# Image matching and registration

## Introduction

There are multiple possible directions the project can involve in. Since other tasks were already taken, I decided to go with “image matching” which is related to template matching, one technique of object detection. It relies heavily on detecting points of interest or “edges” in images.

In the context of the project, image and template matching could possibly be used to construct a map from multiple 2D flat images or from the drone footage, we’ll need to apply edge detection ( point of interest detection) on multiple consecutive images that we got from a video of the sea in order to construct a final map.

## Key Points and Feature Detection in Computer Vision

Identifying key points or features in images is essential for many tasks in computer vision, like matching, aligning, and transforming images. Key points are distinct and unique parts of an image that can be detected and used to align or stitch multiple images together.

## Harris Corner Detection Algorithm

The Harris corner detection algorithm is a common method for finding corners in images, which are important features. This algorithm uses a structure matrix to summarize gradient information around a pixel. By looking at the eigenvalues of this matrix, it can detect corners where there are significant changes in gradient directions. This helps in finding unique points in an image that remain consistent under different transformations, making it easier to match corresponding points in different images.

## Affine and Projective Transformations

Two basic types of image transformations used in computer vision are affine and projective transformations:

* **Affine Transformations**: These transformations preserve parallel lines and include operations like scaling, rotation, translation, and shearing. They are represented by a 2x3 matrix and are crucial for applications that require seamless blending of images.
* **Projective Transformations**: These transformations can handle more complex distortions, such as warping, and are represented by a 3x3 matrix. They are especially useful in creating panoramas by warping images back to a single plane, allowing multiple images to be combined into one cohesive image.

These transformation will represent the most common and possible modifications that a camera picture can apply on any given scene.

## RANSAC Algorithm

The RANSAC (Random Sample Consensus) algorithm is a method for dealing with outliers in data when performing image transformations. RANSAC iteratively selects random subsets of data points to estimate a model and identifies inliers that fit the model well. This way, RANSAC can effectively manage noisy data and find the best fit for image transformations, making it a valuable tool in scenarios where data may contain significant outliers.

## Encoding and Solving Linear Systems

Encoding and solving linear systems of equations using matrices is a key technique in finding affine transformations between matched points. This involves forming matrices from the known values and solving for the unknowns using methods like least squares minimization. This approach ensures accurate transformations even in overdetermined systems, where there are more equations than unknowns.

## Homography and Image Stitching

Homography is a type of projective transformation used to align images of the same planar surface. It involves finding a matrix that projects points from one image to another while maintaining the geometric relationship between them. Homography is crucial for applications like panorama stitching, where multiple images need to be combined into a single, seamless image.

## Feature Descriptors: HOG and SIFT

Feature descriptors are used to describe and match key points between images:

* **Histogram of Oriented Gradients (HOG)**: HOG descriptors help reduce dependence on lighting conditions by normalizing gradient magnitudes and focusing on gradient directions. This method provides some invariance to lighting changes but is not rotation invariant.
* **Scale-Invariant Feature Transform (SIFT)**: SIFT addresses the limitations of HOG by providing both rotation and scale invariance. SIFT detects key points in images and calculates descriptors that are robust to changes in scale, rotation, and illumination. This makes SIFT particularly effective for matching features across different images, even under significant transformations.

## Advanced Image Stitching Techniques (Possible improvements)

To overcome the limitations of straightforward projective transformations, advanced techniques are used:

* **Cylindrical Projection**: Modeling the world as a cylinder for image stitching reduces distortion and allows for the creation of wide-field-of-view images. This involves transforming image points into cylindrical coordinates and then projecting them back onto a flat plane.
* **Alpha Blending and Multiband Blending**: These techniques enhance the visual quality of stitched images by smoothly blending overlapping regions, reducing noticeable seams and discontinuities.

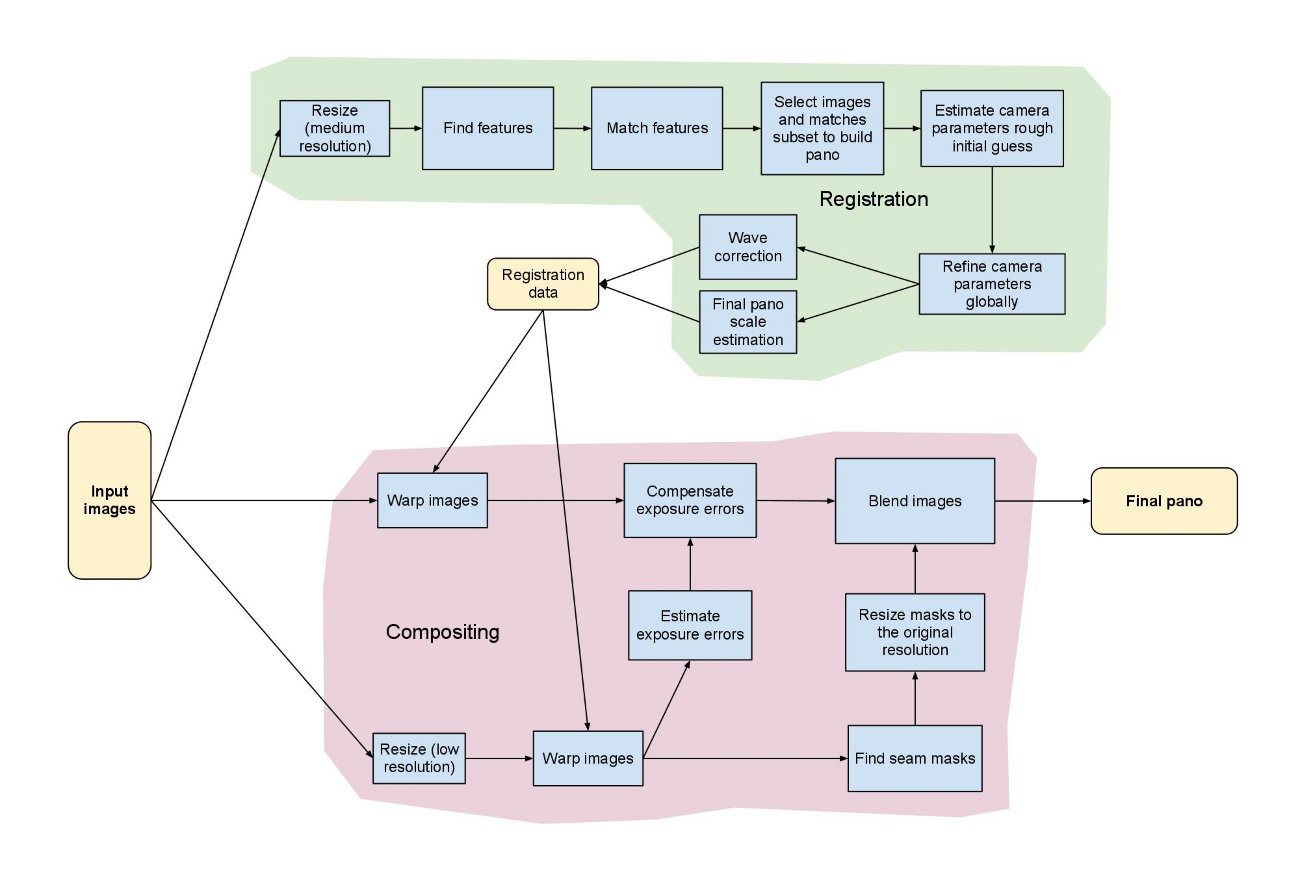
## Transition to Neural Networks

While traditional methods like SIFT were groundbreaking in feature detection and matching, modern computer vision applications increasingly rely on neural networks. Neural networks offer superior performance and flexibility in handling complex image transformations and feature extraction tasks, making them the preferred choice in contemporary computer vision systems.

## Implementation

I’ll build a working program first with data that I got elsewhere, then I’ll work on data of the project if there is any given.

OpenCV has a built in stitching method that is usually used to combine multiple images together into one big panoramic picture. We can use this feature without necessarily understanding the exact details of the stitching pipeline, but here is a diagram showing the steps:



Stitching pipeline from OpenCV documentation

We can start with 2 random frames to test out the result of this feature before applying it to a complete video sample.

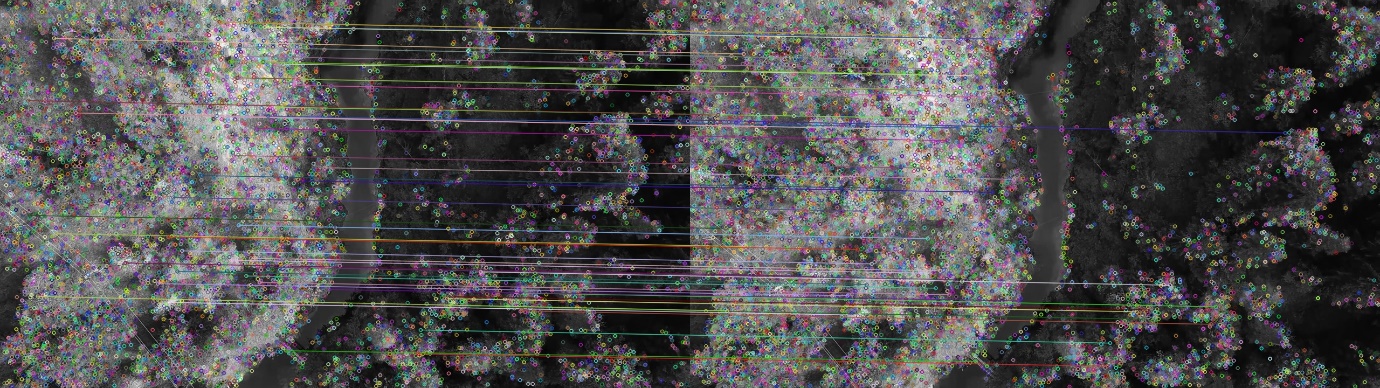
By using the 1st and 4th pictures of the sample video, we get the following result:   
 

Blended picture of 1st and 4th frames

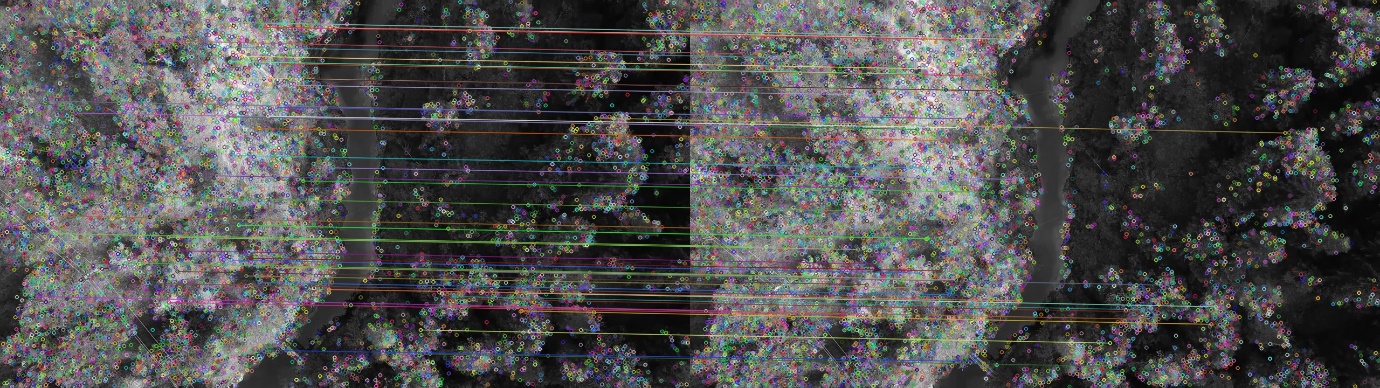
## Keypoints extraction



## Image matching



Brute-Force matching



FLANN matching

Having mostly (or entirely) horizontal lines with no intersections could be a sign that the matching is correct, meaning the keypoints represent the same feature in both pictures.

The rest of the results are in the project repository: <https://github.com/fayssalElAnsari/Posidonea_Oceanica>

## Second week

After learning more about the theory and ideas behind image matching and detection of interest points and edges. We’ll now work on a straightforward application of these notions without the need to program anything from scratch.

The need of suggesting image matching comes from wanting to build an accurate 2D map from the drone footage we took. There are built-in features in some drones that offer this photogrammetry option out of the box. But we already have a video taken of the studied area which we’ll use.

### Autostitch

From the Wikipedia page of “image stitching” the last chapter is about famous software that’s used to extract a map from video footage or from a list of pictures.

We start off by testing “Autostitch”, to use it we’ll need to firstly extract frames from the given video we’ll use chatGPT to produce a simple script that allows us to also show progress and define the desired frames to skip during extraction.

The first 10 frames skipped test exceeds the image size required when working with autostitch so we’ll use 20 frames now.

Autostitch uses both STITCH and RANSAC which we already talked about in a previous chapter. The stitching process is supposed to be pretty streamlined where the user does not have to modifie any input, but it has a small problem that it needs to have all the pictures taken from one singular point, which is not the case for most of the video given to use, there are some snippets where the camera is rotated across the view without translating it in 3D space. So we could work with those if the whole video does not work.

The result we got using **Autostitch**:

### Hugin

Hugin is another program that allows for image stitching with minimal configuration of the user’s part. In Hugin stitching is accomplished by using several overlapping photos taken from the same location, and using control points to align and transform the photos so that they can be blended together to form a larger image. Hugin allows for the easy (optionally automatic) creation of control points between two images, optimization of the image transforms along with a preview window so that the user can see whether the panorama is acceptable. Once the preview is correct, the panorama can be fully stitched, transformed and saved in a standard image format.

But since we do not want nor have the intention of creating a panoramic photo the results were as expected to be unsatisfactory.

### Agisoft Metashape

This software which is heavily used by photogrammetry enthusiasts offers many features with the possibility of tuning different parameters, this software offers many functionalities that span beyond the stitching application we want to apply in this project.

#### Data quality limitations

In general the quality of results we’ll get are heavily dependent on the quality of the data we have. When working in photogrammetry and especially with drone footage to create a 2D map there are a couple of norms to follow to minimize as much as possible errors and incertitude.

There are some programs that allow us to define the region of field we want to map, and automatically the drone will have a predefined path to follow with built-in obstacle avoidance while also keeping a fixed altitude, with a custom overlap area between the taken pictures. In the end this makes it possible to get better data in a more automatic manner while also minimizing the drone operator’s efforts.

There are a couple things that make it a bit harder to work with the data that we currently have such as occasional zoom-in and zoom-outs, turning the drone in 3D space instead of keeping it facing down at all times. Not having the measurements of angles (Yaw, Pitch Roll) included in the given SRT file. And finally since the famous stitching algorithms used in image stitching rely heavily on detection of interest points, and since we’re working with sea pictures it is possible to need to extract more frames from the video In order to keep better track of the points relations, due to the fact that the background will keep on changing when the drone is in the middle of the sea.

#### Procedure

We cannot work directly with the given videos, Agisoft Metashape uses frames as a first step. So we’ll need to import the video into different chunks, one by one, while defining a frame skip and a folder to keep the frames in for project use. The name of each frame should follow a certain convention, this same convention will be used to pair each frame with its geolocation information.

After importing the geolocation csv file that we extracted from the srt file, we’ll have under the reference tab the location of each frame in space, and we can see a detailed 3D image of the frames we have selected.

Now that we have imported all the info we have and we’ll use to stitch the images together we’ll need to follow the steps of this tutorial: <https://www.agisoft.com/pdf/PS_1.3%20-Tutorial%20(BL)%20-%20Orthophoto,%20DEM%20(GCPs).pdf>

We could try to only take the frames where the camera is facing down and see what that gives us as a result, this way we’ll not need to keep track of the angular orientation in 3D space. We’ll have less frames but as long as there is some overlap between the pictures there should be a possibility to align the images together and align them to produce our 2D map.

#### Starting test

As a starting practice we’ll work with a top down drone footage available online where the camera is panning across a beach: <https://www.youtube.com/watch?v=nrhQqQ5fc9E>

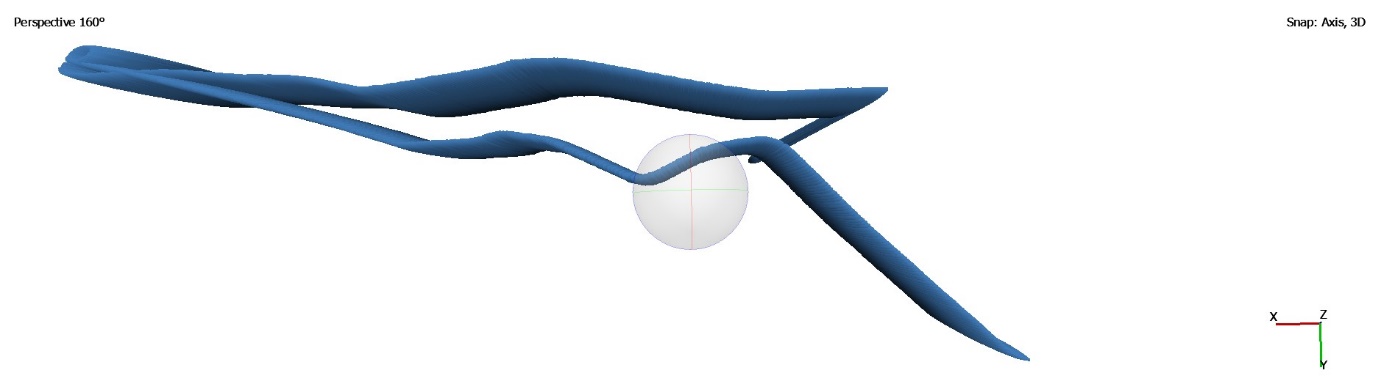
After following the steps of building an orthomosaic view we get this result:



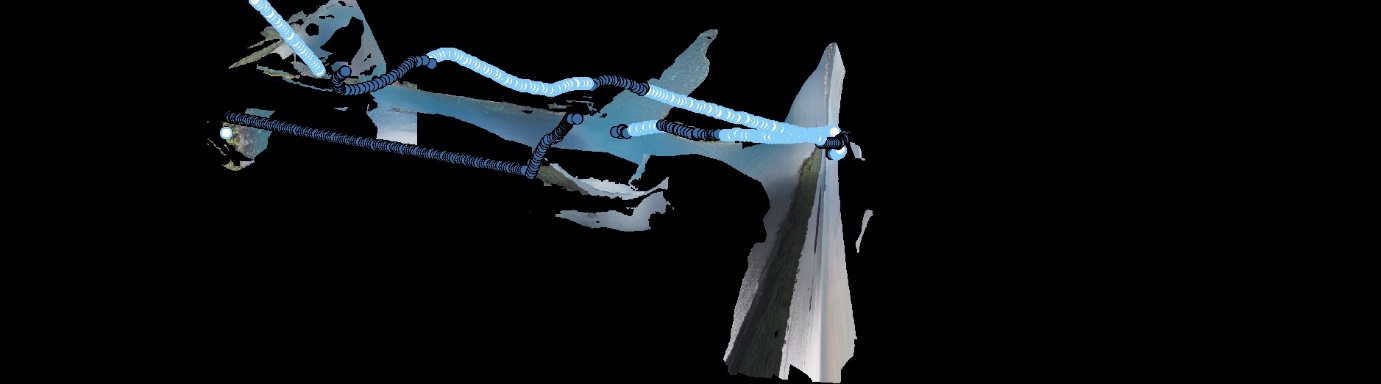
Going back to our own drone video footage.



The end result is unsatisfactory.

We’ll use another video from the second data folder given to us by Louis Vraie, same procedure. We’ll extract and load the positions as a reference. We can see the trajectory the drone followed during its trip in 3D space:  
 

Even though it is not an organized trip we should be able to select the frames we’ll use to get the best possible results. This video has many shots with the beach visible and some other objects that could serve as keypoints for stitching such as trees and rocks.



Still an unsatisfactory result

We might need to do a lot of manual editing in order to improve the result’s quality, or we could as stated earlier manually extract frames that seem to be directly facing down with their location data and only use these frames, but this could take some time.

# Conclusion

Despite the various attempts with different software, the quality of our results was consistently poor. Improving the results might require significant manual editing or better-quality footage with more controlled data collection parameters.

## Acknowledgments

I would like to extend my heartfelt gratitude to Professor Jean Martinet and the previous team, especially Louis Vraie, for their tremendous efforts and the invaluable resources they have provided. Their dedication and support have been instrumental in ensuring a smooth and efficient onboarding process. I am deeply appreciative of their guidance and the helpful insights they have shared, which have greatly facilitated my understanding and progress in this project. Thank you for making this experience both enriching and enjoyable.

## References and sources used

<https://github.com/mikeroyal/Photogrammetry-Guide>

<https://docs.opencv.org/4.x/d1/d46/group__stitching.html>

<https://en.wikipedia.org/wiki/Template_matching>

<https://en.wikipedia.org/wiki/Hugin_(software)>

<https://www.youtube.com/watch?v=taty6lPVcmA&list=PLjMXczUzEYcHvw5YYSU92WrY8IwhTuq7p&t=3026s>

<https://ai.stanford.edu/~syyeung/cvweb/tutorial1>