	19 71 179	20 66 163	30 36 53	32 31 35	32 31 35
32 31 35	32 31 35	32 31 36	27 48 95	19 70 175	19 71 179	19 71 179	19 71 179	19 71 179	21 64 155	29 41 70	32 31 35	32 31 35	32 31 35
32 31 35	32 31 35	32 31 35	32 31 35	29 41 70	24 55 121	23 59 134	23 59 134	26 49 101	30 36 53	32 31 35	32 31 35	32 31 35	32 31 35
32	32	32	32	32	32	32	32	32	32	32	32	32	32

Figure 5.3 Zoomed image of a star in constellation model

As it can be seen in the previous figure, the main RGB color for the stars is composed with a value of 19 for the blue component, 71 for the green component and 179 for the red component. The order of BGR is given by the implementation of the Vec3b type in OpenCV.

With this information, I've created a simple function that filters the image for the previously mentioned values, allowing some variation for each color in order to also capture the pixels on the edge of the star, which are increase in darkness with the distance from the center of the star. The function also converts the image from color to binary by setting the star-pixels to the value of 255, or white, and all other pixels to the value of 0, or black.

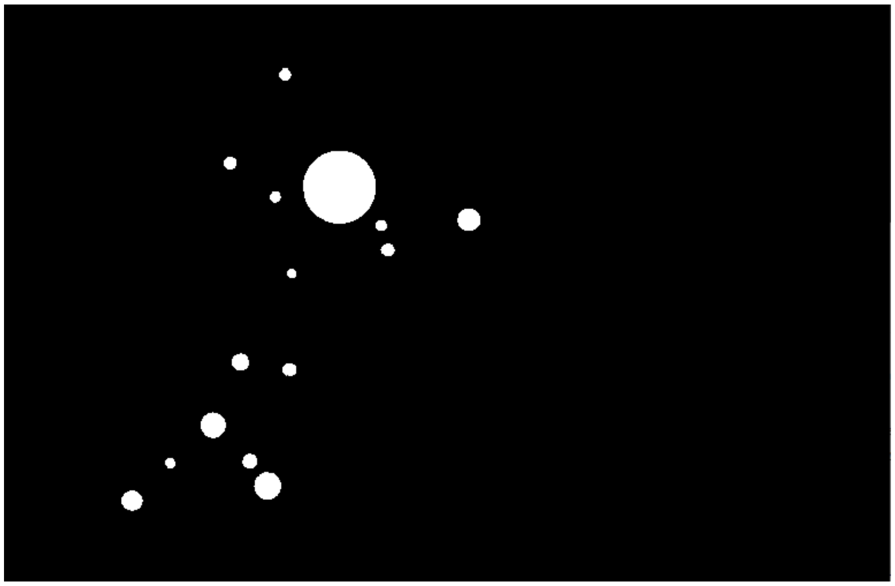


Figure 5.4 Filter stars result image

5.1.3. *Optional morphological operations*

Figure 5.2 illustrates a „perfect" example of a constellation, in the sense that all stars are clearly defined and separated from eachother. One troublesome example of a constellation would be Corvus in figure 5.5, where the two most top-left stars are extremely close together, and from this cause the labeling algorithm would consider them as one star. In order to avoid this situation, an erosion is a simple fix as it does not influence any upcoming process in the database building. This is true as we are interested later on to transform stars from objects to points, therefore the overall area is not a concern as long as the center of mass remains the same.

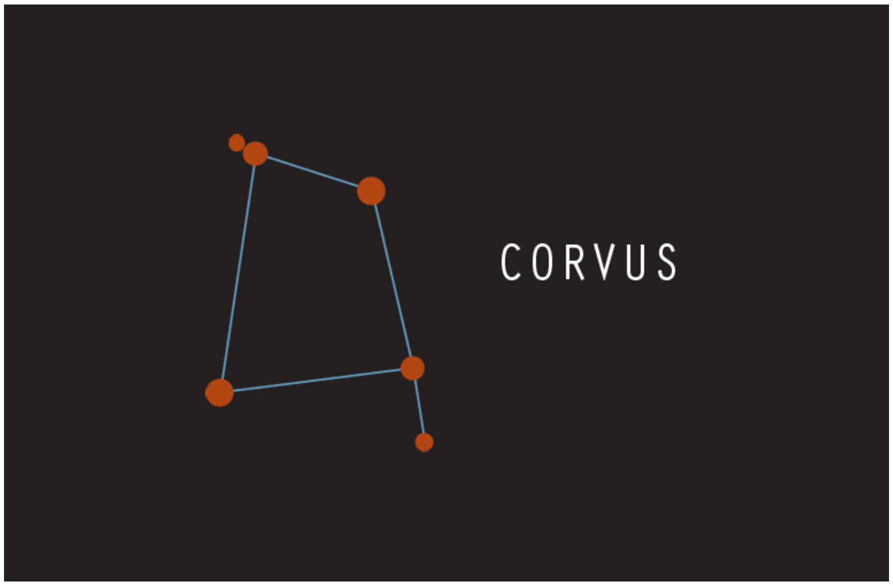


Figure 5.5 Model for constellation Corvus [20]

A close-up on the stars discussed previously, before and after the process of erosion on the left and right respectively:

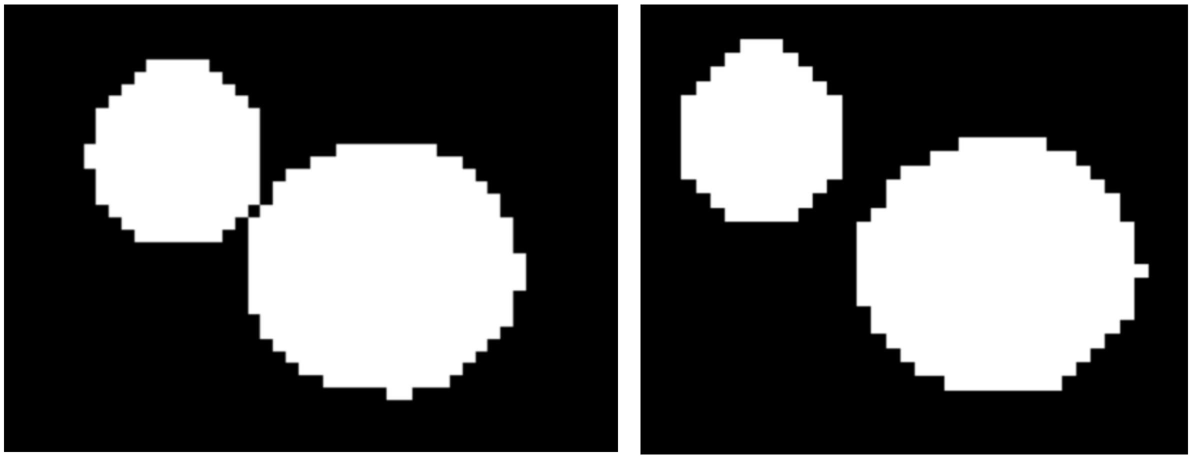


Figure 5.6 Erosion used for close stars

To represent a different types of stars than normal, in this case the eruptive variable star Navi, the models can have representations such as:

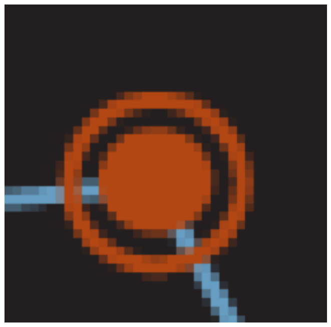


Figure 5.7 Eruptive variable star Navi in Cassiopeia [20]

To prevent the detection algorithm to consider the star in figure 5.7 as two objects, we can use a closing operation. However, as we can see in the left of the next figure, only one dilation followed by one erosion is not enough. To provide a correct result for this star, a closing operation consisting of two dilations followed by two erosions is necessary

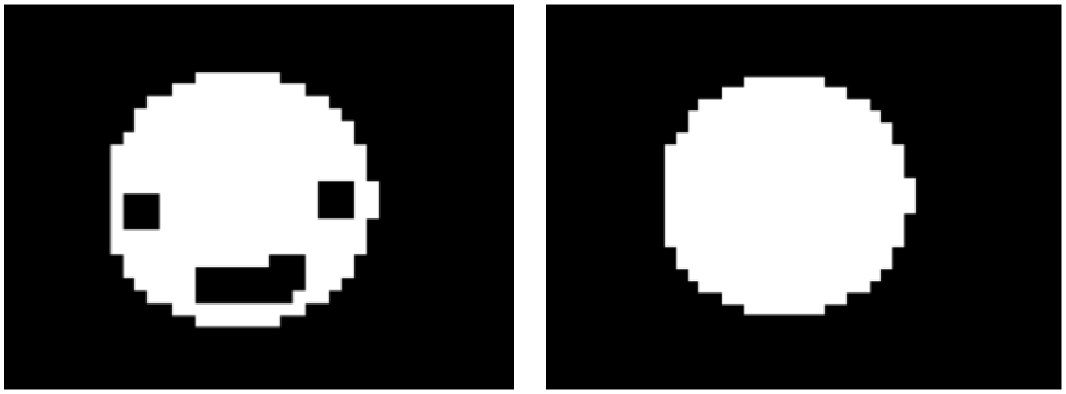


Figure 5.8 Closing with 1 dilation and 1 erosion(left), closing with 2 dilations and 2 erosions(right)

5.1.4. Connected-component labeling

For the labelling process, I’ve implemented the BFS method described at point 4.1.8.2 and for which the pseudocode can be seen in figure 4.24.

Labeling the image from figure 5.4:

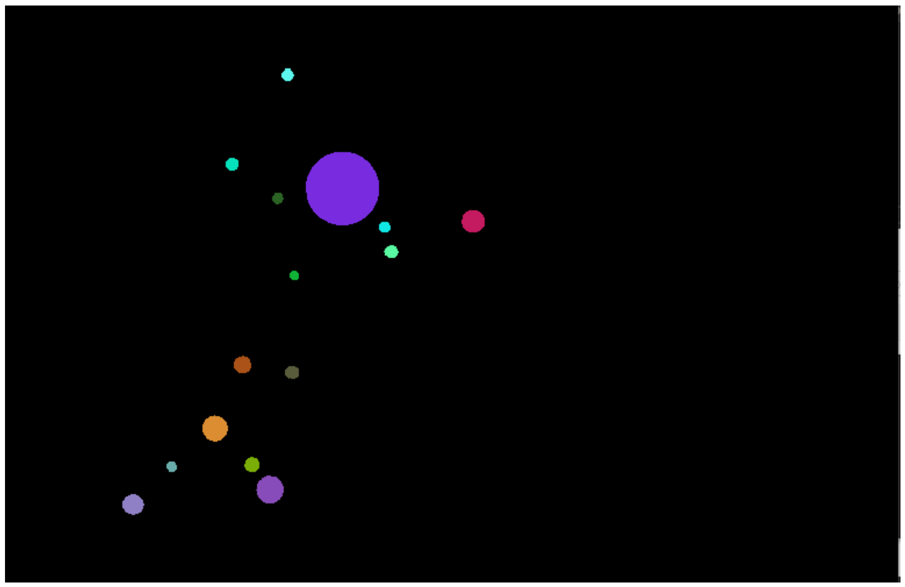


Figure 5.9 Labeling constellation Canis Major

The differences made by the erosion depicted in figure 5.6 can be observed in the next figure:

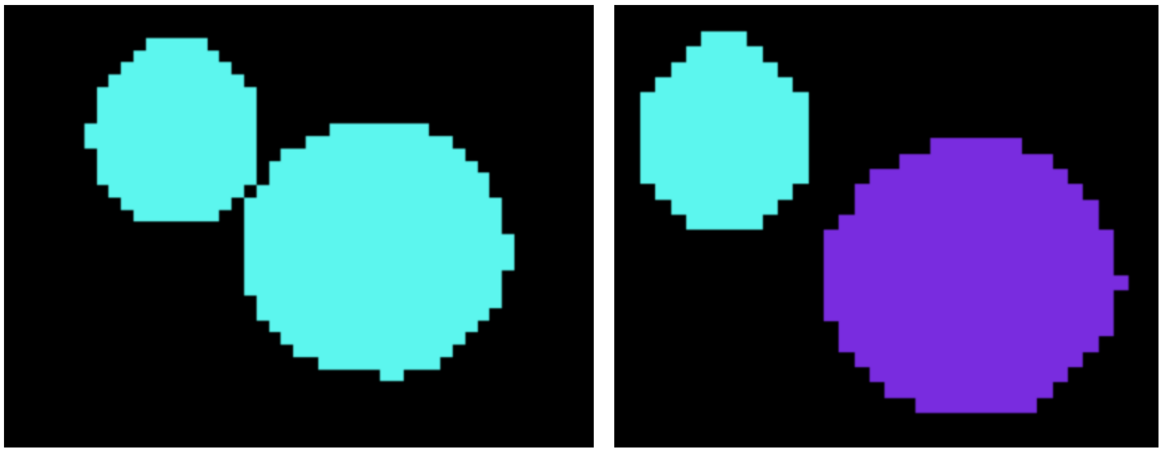


Figure 5.10 Labeling stars from Corvus before and after erosion in figure 5.6

The differences made by the closing process depicted in figure 5.7 and 5.8 can be observed in the next figure:

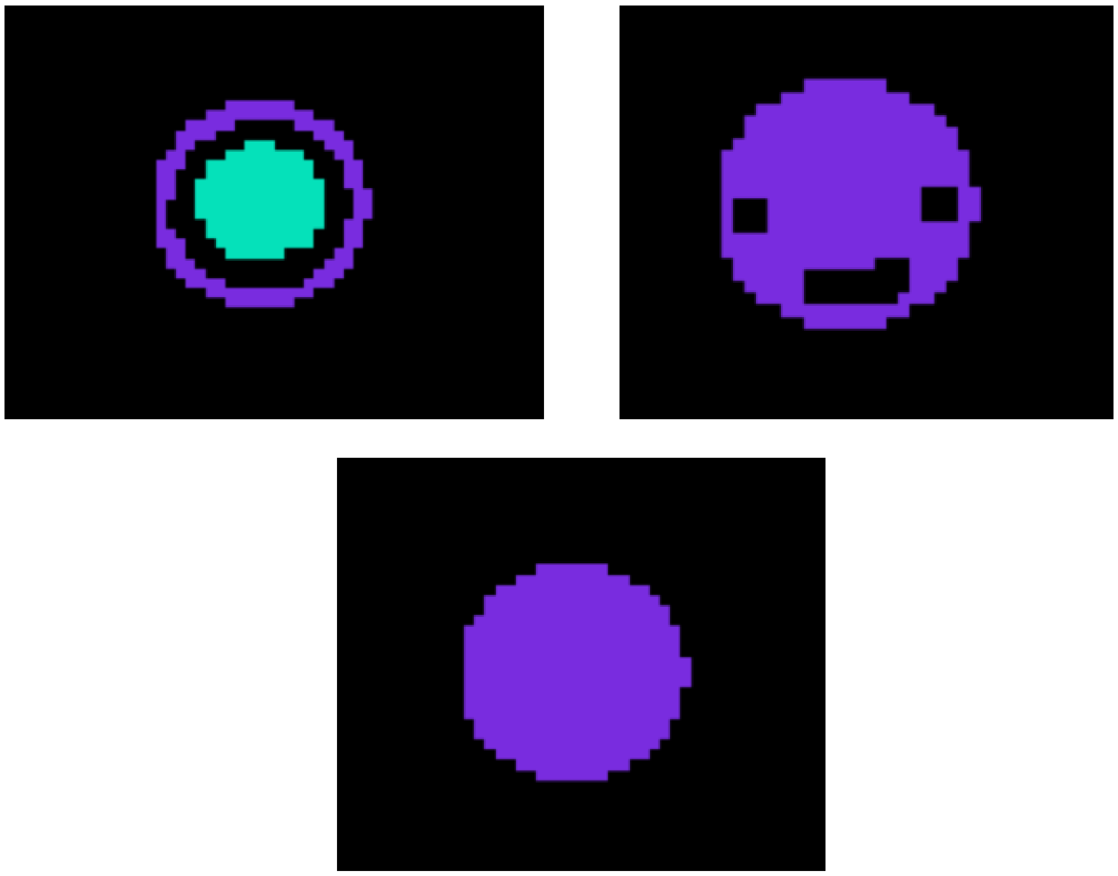


Figure 5.11 Labeling variable star Navi from Cassiopeia in figures 5.7 and 5.8: top-left for initial image, top-right for closing with 1 dilation and 1 erosion and bottom for closing with 2 dilations and 2 erosions

5.1.5. Geometrical properties computations

After each star is succesfully labeled, the purpose of this stage is to reduce them to single points, which are their respective centers of mass. The computations for the area was discussed at point 4.1.4 and was implemented as such. The computation of the centers of mass was discussed at point 4.1.5 and was implemented as such, with the area computed previously.

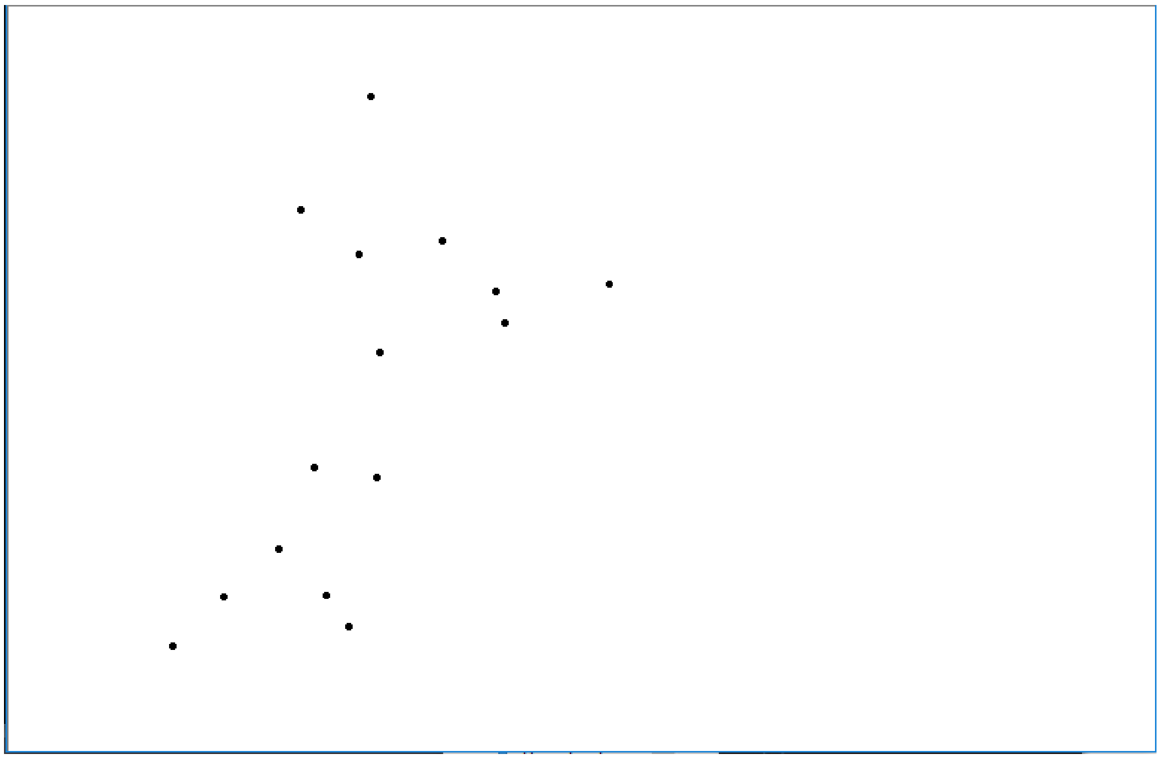
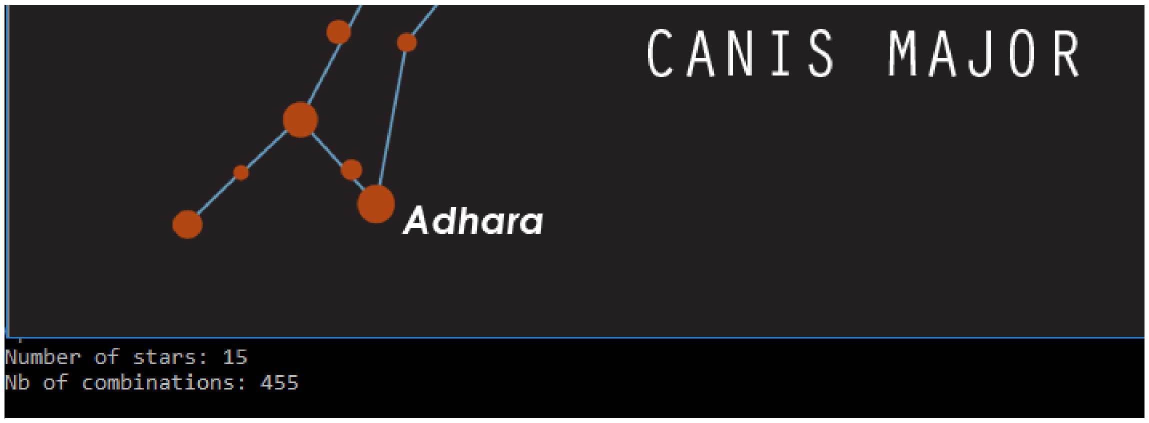


Figure 5.12 Centers of mass of each star from constellation Canis Major (image altered for visibility)

The center of mass computation also creates a separate integer matrix of areas, such that if there is a center of mass point at row X and column Y that represents star Z, the areas matrix in row X and column Y holds the value for the area of star Z.

5.1.6. Generation of triangles

After computing all the points that represent all the stars, triangles are generated for each combination of 3 points. So, for a constellation such as Canis Major with a number of 15 stars, the total number of triangles generated is combinations of 15 taken 3, which equals 455 triangles.



```
Number of stars: 15
Nb of combinations: 455
```

Figure 5.13 Triangle generation for constellation Canis Major

5.1.7. Information processing and storing

With the triangles generated, we now need to compute the distances between points. The distance metric chosen is the Euclidean metric, discussed at point 4.1.3.3. The really important points in this section is that all distances should be normalized to the minimum distance, and all points should maintain the same relative position in their vector.

E.g. When we find that two triangles ABC and DEF are similar, we should also be able to tell that point A corresponds to point D, B corresponds to point E and C corresponds to point F. However, another scenario would be that point A corresponds to point E, point B corresponds to point F and point C corresponds to point D. This can also be a correct result, but the correspondence between points needs to be established.

I chose to implement this points in the following way: have a list of distances and a list of points that belongs to each triangle. A distance is defined by two points, p1 and p2 and has a value. Therefore:

- Compute distances for each side of the triangle
- Find the minimum distance out of the three distances
- Minimize all distances with the minimum distance
- Sort the distances ascending
- Check and restore consistency and continuity in distances endpoints

The end goal is to be able to match corresponding points in triangles by their distances starting points, such that for triangle ABC with sides of increasing length being BC=1, AC=2 and BA=3 the distances can be seen as such: d1(B,C)=1, d2(C,A)=2, d3(A,B)=3.

The consistency and continuity can be seen as the points match in the following way:

- d1's starting point is the same with d3's end point
- d2's starting point is the same as d1's end point
- d3's starting point is the same as d2's end point

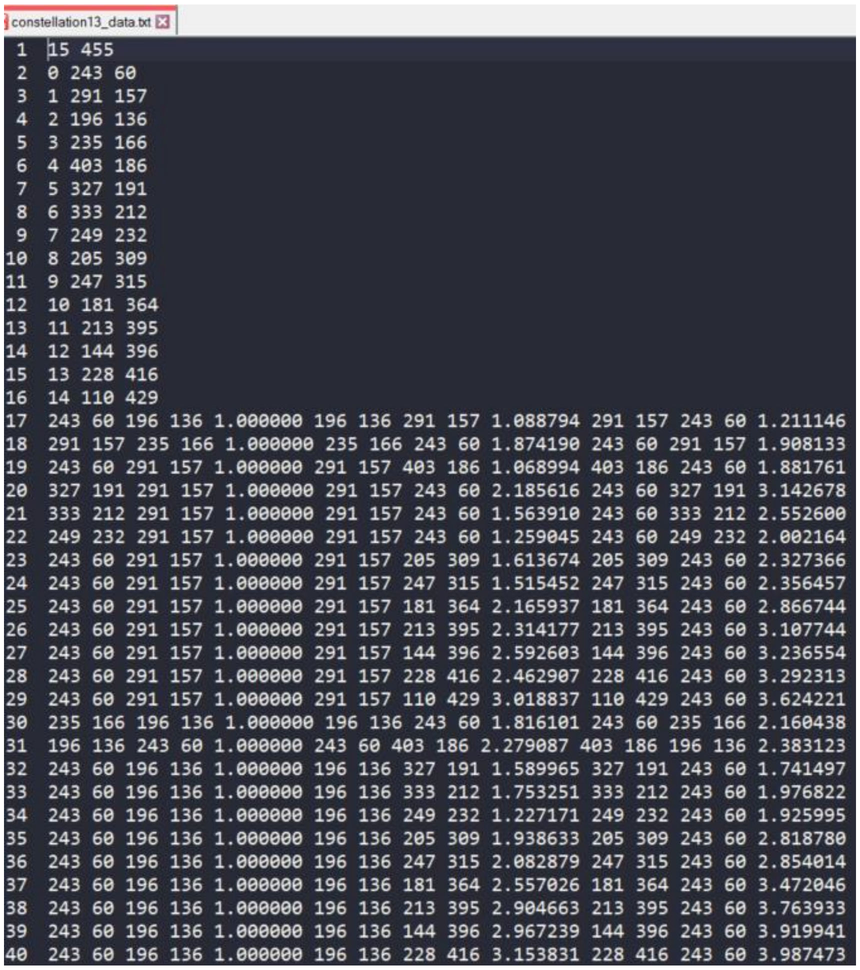
By maintaining this structure, we know that each of the distances starting points are different and the distances starting points of one triangle correspond to the distances starting point of another triangle.

To store this information, the easiest option is a plain text file. Security is not a concern for this application, as the user does not any account to use the application. Relational databases would also be a needless complication because the database would either have to be ran locally by the user, which would tremendously decrease usability of the whole application, or the database would have to be hosted online, meaning the application would need connection to the internet, which is something that I've established as a core requirement of the application.

5.1.8. Intermediate constellation model information file

Considering point 5.1.7, an example for an intermediate constellation model information file for constellation Canis Major can be seen in the next figure. It contains on the first line the number of points, which is 15, and the number of triangles generated from those points, which is 455, separated by a whitespace. On the following 15 lines we have the coordinates of the point and then the 455 lines representing each triangle. A triangle line is of the form:

distance1_point1_coordinates distance1_point2_coordinates distance1_value distance2_point1_coordinates distance2_point2_coordinates distance2_value distance3_point1_coordinates distance3_point2_coordinates distance3_value



```
15 455
0 243 60
1 291 157
2 196 136
3 235 166
4 403 186
5 327 191
6 333 212
7 249 232
8 205 309
9 247 315
10 181 364
11 213 395
12 144 396
13 228 416
14 110 429
243 60 196 136 1.000000 196 136 291 157 1.088794 291 157 243 60 1.211146
291 157 235 166 1.000000 235 166 243 60 1.874190 243 60 291 157 1.908133
243 60 291 157 1.000000 291 157 403 186 1.068994 403 186 243 60 1.881761
327 191 291 157 1.000000 291 157 243 60 2.185616 243 60 327 191 3.142678
333 212 291 157 1.000000 291 157 243 60 1.563910 243 60 333 212 2.552600
249 232 291 157 1.000000 291 157 243 60 1.259045 243 60 249 232 2.002164
243 60 291 157 1.000000 291 157 205 309 1.613674 205 309 243 60 2.327366
243 60 291 157 1.000000 291 157 247 315 1.515452 247 315 243 60 2.356457
243 60 291 157 1.000000 291 157 181 364 2.165937 181 364 243 60 2.866744
243 60 291 157 1.000000 291 157 213 395 2.314177 213 395 243 60 3.107744
243 60 291 157 1.000000 291 157 144 396 2.592603 144 396 243 60 3.236554
243 60 291 157 1.000000 291 157 228 416 2.462907 228 416 243 60 3.292313
243 60 291 157 1.000000 291 157 110 429 3.018837 110 429 243 60 3.624221
235 166 196 136 1.000000 196 136 243 60 1.816101 243 60 235 166 2.160438
196 136 243 60 1.000000 243 60 403 186 2.279087 403 186 196 136 2.383123
243 60 196 136 1.000000 196 136 327 191 1.589965 327 191 243 60 1.741497
243 60 196 136 1.000000 196 136 333 212 1.753251 333 212 243 60 1.976822
243 60 196 136 1.000000 196 136 249 232 1.227171 249 232 243 60 1.925995
243 60 196 136 1.000000 196 136 205 309 1.938633 205 309 243 60 2.818780
243 60 196 136 1.000000 196 136 247 315 2.082879 247 315 243 60 2.854014
243 60 196 136 1.000000 196 136 181 364 2.557026 181 364 243 60 3.472046
243 60 196 136 1.000000 196 136 213 395 2.904663 213 395 243 60 3.763933
243 60 196 136 1.000000 196 136 144 396 2.967239 144 396 243 60 3.919941
243 60 196 136 1.000000 196 136 228 416 3.153831 228 416 243 60 3.987473
```

Figure 5.14 Snippet of intermediate processing file for constellation Canis Major

5.1.9. Manual line addition

The easiest option to record the lines was by manually modifying the processing file of each constellation. Knowing that the labeling algorithm uses breath first search and scans the image from top-left to bottom-right, I did a quick sketch on each constellation, by drawing the indexes of each star just by visually inspecting the image. Then, I modified the processing file to add the name of the constellation, the number of lines and the indexes of the stars between which the lines exist.

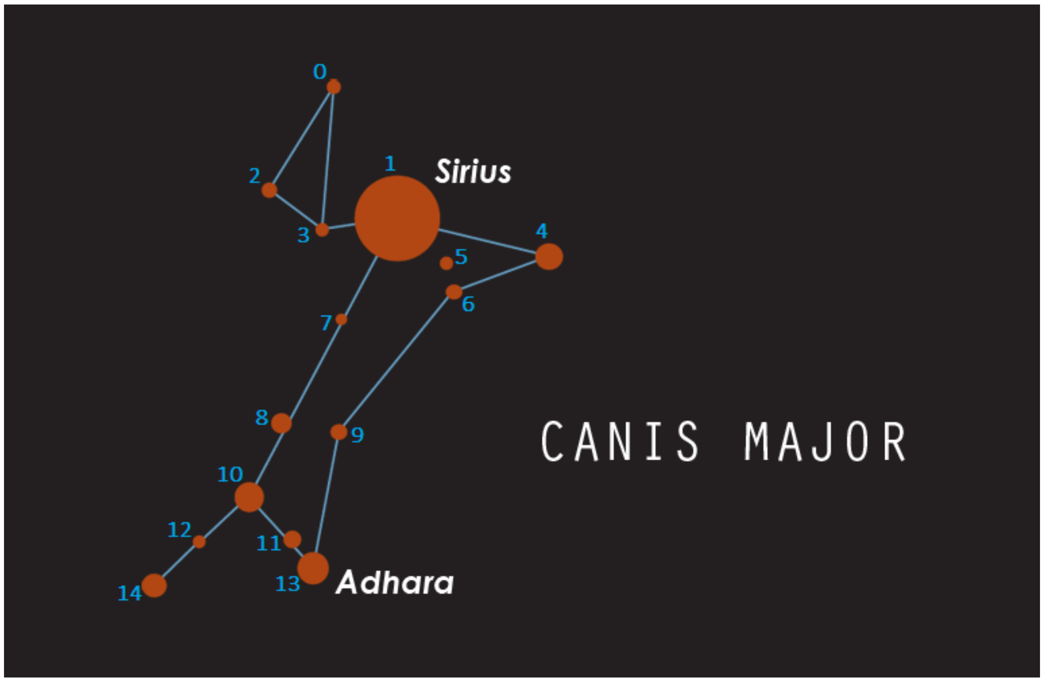
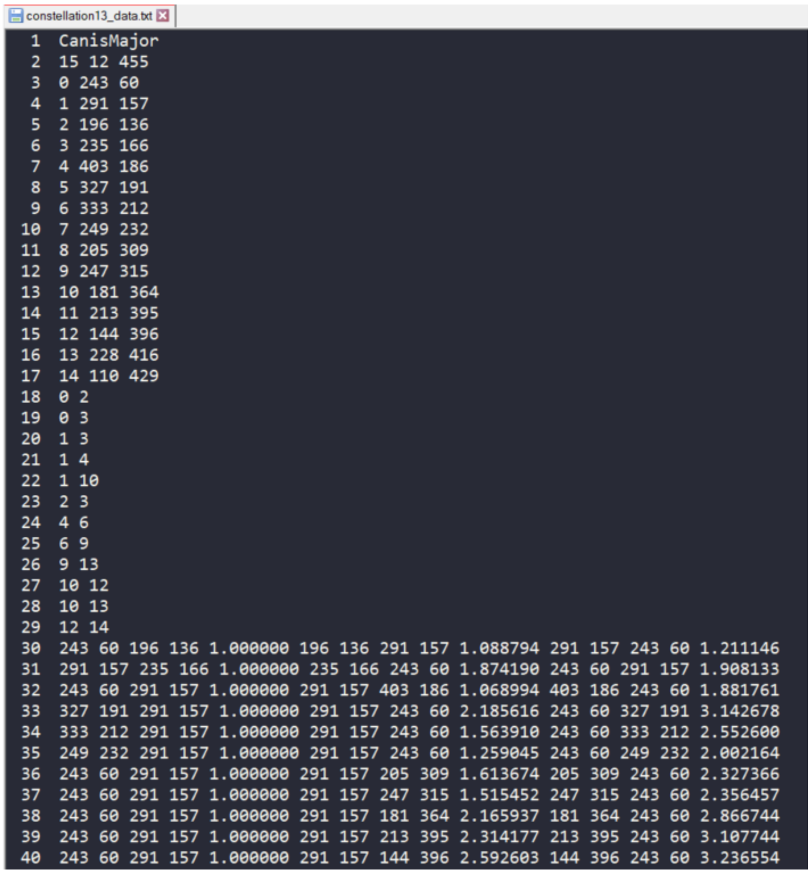


Figure 5.15 Visually indexing stars on constellation Canis Major

5.1.10. Final constellation model information file

After adding the name of the constellation and the lines between stars accoring to point 5.1.9, the structure of the file is as follows, with the first 2 bullet points representing each a line in the processing file:

- constellation_name
- nb_of_points nb_of_lines_between_stars nb_of_triangles
- nb_of_points lines with a point's index and coordinates, of the form „index x_coordinate y_coordinate"
- nb_of_lines_between_lines lines of the form „index0 index1", index0 and index1 indicating between which stars a line should be drawn
- nb_of_triangles lines of the form described at point 5.1.8



```
CanisMajor
15 12 455
0 243 60
1 291 157
2 196 136
3 235 166
4 403 186
5 327 191
6 333 212
7 249 232
8 205 309
9 247 315
10 181 364
11 213 395
12 144 396
13 228 416
14 110 429
0 2
0 3
1 3
1 4
1 10
2 3
4 6
6 9
9 13
10 12
10 13
12 14
243 60 196 136 1.000000 196 136 291 157 1.088794 291 157 243 60 1.211146
291 157 235 166 1.000000 235 166 243 60 1.874190 243 60 291 157 1.908133
243 60 291 157 1.000000 291 157 403 186 1.068994 403 186 243 60 1.881761
327 191 291 157 1.000000 291 157 243 60 2.185616 243 60 327 191 3.142678
333 212 291 157 1.000000 291 157 243 60 1.563910 243 60 333 212 2.552600
249 232 291 157 1.000000 291 157 243 60 1.259045 243 60 249 232 2.002164
243 60 291 157 1.000000 291 157 205 309 1.613674 205 309 243 60 2.327366
243 60 291 157 1.000000 291 157 247 315 1.515452 247 315 243 60 2.356457
243 60 291 157 1.000000 291 157 181 364 2.165937 181 364 243 60 2.866744
243 60 291 157 1.000000 291 157 213 395 2.314177 213 395 243 60 3.107744
243 60 291 157 1.000000 291 157 144 396 2.592603 144 396 243 60 3.236554
```

Figure 5.16 Final processing file of constellation Canis Major

5.2. Detection system

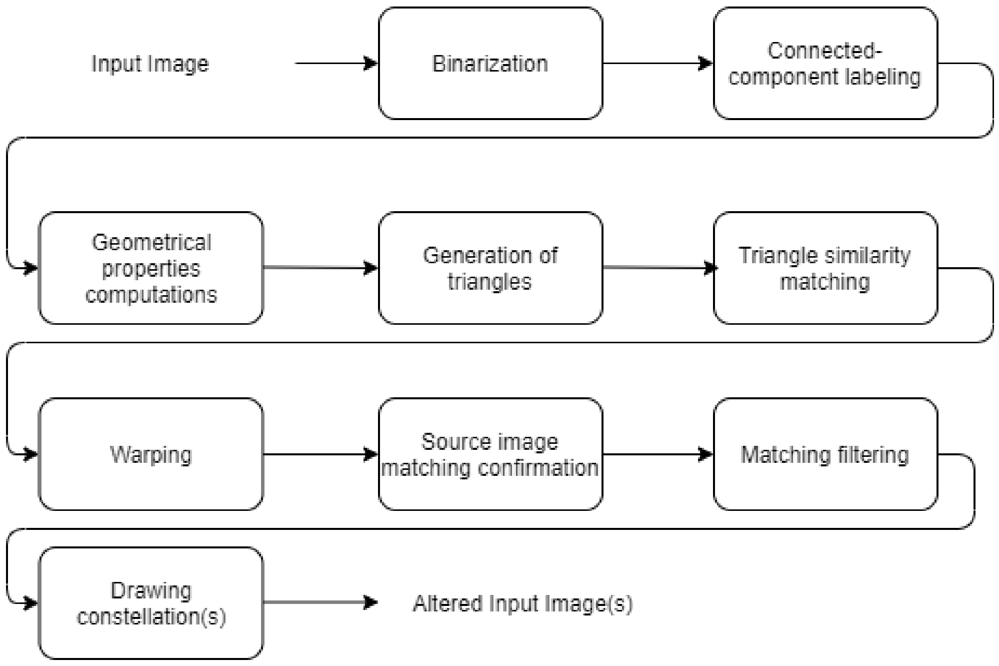

Input Image
Binarization
Connected-component labeling
Geometrical properties computations
Generation of triangles
Triangle similarity matching
Warping
Source image matching confirmation
Matching filtering
Drawing constellation(s)
Altered Input Image(s)

Figure 5.17 Detection system diagram

Each stage of the detection system shall be discussed in more detail in the next subsections.

5.2.1. Input Image

The input images used in development and in testing are snippets from the Stellarium Astronomy Software[21], which is a planetarium that renders a realistic sky. It has many features that allow it for easy manipulation and use. I was unable to find a data set of astronomical images that were also labeled, so I decided to build my own data set of sorts, by capturing photos in Stellarium. The pictures differ by orientation, amount of zoom, location on globe and time. There are 10 pictures per constellation, so for 89* constellations we have a total of 890 images. *There are 89 constellations as the Serpens constellation is divided into Serpens Cauda and Serpens Caput.

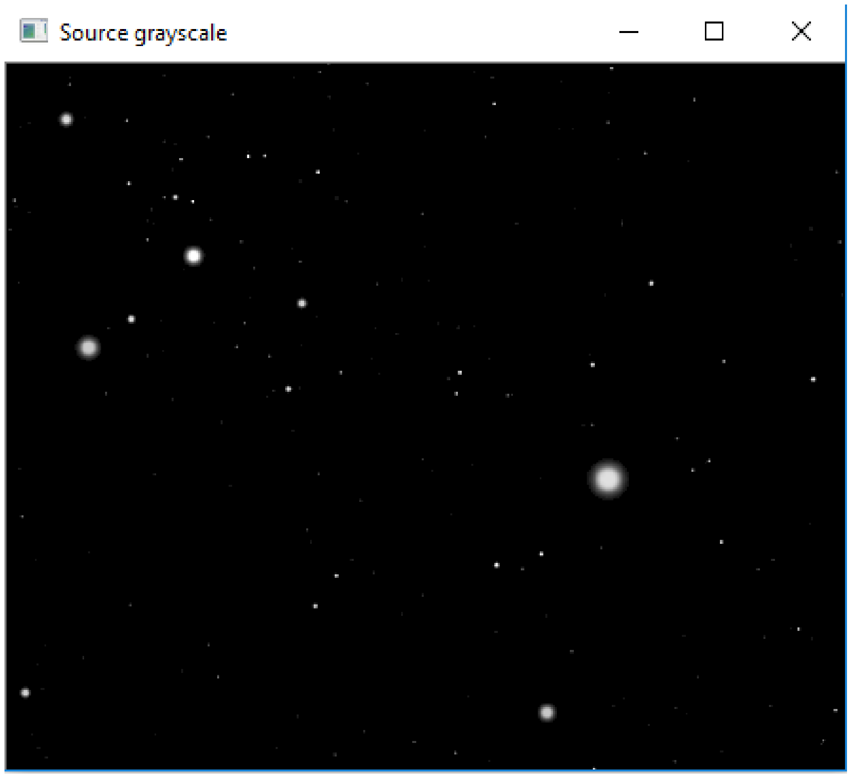


Figure 5.18 Example of input image

5.2.2. Binarization

In case the uploaded image of the user is in color, the image is converted to grayscale with the OpenCV imread function, by providing it with the IMREAD_GRAYSCALE parameter. For the grayscale to binary conversion, I've used a simple global threshold with an experimentally chosen value of 25, discussed at point 4.1.8.1, which yields very good results.

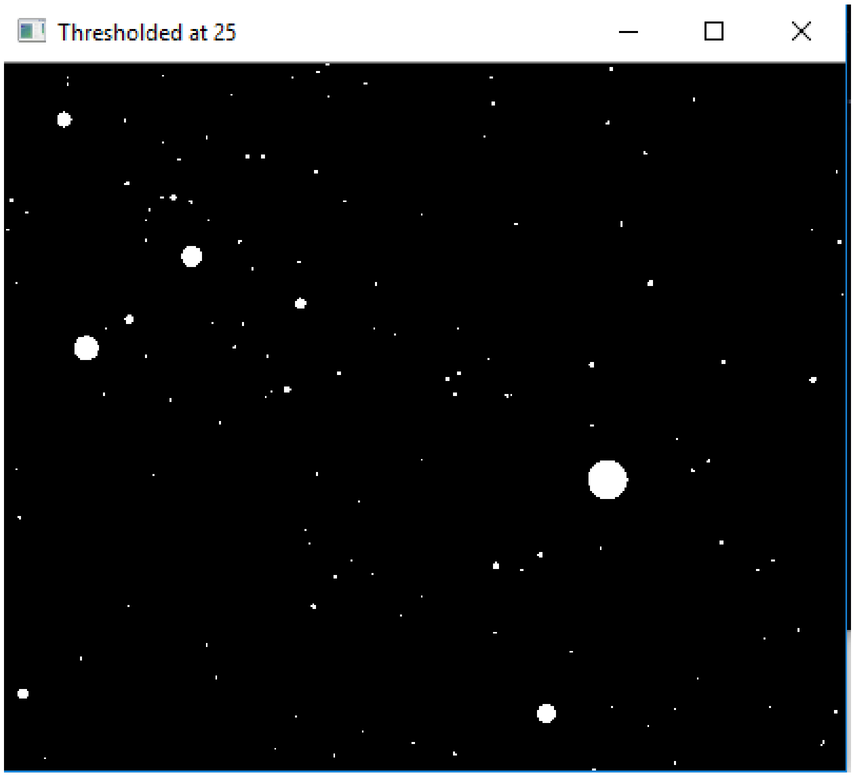


Figure 5.19 Binarized image with global threshold 25

5.2.3. Connected-component labeling

Connected-component labeling has been previously detailed in point 5.1.4, in the discussion of the constellation database building. The implementation is the same for the detection, such that the image is scanned from top-left to bottom right and the breath first search algorithm is used whenever a new object is discovered.

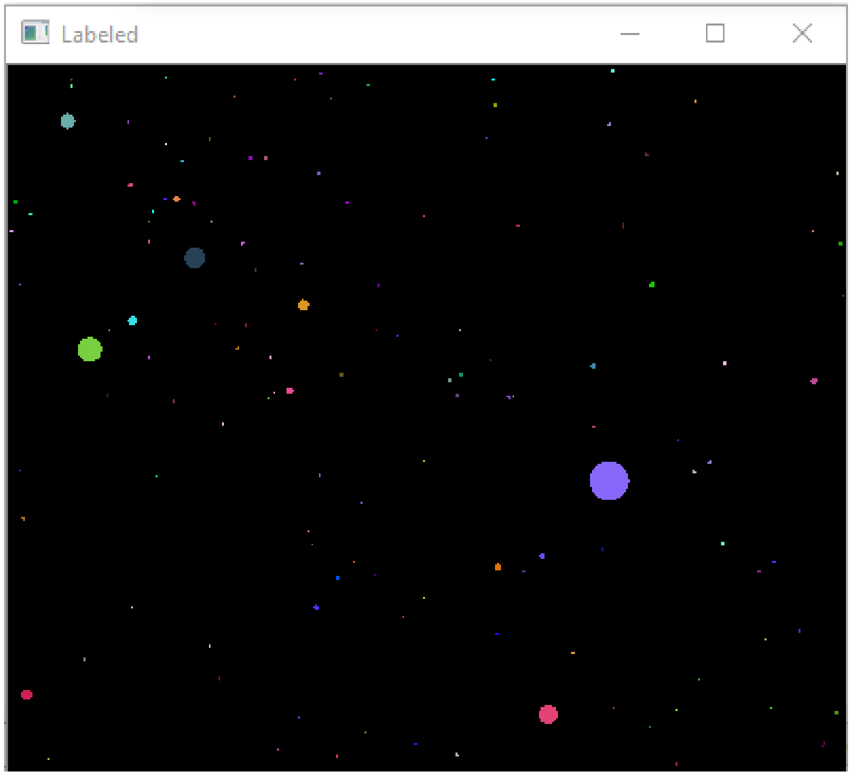


Figure 5.20 Labeled image

5.2.4. *Geometrical properties computations*

This section is the same as section 5.1.5, the center of masses and areas are computed in order to be used later. The areas of each star, in this case, could also be called the luminosity of the star.

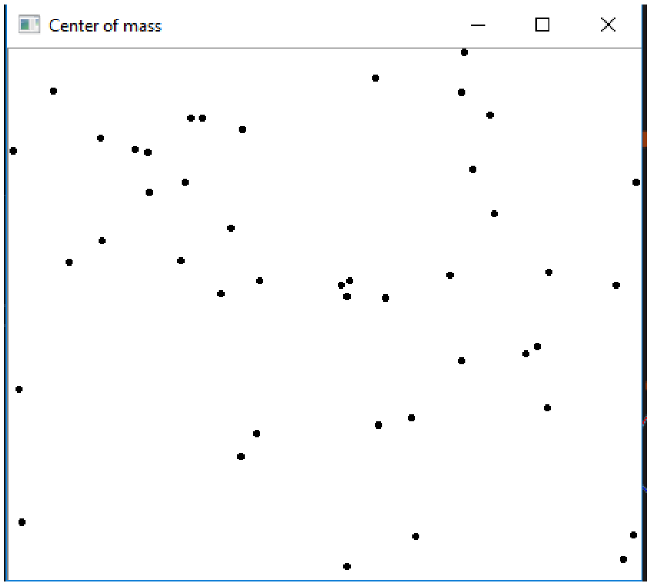


Figure 5.21 Centers of mass (image was altered for visibility)

The center of mass computation also has an in-built area filter, allowing only stars with an area greater than a threshold to remain valid for matching. This threshold has a value of 2, chosen experimentally. This is the reason why there are less stars in figure 5.21 than in figure 5.20.

5.2.5. Generation of triangles

The main generation of triangles process is identical to the process described at point 5.1.6, with one difference. Using all the stars in the image, would result in a gargantuan amount of triangles generated for the input image which would be directly proportional with the number of stars in the image. Instead, I've decided to first sort the stars descending with respect to their areas and only generate triangles with the most brightest X stars, aspect also mentioned in paper [3], page 4. By choosing X=20, this allows us to have at most combinations of 20 taken 3 triangles in the input image, which is equal to 1140 triangles.

Besides the area filter mention in section 5.2.4, a filter for the triangles also exists, in order to prevent the degenerate triangles, aspect mentioned in section 4.1.2. The filter checks if for each triangle that the sum of distance1 and distance2 is within an interval defined as [distance3 – degenerate_threshold, distance3 + degenerate_threshold]. If this is the case, the triangle is discarded as it is considered degenerate.

5.2.6. Triangle similarity matching

Sections 5.1.7 and 5.1.10 went into detail about how the continuity and consistency is maintained while building the triangles in order to ensure the process of similarity matching is behaving correctly.

Similarity is determined with the side-side-side method, so by comparing the distances of each side of one triangle with the distances of each corresponding side of another triangle, we can determine similarity. Since the distances are normalized to the smallest side of each triangle, the difference between two triangles is computed as the sum of absolute difference between each side. If the difference between the triangles is different than some threshold, than the triangles are considered similar.

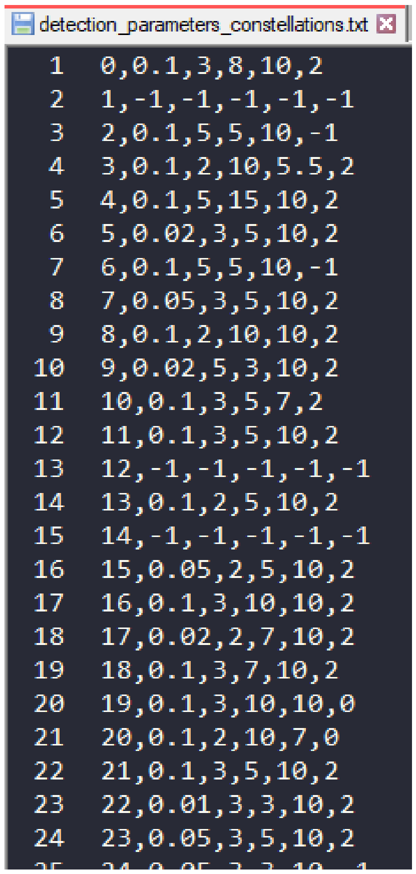
Each constellation has custom parameters:

- triangles difference threshold – differences below threshold consider the triangles as similar.
- the area or luminosity of stars threshold – to consider only stars with an area greater than the threshold as eligible to be part of the current constellation
- position displacement error – in case the warping is slightly offset due to rounding, calculations and the difference between the image models and the input images from Stellarium, it allows stars to be in a certain radius of the supposed expected location
- mean luminosity of constellation threshold – in order to prevent false detections consisting of only low luminosity stars
- the method of resolving multiple detection – multiple detections of the same constellation can be resolved by either choosing the detection with

the highest luminosity, or the detection with the lowest triangles difference, or any of the methods.

These custom parameters were obtained after a thorough series of experiments, where each combination of the parameters were tested in order to determine the best results. It resulted in a .txt file, which can be seen in figure 5.22. Each line is of the form: constellation_nb, triangles_difference_threshold, area_threshold, position_displacement_error, mean_luminosity_threshold, method_of_resolving_multiple_detections

If the last parameters is -1, then that constellation is not taken into consideration in the detection process because of the number of stars in the constellation being too low. Three star constellations are nearly impossible to detect correctly without any kind of information about the location where the input image was taken, and two stars are impossible to detect due to the whole process of similarity matching requiring at least three stars in a constellation.



```
0,0.1,3,8,10,2
1,-1,-1,-1,-1,-1
2,0.1,5,5,10,-1
3,0.1,2,10,5.5,2
4,0.1,5,15,10,2
5,0.02,3,5,10,2
6,0.1,5,5,10,-1
7,0.05,3,5,10,2
8,0.1,2,10,10,2
9,0.02,5,3,10,2
10,0.1,3,5,7,2
11,0.1,3,5,10,2
12,-1,-1,-1,-1,-1
13,0.1,2,5,10,2
14,-1,-1,-1,-1,-1
15,0.05,2,5,10,2
16,0.1,3,10,10,2
17,0.02,2,7,10,2
18,0.1,3,7,10,2
19,0.1,3,10,10,0
20,0.1,2,10,7,0
21,0.1,3,5,10,2
22,0.01,3,3,10,2
23,0.05,3,5,10,2
```

Figure 5.22 Parameters for each constellation

5.2.7. *Warping*

After succesfully finding a pair of triangles considered similar, an affine transformation is used to warp the points in the constellation model to the points in the input image. The affine transformation matrix is obtained with the getAffineTransform method from the OpenCV library, which takes as parameters the pair of similar triangles.

The model points are then warped and transformed into points with coordinates in the input image.

The affine transformation matrix is basically the figure 4.18, but the matrix A and vector B are concatenated in order to form a matrix with 2 rows and 3 columns.

5.2.8. Source image matching confirmation

After the points from the model are warped into the input image, we now have to confirm that there are actually underlying points in the input image, where the warped points indicate there should be. The position displacement error variable plays an important role here, to allow for some error window in terms of the exact location of the warped points, such that if there isn't a star in the input image in the exact location indicated by the warped points, the position displacement error allows the closest star in a certain radius to be selected as the searched star. After each star found, they are added to a list and a mean luminosity is computed for the whole list. If this list has as many stars as the constellation model, then a detection is formed.

5.2.9. Matching filtering

Because a constellation can be detected multiple times in an image, it's obvious that at most one of the detections is correct, if the constellation is even present in the image. Due to this, each constellation is allowed to show a single best detection, determined by the variable for resolving multiple detections, discussed at section 5.2.6.

5.2.10. Drawing constellation(s)

Drawing the constellations on the image consists of drawing the:

- (Optional) Matched pair of triangles
- Lines
- Stars
- Name of the constellation

Drawing the matched pair of triangles is optional, because it is only interesting to people that are curious about the way the system works. For regular users, this feature would only be distracting from the real purpose of the application. The matched pair of triangles is drawn with the help of the line function from the OpenCV library.

Drawing the lines of the triangles is one of the main important things because it makes the constellation extremely visible. Each detection holds a list of integers called matchedIndexes and a list of the matched points in the source image called srcPoints. As lines are defined by the indexes of the points in the constellation model, as shown in figure 5.16 at lines 18-29, the matchedIndexes list holds the correspondence between the points in the model and the points in the detection. I've also implemented this drawing with the line function from the OpenCV library.

Drawing the stars is done by simply calling the OpenCV function circle, on the list of matched points in the source image of the detection, with a radius of 7.

Drawing the name of the constellation is done with the OpenCV function putText, with the name being taken as the first string in the constellation model processing file(figure 5.16, line 1).

5.2.11. Altered input image(s)

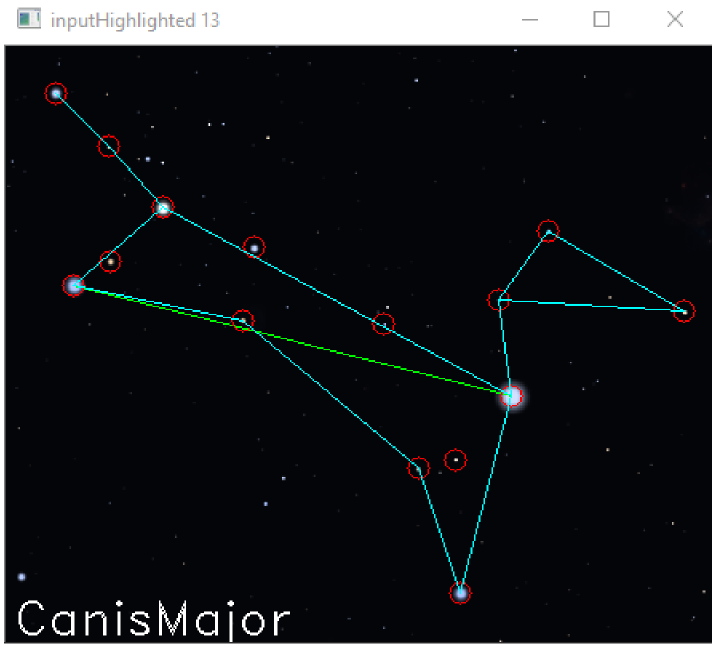


Figure 5.23 Final result of Canis Major detection, with the matching triangle shown(red and blue sides are obstructed by the light blue constellation lines)

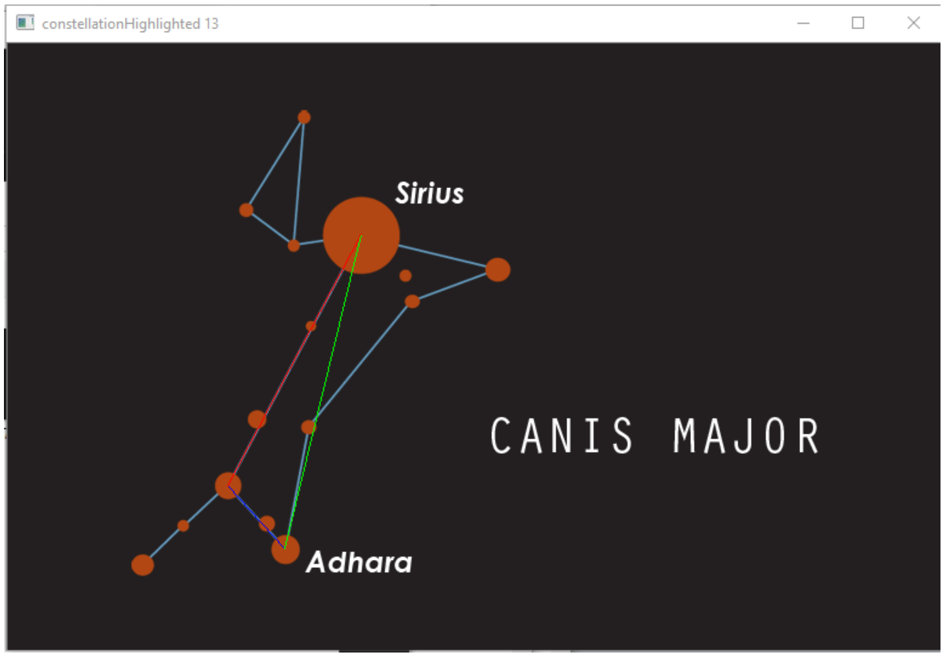


Figure 5.24 Canis Major model with the matching triangle shown (image optionally shown to user)

The title says image(s) because if multiple constellations are detected, then other constellations are also shown. This allows for detection of multiple constellations in one image, rather than just one "best" constellation.

5.3. System Sequence Diagram

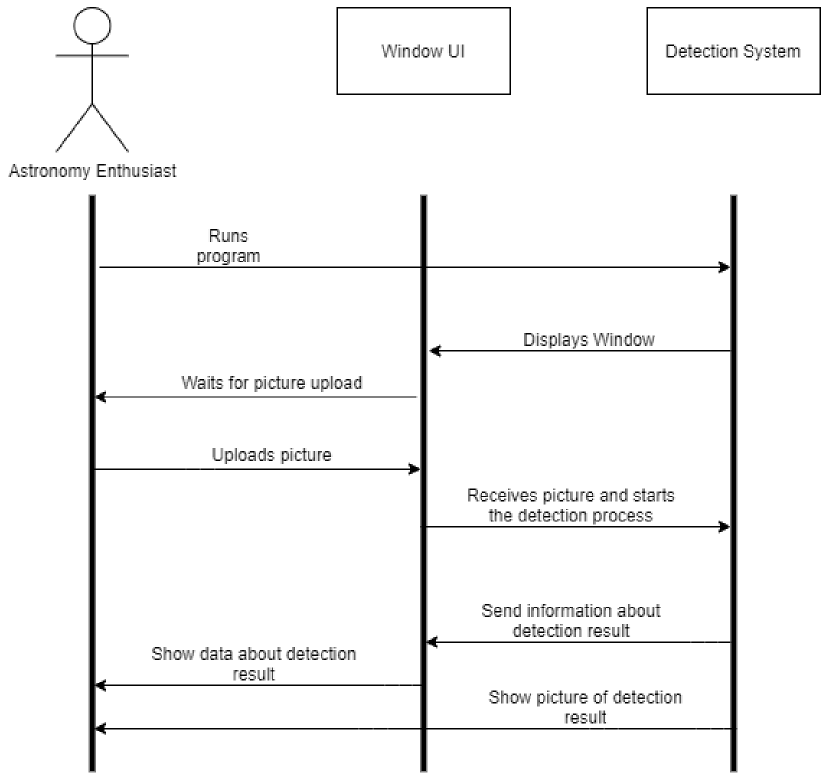

Window UI
Detection System
Astronomy Enthusiast
Runs program
Displays Window
Waits for picture upload
Uploads picture
Receives picture and starts the detection process
Send information about detection result
Show data about detection result
Show picture of detection result

Figure 5.25 System Sequence Diagram

5.4. Use case specification

The whole application only has one use case: the detection of constellations. This use case covers both the purpose and functionality of the application. The only actor involved in the use case is the user of the system, the Astronomy Enthusiast, which is interested in having a simple way to upload his picture from the computer and receive an altered version of the image in a short time that shows the detected constellations. A flow of events can be seen in figure 5.26, which will be detailed after.

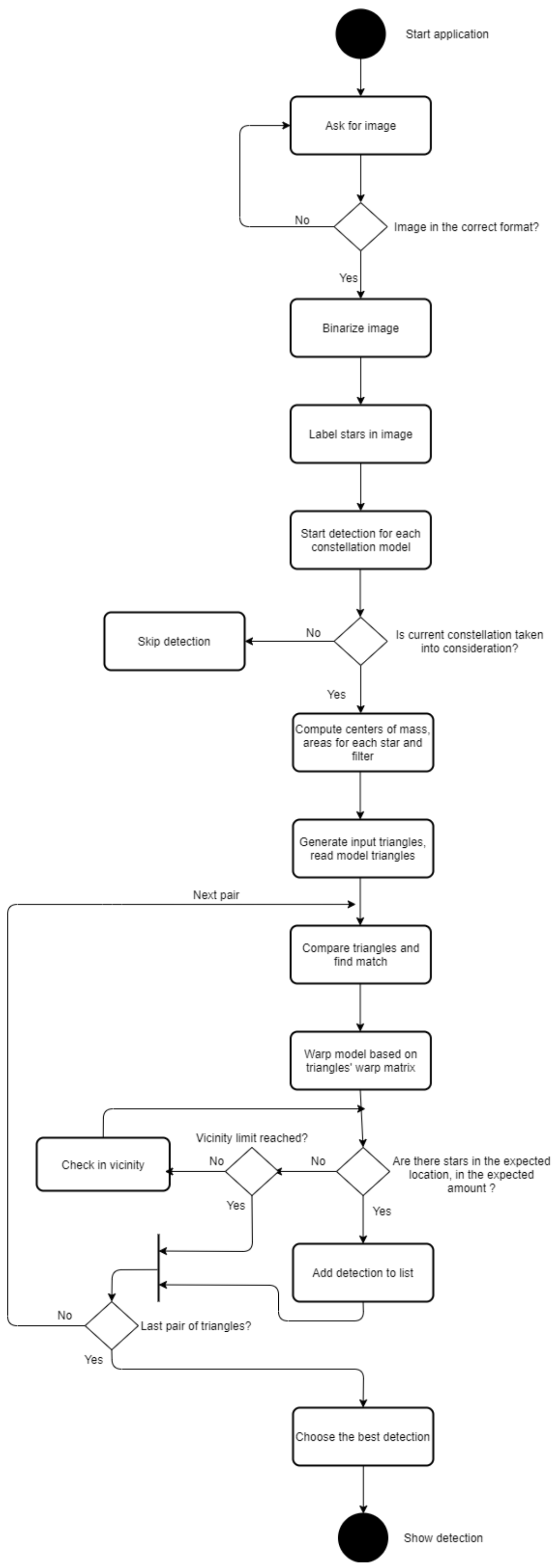

Start application
Ask for image
No
Image in the correct format?
Yes
Binarize image
Label stars in image
Start detection for each constellation model
Skip detection
No
Is current constellation taken into consideration?
Yes
Compute centers of mass, areas for each star and filter
Generate input triangles, read model triangles
Next pair
Compare triangles and find match
Warp model based on triangles' warp matrix
Vicinity limit reached?
Check in vicinity
No
No
Are there stars in the expected location, in the expected amount ?
Yes
Yes
Add detection to list
No
Last pair of triangles?
Yes
Choose the best detection
Show detection

Figure 5.26 Flow of events

This use case starts when the actor wants to use the application.

a. The system opens a window that allows the upload of the image.
b. The actor selects an image on its computer to upload.
c. The system binarizes the image.
d. The system labels the stars in the image.
e. The system computes the centers of mass and areas of each star and filters them.
f. The system generates input triangles and reads model triangles.
g. The system compares triangles and finds matches between similar triangles.
h. The system warps the constellation model.
i. The system checks for detection.
j. The system shows the detection found.

Alternative flows can be occur and they are as follows:

1. Re-enter image

This flow can occur in step b.
The system checks if the uploaded image is in the correct format. Go to step a.

2. Check for detection on current location

This flow can occur in step i.
The system checks if stars are present in the input image, where the constellation model is warped. Go to step C.

3. Check if vicinity limit has been reached

This flow can occur in step i.
The system checks if the vicinity limit of the position displacement error has been reached. Go to step B.

4. Check if last pair of triangles

This flow can occur in step i.
The system checks if the current pair of compared triangles is the last one. Go to step g.

5.5. Used technologies

In order to develop the application, I’ve used the C++ programming language, the OpenCV library for C++ and Microsoft Visual Studio as an integrated development environment.

Chapter 6. Testing and Validation

6.1. Custom parameters of each constellation

The parameters: triangles difference threshold, the area or luminosity of stars threshold for individual stars, position displacement error, mean luminosity of constellation threshold and the method of resolving multiple detection, detailed at section 5.2.6, were obtained after extensive testing. Various combinations of these parameters have been tried for each constellation, and the combination that resulted in the most correct detections has been chosen.

6.2. Detection process testing

The application went through three phases that were evaluated and tested. The test set consists of 95 total images. Six of them contain two constellations per image, while the rest of 89 have been chosen randomly from the dataset, containing one constellation per image. This gives us a result of 101 total correct constellations in the 95 total images.

The three phases of the application are composed as such:

1. Using parallelization
2. Using parallelization and triangle degenerate filter
3. Using parallelization, triangle degenerate filter and brightest stars limit

Phase 1 consists of modifying the sequential detection of each constellation to parallel detection. Given the detection of one constellation does not influence the detection of another constellation in our case, adding parallelism was a greatly justified addition. Phase 1, however, had some rough results: an average time of detection/image of 71.06 seconds and a false positive rate of 1.22/image even though it did have a recall of 84.15%.

Phase 2 consists of adding a triangle degenerate filter on top of phase 1. Triangle degeneracy has been detailed at section 4.1.2. The filter improved the system all around, with no losses. The results of phase 2 were: an average time of detection/image of 51.56 seconds, a false positive rate of 0.82/image and a recall of 84.15%.

Phase 3 consists of adding a brightest star limit on top of phase 2. This filter consists of sorting the stars in the input image descending by their luminosity and then only creating triangles from combinations of the brightest stars. Experimentally, limit of the brightest stars was chosen to be 20. This phase drastically improved the detection, with minimal losses in recall. Thus, the results of phase 3 were: an average time of detection/image of 2.76 seconds, a false positive rate of 0.03/image and a recall of 83.16%.

Phase	Total images	Total correct constellations	Total correct constellations detected	Total false constellations detected	Total time(seconds)	Average time of processing (seconds)/ image	False positives/image	Recall
Phase 1	95	101	85	116	6751	71.06	1.22	84.15%

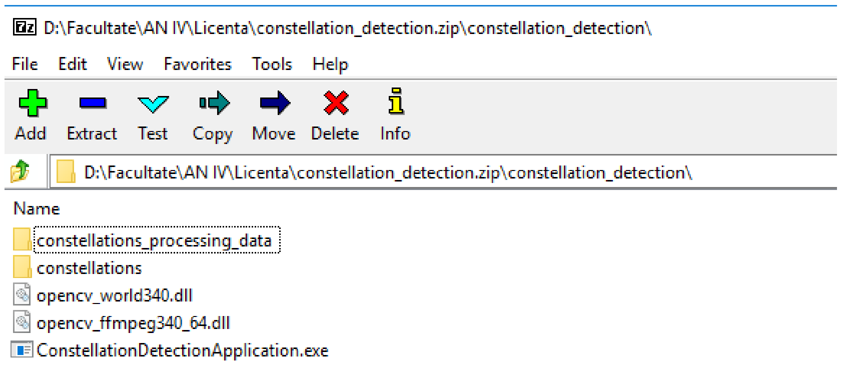
Phase 2	95	101	85	78	4898.65	51.56	0.82	84.15%
Phase 3	95	101	84	3	262.72	2.76	0.03	83.16%

Table 1 Testing results

Chapter 7. User's manual

The user should first install the Microsoft C++ Redistributable 2019 package on machine running the Windows 10 x64 operating system. Other versions of the Microsoft C++ redistributable package and operating system should also be compatible, although this was not tested.

The application was designed in order to be very easy to setup and use by a basic user. Therefore, the user would receive a .zip file, of which contents need to be extracted. It contains the executable file ConstellationDetectionApplication, which is the one that needs to be run for using the application. The files opencv_world240.dll and opencv_ffmpeg340_64.dll are required by the application in order to use the OpenCV library. The folder "constellations" contains .png files of the constellation models and the folder "constellations_processing_data" contains the constellations database, detailed at section 5.1, stored in .txt files. The zip has a size of 29 MB and the extracted contents have a size of 82 MB.

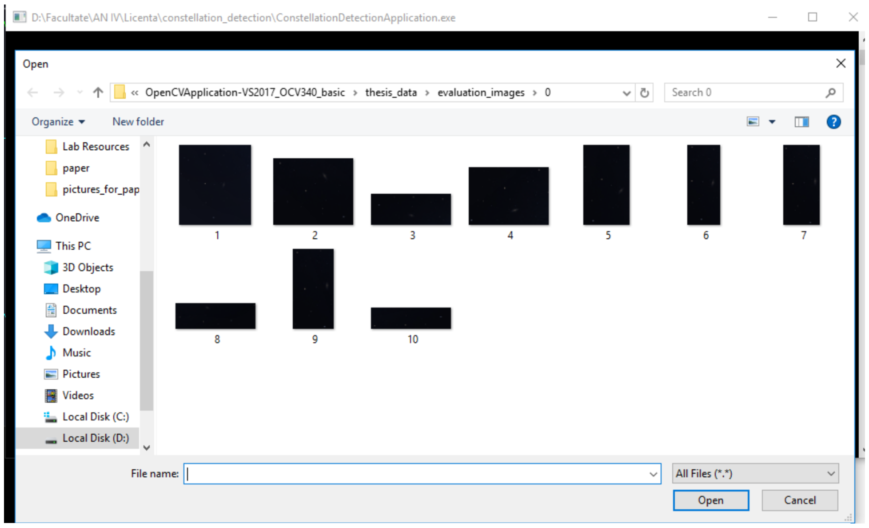


D:\Facultate\AN IV\Licenta\constellation_detection.zip\constellation_detection\

Name
constellations_processing_data
constellations
opencv_world340.dll
opencv_ffmpeg340_64.dll
ConstellationDetectionApplication.exe

Figure 7.2 Application .zip archive file

As stated previously, the user runs the executable file, and the system will prompt a file dialog for the user to upload it's picture.



D:\Facultate\AN IV\Licenta\constellation_detection\ConstellationDetectionApplication.exe

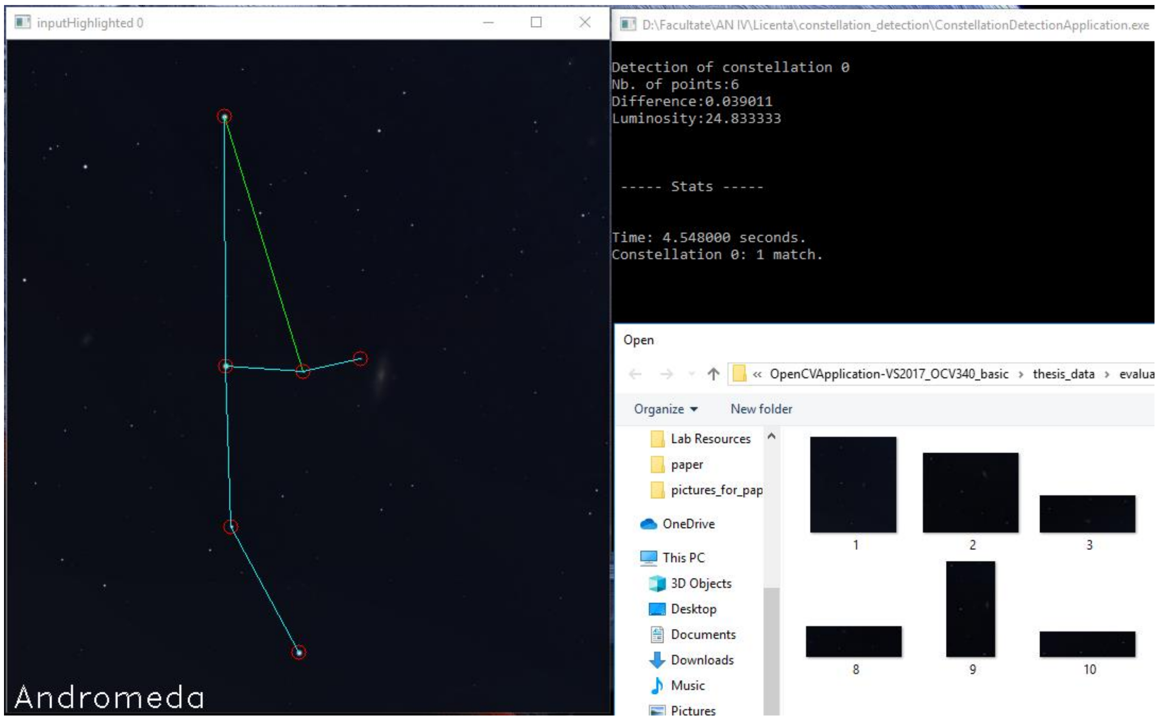
« OpenCVApplication-VS2017_OCV340_basic › thesis_data › evaluation_images › 0

Lab Resources
paper
pictures_for_pap
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Local Disk (D:)

1 2 3 4 5 6 7
8 9 10

Figure 7.3 File dialog to select image

After selecting the image with the file dialog, the application start the detection. At the end, it will provide some statistics and the images of the detection process. The application will also start another file dialog, waiting for another file to be uploaded. In order to close the application, the user needs to close the console opened by the executable file by clicking on the red X. This will close all other windows of the application.



```
Detection of constellation 0
Nb. of points:6
Difference:0.039011
Luminosity:24.833333



 ----- Stats -----


Time: 4.548000 seconds.
Constellation 0: 1 match.
```

Figure 7.4 After selecting an image and waiting 4.54 seconds for the detection process

Chapter 8. Conclusions

The application goals were met entirely. The functional requirements were proved to be met by chapter 5, as the detailed implementation shows the functional requirements in section 2.2.1 are in accord. The application allows the user to upload an image an provides them with an altered image, showing the detected constellation. The non-functional specifications mentioned in section 2.2.2 were achieved as follows:

- the applications' performance is very good, with an average time of detection/image of 2.76 seconds;
- the application's usability is also very good, as the user can receive a .zip file and only have to extract the contents and run an executable in order to use the application;
- the availability is great because the application does not use the internet or does not have any other external dependencies other than on the operating system on which it runs;
- the scalability, concerning developers, is good but could be improved by modularizing the implementation further;
- the reliability is also good, considering the 3% chance of a false detection/image and the recall of 83.16%. Reliability could be improved by more accurate constellation models

8.1. Further developments

As mentioned in the previous section, some more accurate constellation models could produce a higher percentage of correct constellation detections. Improving the precision of computations in the detection process could also improve the recall and overall accuracy of the system, as currently the implementation of the detection process has some conversions between float and integer which might influence the reliability of the system.

An improvement in the overall view of the application would be to take into consideration how the constellations will change in the future. The constellation models used for the application depict how the constellations look if an observer would look at the sky in the current time. This is more than enough as within a human life-span the differences in the distances between the stars of a constellation are unperceivable by the human eye. However, it would be interesting to supply the application with models of constellations from the future where the differences in distance can indeed be seen and recognized by the human eye. This could also provide a new feature to the application, do determine the period in which the image was taken by recognizing that the distance of a certain constellation belong to a certain period in time.

Having information regarding the location and time from where the photograph was taken could improve all the aspects of the application. This would however reduce usability if the user would have to input the information themselves. A more approachable solution would be if the image contained meta-data which could be interpreted by the application and used in order to filter out constellations that would be impossible to be seen.

As an opposite of the last proposed further development, it would also be interesting to develop a feature that allows the application to identify the possible locations and time frames from where the photo was taken, by considering the constellations present in the image.

Bibliography

[1] E. Delporte, Délimitation scientifique des constellations: (tables et cartes), Cambridge University Press, 1930

[2] I. Ridpath, Star Tales, ISD LLC, 2018, chapter 1, p.4. [Online]. Available: http://www.ianridpath.com/startales/startales1d.html

[3] C. Padgett, K. Kreutz-Delgado, S. Udomkesmalee, Evaluation of Star Identification Techniques, Journal of Guidance, Control and Dynamics, 1997

[4] F. Zhou, T. Ye, Lost-In-Space Star Identification Using Planar Triangle Principal Component Analysis Algorithm, Mathemathical Programs in Engineering, 2015

[5] D. Rijlaarsdam, H. Yous, J. Byrne, D. Oddenino, G. Furano, D. Moloney, Efficient Star Identification Using a Neural Network, Sensors, 2020

[6] M.J. Domeika, E.W. Page, G.A. Tagliarini, Neural network approach to star field recognition, Applications and Science of Artificial Neural Networks, 1995

[7] Hopfield Network, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Hopfield_network

[8] X. Liu, J. Suyao, J. Wang, Constellation Detection, Stanford University, 2015. [Online], Available: https://web.stanford.edu/class/ee368/Project_Spring_1415/Reports/Ji_Liu_Wang.pdf

[9] Astrometry.net, https://astrometry.net/

[10] Similarity in geometry, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Similarity_(geometry)

[11] Euclid, Elements, Book VI, C. 300 BC, Propositions 4-6

[12] Affine transformation, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Affine_transformation

[13] Degeneracy in mathermathics, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Degeneracy_(mathematics)

[14] Taxicab geometry, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Taxicab_geometry

[15] Chebyshev distance, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Chebyshev_distance

[16] Euclidan distance, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Euclidean_distance

[17] Image Processing - Laboratories 2, 4, 5, 7, Laboratory guide, Technical University of Cluj-Napoca. [Online] Available: https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/137-6.pdf

[18] OpenCV Library Documentation, Version 3.4.14

[19] S.D. Yanowitz, A.M. Bruckstein, A New Method for Image Segmentation, Computer Vision, Graphics and Image Processing, 1988.

[20] Astronomy online website, http://astronomyonline.org/Observation/Constellations.asp

[21] Stellarium Astronomy Software. [Online]. Available: http://stellarium.org/
[22] Ursa Major, Wikiwand. [Online]. Available: https://www.wikiwand.com/en/Ursa_Major
[23] Polygon center of mass, Sky Coyote. [Online]. Available: https://www.skycoyote.com/cntr/

Appendix 1 (only if needed)

…
Relevant code sections
…
Other relevant info (proofs etc.)
…
Published papers (if any)
etc.