TÉCNICO LISBOA

Vision-based Navigation supported by Convolutional Neural Networks for Lunar and Planetary Landing Missions

Pedro Maia Pinheiro

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Rodrigo Martins de Matos Ventura
Dr. Nuno Tiago Salavessa Cardoso Hormigo Vicente

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. José Alberto Rosado dos Santos Victor

July 2021

aos meus pais, por todo o apoio

Acknowledgments

First of all, I would like to thank my supervisors, Professor Rodrigo Ventura and Tiago Hormigo from Spin.Works for giving me the opportunity to perform this investigation, introducing me to the image processing and planetary landing fields, for which I have developed a fascinating interest and for continuously guiding me and presenting exciting new challenges.

A special acknowledgement, also, to João Oliveira from Spin.Works for having the time and patience to review with me many image processing concepts and implementation details and for all the discussions about the results that inspired me to embrace this problem.

This Thesis would not be possible without the precious help from Spin.Works, introducing me to their approach to the problem and providing several documentation and all the datasets that were essential to this work.

I would also like to thank my family, specially my parents and my brother for their continuous support, encouragement and caring over all these years. Without their help and motivation I would not start this wonderful journey of studying at Instituto Superior Técnico.

My sincere thanks to my loving girlfriend for having stood by my side, encouraging me every time I felt lost or not motivated. Her company has been essential and I hope it never changes.

Last, but not least, I am eternally grateful for the help from my friends, which includes all the travels, parties and moments that made my onboarding at a new city peaceful and amazing.

Resumo

Os sistemas de navegação auxiliada por visão são uma tecnologia relevante na indústria espacial. Este método é baseado no uso de câmaras como o principal componente de um sistema de navegação para estimar a posição e atitude relativas de um veículo espacial, nomeadamente para operações de proximidade, tais como missões de aterragem lunar e planetária. Nas últimas décadas, as técnicas de processamento de imagem convencionais têm vindo a ser substituídas por redes neuronais convolucionais num vasto número de tarefas e domínios, dado que os métodos de Inteligência Artificial têm vindo a ultrapassar em grande parte os testes de referência.

Inspirados por estes avanços promissores, nesta Tese, investigamos alternativas usando Aprendizagem Profunda para algoritmos clássicos de processamento de imagem, que possam vir a ser usadas numa navegação auxiliada por imagem, no âmbito de missões de aterragem planetária. Desta forma, propomos uma abordagem completa para avaliar detetores de pontos de interesse com o objetivo de estimar estados do movimento em missões de aterragem planetária, usando dados representativos, simulados e ainda o mais recente vídeo da NASA da aterragem do rover Perseverance. Com este objetivo, apresentamos uma solução baseada em homografias.

Uma avaliação qualitativa e quantitativa é apresentada, comparando detetores de pontos de interesse clássicos e um baseado em arquiteturas de Aprendizagem Profunda. Ao obtermos resultados promissores na aplicação de aprendizagem automática para este problema, introduzimos uma alternativa aos algoritmos clássicos de visão computacional. Para além disso, discutimos possíveis trabalhos futuros para melhorar os resultados.

Palavras-chave: Navegação auxiliada por visão, Aterragem Planetária, Homografia, Aprendizagem Profunda, Redes Neuronais Convolucionais, Detetores de pontos de interesse

Abstract

Vision-based navigation systems are a prominent technology in the space industry. This method is based on using camera images as the primary navigation system to estimate spacecraft relative position and attitude, namely for rendezvous and proximity operations, such as lunar or planetary landing missions. In the last few decades, conventional image processing techniques are being replaced by Convolutional Neural Networks in a vast number of tasks and domains, since these Artificial Intelligence methods are outperforming on most benchmarks.

Inspired by these promising advances, in this Thesis, we investigate Deep Learning alternatives to classical image processing algorithms, which may be applicable to image-based navigation in the scope of planetary landing missions. Thus, we propose a complete framework to evaluate any image feature detector for the task of motion states estimation during a planetary landing mission, using both representative and simulation datasets, as well as the most recent Perseverance Rover's landing video from NASA. To accomplish this goal, a solution based on homography relation is designed.

A qualitative and quantitative evaluation of the whole pipeline is presented, comparing both classical feature detectors and one based on Deep Learning architectures. From our promising results for applying machine learning to this problem, we introduce an alternative to classical Computer Vision algorithms. Furthermore, we discuss some possible future work to improve the results.

Keywords: Vision-based Navigation, Planetary Landing, Homography, Deep Learning, Convolutional Neural Netwroks, Feature Detection

Contents

Acknowledgments v

Resumo vii

Abstract ix

List of Tables xiii

List of Figures xv

Acronyms xix

1 Introduction **1**

1.1 Motivation 1

1.2 Problem Description 2

1.3 Outline of the Approach 2

1.4 Contributions 2

1.5 Thesis Outline 3

2 Background **5**

2.1 Feature Detection and Description 5

2.1.1 Harris Corners & Shi-Tomasi Corners 6

2.1.2 SIFT & SURF 7

2.1.3 FAST, BRIEF & ORB 10

2.2 Feature Tracking 12

2.2.1 Optical Flow 13

2.2.2 Feature Matching 13

2.3 3D Motion from 2D Image Transformations 14

2.3.1 2D Image Transformations 14

2.3.2 Projective Homography Estimation 16

2.3.3 Random Sample Consensus 18

2.3.4 Pinhole Camera Model 19

2.3.5 Euclidean Homography 21

3 Deep Learning **25**

3.1 Artificial Neural Networks 26

3.2 Backpropagation 27

3.3 Convolutional Neural Networks 27
3.4 Deep Interest Point Detection 29

4 Implementation 33

4.1 Software Tools 34
4.2 Evaluation Datasets 34
4.2.1 HPatches 34
4.2.2 Spin.Works UAV Landing Mission Dataset 35
4.2.3 Spin.Works Moon Landing Mission Simulation 37
4.2.4 Perseverance Rover's Descent and Touchdown on Mars (Official NASA Video) . . 38
4.3 Feature Detection Algorithms 39
4.3.1 Harris Corners 39
4.3.2 SIFT 39
4.3.3 SuperPoint 39
4.4 Homography Estimation & Motion Reconstruction 43
4.4.1 Ground Truth Homographies 43
4.4.2 Test Set Homographies 44
4.4.3 Non-Linear Least Squares 44
4.5 Evaluation Metrics 47

5 Results 49

5.1 HPatches 49
5.1.1 Repeatability 49
5.1.2 Homography Estimation 50
5.2 Spin.Works UAV Landing Mission 51
5.2.1 Homography Estimation 51
5.2.2 Qualitative Results 52
5.2.3 Navigation States Estimation 54
5.3 Spin.Works Moon Landing Mission Simulation 61
5.3.1 Navigation States Estimation 62
5.4 Perseverance Rover's Descent and Touchdown on Mars 67
5.5 Discussion 70

6 Conclusions 73

6.1 Achievements 73
6.2 Future Work 74

Bibliography 75

List of Tables

2.1 Hierarchy of 2D Transformations 17

5.1 Detector Repeatability on HPatches. Repeatability measures the probability that a point is detected in the second image. SuperPoint is the most repeatable under illumination changes, while Harris Corners exhibits better performance under viewpoint changes. . . . 49

5.2 Mean of the absolute value of the absolute deviations at each time step for UAV pose recovery using different feature detection methods. The values representing translations (t_x, t_y, t_z) are expressed in meters and the ones for attitude (Roll, Pitch, Yaw) are represented in degrees 57

5.3 Standard Deviation of the absolute value of the absolute deviations at each time step for UAV pose recovery using different feature detection methods. 57

5.4 Maximum of the absolute value of the absolute deviations at each time step for UAV pose recovery using different feature detection methods. 57

5.5 Statistics for the absolute value of the absolute deviations during UAV pose recovery, using SuperPoint detector with delta between frames equal to 20. Results should be compared to the last line in Tables 5.2, 5.3 and 5.4. Bold means that values are better than using delta between frames equal to 1 from the last tables. 58

5.6 Number of frames until degradation of motion states estimates for UAV landing video sequence, from different reference frames 60

5.7 Ratio of the intersection over the reference, for reference and current frames calculated from SuperPoint results of table 5.6 61

List of Figures

2.1 Corner classification as a function of the eigenvalues 7

2.2 Difference of Gaussians - Octave structure 8

2.3 Local extrema of the Difference os Gaussians 8

2.4 SIFT descriptor computation 9

2.5 SURF Box Filter using integral images 9

2.6 Haar wavelets filters 10

2.7 Accelerated Segment Test - 16 pixels circle around corner candidate 11

2.8 Image pyramid 12

2.9 2D Planar Transformations 15

2.10 Pinhole camera model representation 19

2.11 Principal point offset. Image and camera coordinate frames 20

2.12 Representation of a 2D plane in the world and respective notation 22

3.1 Architecture of LeNet-5 29

3.2 Stacked Hourglass Architecture 30

4.1 Spin.Works UAV from one landing mission on a quarry. On the left, we see the UAV with a monocular camera mounted underneath the electronic systems, pointing to the ground. On the right, we see it flying over a surface that looks similar to Mars, using a vision-based navigation system. The geometry of the problem is depicted on Figure 2.12. The images taken during this mission were extensively used during the performance evaluations made in this thesis. The dataset is described in section 4.2.2 33

4.2 Example of HPatches images. Above there are two images of the same scene with view-point changes and, below, images exhibit large changes in illumination while maintaing the viewpoint 35

4.3 Example images taken by the downward-looking camera during the Spin.Works UAV Landing Mission at a quarry 36

4.4 Example images from Spin.Works Moon Landing Mission Simulation 37

4.5 Example images from Perseverance Rover's Descent and Touchdown on Mars (Official NASA Video) 38

4.6 SuperPoint Training Overview. First, an initial detector network is trained using synthetic data. Then, that network produces pseudo-ground truth points on MS-COCO images. Finally, SuperPoint is trained using the labels generated before. Adapted from [22] 40

4.7 SuperPoint Architecture. Decoder with VGG-like architecture with 3x3 convolutional layers sized 64-64-64-64-128-128-128-128. Every two layers there is a 2x2 max pooling layer. Detector Decoder with a a single 3x3 convolutional layer of 256 units followed by a 1x1 convolutional layer with 65 units and finally a channel-wise softmax layer. Descriptor Decoder composed by a 3x3 convolutional layer of 256 units followed by a 1x1 convolutional layer with 256 units. All convolution layers in the network are followed by ReLu non-linear activation and BatchNorm normalization. From [22] 40

4.8 Homographic Adaptation. Homographic Adaptation is a form of self-supervision for boosting the geometric consistency of an interest point detector. From [22] 41

5.1 HPatches example (with viewpoint changes) of homography estimation using matches from SuperPoint interest points. Green lines represent matches that are consider inliers on homography estimation. Red points are detections that were not matched or whose matches were considered outliers. 50

5.2 Homography estimation accuracy curves on HPatches. The accuracy represents the ratio between correctly estimated homographies and the total number of image pairs, for re-projection error thresholds ranging from 1 to 10. 50

5.3 Homography estimation accuracy curves on UAV landing dataset from Spin.Works. The accuracy represents the ratio between correctly estimated homographies and the total number of image pairs, for re-projection error thresholds ranging from 1 to 10. 52

5.4 Feature correspondences between frame 1 and frame 81 of the UAV landing video sequence 53

5.5 Feature correspondences between frame 101 and frame 151 of the UAV landing video sequence 54

5.6 Reference navigation data (translations and attitude) for the UAV landing mission dataset from Spin.Works, using NED navigation frame. 54

5.7 UAV landing pose recovery deviations, using homographies calculated directly from navigation data instead of image processing techniques. Frame 1 is used as reference. 55

5.8 UAV landing pose recovery deviations, using homographies calculated from image processing techniques with different features detectors. Frame 20 is used as reference and delta between frames is equal to 1. Inter-frame homography matrices are multiplied to get the transformation of the current frame with respect to the reference frame. 56

5.9 UAV landing pose recovery deviations, using homographies calculated from image processing techniques with different features detectors. Frame 1 is used as reference and delta between frames is equal to 20. Inter-frame homography matrices are multiplied to get the transformation of the current frame with respect to the reference frame. Last estimates cut from both classical algorithms, as Harris and SIFT fail badly on the final transitions. 59

5.10 UAV landing mission - Pose estimation from direct SuperPoint matchings from reference frame (frame 1) until frame 100 60

5.11 Reference navigation data (translations and attitude) for the Moon landing simulation dataset from Spin.Works, using NED navigation frame. 62

5.12 Moon landing simulation pose recovery deviations from reference states, using homographies calculated from image processing techniques with different features detectors. Frame 1 is used as reference and delta between frames is equal to 1. Inter-frame homography matrices are multiplied to get the transformation of the current frame with respect to the reference frame. 63

5.13 Moon landing simulation pose recovery deviations from reference states, using homographies calculated from SuperPoint-cv.cornerSubPix detections. Frame 1 is used as reference and delta between frames is equal to 1. Results should be compared to Figure 5.12 64

5.14 Moon landing simulation pose recovery, using homographies calculated by image processing techniques, with frame 1 as reference and delta between frames equal to 20 . . . 65

5.15 Moon landing simulation pose recovery, using homographies calculated by image processing techniques, with frame 1 as reference and delta between frames equal to 30 . . . 66

5.16 Area reduction from reference frame region seen by the camera 67

5.17 Perseverance landing pose recovery, using homographies calculated by SuperPoint keypoints, with frame 50 as reference and delta between frames equal to 10 68

5.18 Perseverance landing pose recovery, using homographies calculated by image processing techniques, with frame 700 as reference and delta between frames equal to 1 69

5.19 Perseverance landing pose recovery, using homographies calculated by image processing techniques, with frame 700 as reference and delta between frames equal to 10 70

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Networks.

CNN Convolutional Neural Network.

DL Deep Learning.

DLT Direct Linear Transform.

ECEF Earth-centered, Earth-fixed.

EDL Entry, Descent and Landing.

ENU East, North, Up.

ESA European Space Agency.

GPS Global Positioning System.

ML Machine Learning.

NASA National Aeronautics and Space Administration.

NED North, East, Down.

NLLS Non-Linear Least Squares.

NMS Non-maximum Suppression.

RANSAC Random Sample Consensus.

SfM Structure from Motion.

SVD Singular Value Decomposition.

TRN Terrain Relative Navigation.

UAV Unmanned Aerial Vehicle.

Chapter 1

Introduction

1.1 Motivation

In the scope of a recent European Space Agency (ESA) mission[1] led by Spin.Works, extensive flight testing over Mars-representative terrain was carried out to demonstrate real-time embedded vision-based navigation and hazard detection and avoidance algorithms for planetary landing applications. Navigation systems like the Global Positioning System (GPS) are only available on Earth, which means that spacecraft exploring other bodies in space need to estimate their position by different methods. During the Apollo moon landings, astronauts used visual navigation during the final descent to avoid craters and land safely. Nowadays, sensors, algorithms and onboard computing are able to substitute and even outperform that human capabilities and enable safe landings in space.

During the last years, cameras are replacing conventional sensors for entry, descent and landing missions as they are versatile and lightweight, allowing absolute and relative navigation, hazard detection and avoidance, to provide precision and reach safe locations, on the Moon, Mars, and beyond. Also, its natural adaptation capacity to the environment and mimicking the human capacity for detecting hazards, makes vision-based navigation the most promising technology for lunar and planetary landings.

In recent years, Convolutional Neural Networks (CNNs) are replacing classical image processing methods in a vast number of tasks and domains. Deep Learning (DL) algorithms are pushing the boundaries of what is possible in the Computer Vision field and most benchmarks are being surpassed by using these techniques. Space industry is also starting to rely on these networks to solve problems, such as target detection and identification or pose estimation of space targets.

The purpose of this thesis, performed at Instituto Superior Técnico in collaboration with Spin.Works, is to investigate the use of Artificial Intelligence (AI), namely CNNs, which may be applicable to image-based navigation in the scope of planetary landing missions. These methods should ideally achieve the same or better navigation performance than conventional image processing techniques.

[1]Spin.Works S.A., "Avoidance Algorithms Extended development and Realistic Testing (AVERT) activity", under the CCN2 to ESA Contract No. 4000107704/13/NL/HB. (video `https://youtu.be/h27ky9adW4o`) 2019

1.2 Problem Description

Terrain Relative Navigation (TRN) was essential during the recent Entry, Descent and Landing (EDL) of NASA's Perseverance rover and it highly relies on an image-based method[2]. Vision-based relative navigation uses a camera to identify surface features and compare their locations along the frame sequence in order to figure out the relative position and attitude with respect to the ones from previous time instants.

For planetary landing missions, we aim to find a planar surface far from craters or other types of hazards. Besides that, TRN starts a few kilometers above the ground. Therefore, we can rely on the assumption that the surface seen by the camera is planar and the problem can be tackled using homographies. Under these assumptions, homographies give exact or almost exact frame-to-frame transformations.

Given an image of the ground surface taken at a certain time instant, we aim to find the homography with respect to images from previous time steps, in order to estimate the spacecraft's relative position and attitude with respect to previous references.

1.3 Outline of the Approach

The first step is to detect and describe keypoints on two different images of the same scene. Then, we define a distance measure and use those descriptors to find correspondences between the points on both images. The encountered correspondences are used to estimate an homography matrix that represents the transformation between the reference and current image. Finally, that relation is converted into euclidean coordinates and decomposed into relative rotation and translation between the cameras that acquired both images. If we choose a fixed reference frame, we can estimate attitude and translation of the spacecraft along the landing trajectory.

Our focus goes to the detection and description step, where we perform experiments using both classical and deep learning algorithms to compare the navigation estimates and find the advantages of moving into the Machine Learning (ML) approach.

1.4 Contributions

In this Thesis, we perform an in-depth analysis of some popular classical feature detection methods against a machine learning one, for the task of developing a vision-based navigation system applicable to planetary landing missions. We identify the main contributions as follows:

1. We propose a framework for evaluating feature detectors for the task of motion states estimation during a planetary landing mission, applicable to any video sequence dataset, assuming that ground truth positions and attitudes of the spacecraft+camera system are known.

[2]URL: `science.nasa.gov/technology/technology-highlights/terrain-relative-navigation-landing-between-the-hazards`, accessed April 19th, 2021

2. We ran several experiments to compare classical feature detectors and one relying on machine learning with the purpose of getting the most accurate estimates of position and attitude on a landing trajectory.

3. We have shown that it is possible to use deep learning feature detectors, namely SuperPoint, to accurately perform an image-based navigation perception system in landing missions, and that it actually outperforms classical methods such as Harris Corners or SIFT, specifically when estimating relative pose between highly spaced frames.

4. We confirmed the possibility of using simulation datasets to evaluate the vision-based system performance and we have done experiments using the most recent and representative dataset of our problem, the Perseverance Rover's Descent and Touchdown on Mars from NASA.

1.5 Thesis Outline

This thesis is organized as follows. In Chapter 2, we explain the theory behind the whole pipeline of extracting pose information from images of a video sequence. First, we talk about the existent feature detectors and feature tracking and, then, we present the transformations between 2D images, and between 2D images and the 3D world. In Chapter 3, we review the DL concepts and the state-of-the-art approaches for learned feature detectors.

In Chapter 4, we construct our implementation in detail, starting from the datasets used, the feature detection algorithms implemented, the method to reconstruct the motion of the spacecraft from the image correspondences and finally the metrics chosen for evaluations.

Finally, in Chapter 5, we describe our experiments and present the results achieved by the tested algorithms, in representative, simulation and real datasets, and, in Chapter 6, we report our achievements and some possible future work.

Chapter 2

Background

Homographies give image-to-image transformations under some conditions. First, when images are acquired by a camera that only rotates around its centre of projection. Second, when images are taken with large distances to objects and, finally, in the case of planar scenes. Landing of spacecrafts is done in surfaces that are reasonably planar. Therefore, a homography is a good model for what happens to the same 3D point when it is seen from different viewpoints by the aircraft. Then, using homographies is a good approach to recover position and attitude of spacecrafts with respect to the ground in order to build a reliable vision-based navigation system.

Traditional image processing techniques have been used during the past years to solve this problem in a sequential process of detecting important points in images taken in successive instants of time, find a relation between these points in different images and then use these relations to compute the transformation between different viewpoints. These steps and the different traditional computer vision approaches to them are described in this chapter.

2.1 Feature Detection and Description

Cameras project 3D points from the world into 2D points. Therefore, when the same scene is seen by cameras from different perspectives, images contain common information.

The ability to look to different images and find common information, common patterns or specific features that could be easily tracked and compared is present in humans inherently. But how could we define those features and, most important, convert their search to a computer program?

In the past years, engineers have proposed several handcrafted algorithms to detect and describe those points based on heuristics. These traditional methods will be presented in the following sections.

These points, also called keypoints or features, should be detected and described in a way that reliably distinguishes them from other points despite variations in illumination, rotation, and scale. Ideally, they are also distinguishable in the presence of noise and remain consistent despite any transformation.

Common requirements of the applications that require tracking features are robustness to illumination and viewpoint changes and real-time processing capabilities.

2.1.1 Harris Corners & Shi-Tomasi Corners

One of the first and most popular attempts to find those points was done in 1988 in [1]. If we take a flat region in an image, no gradient change is observed in any direction. Similarly, in an edge region, no gradient is observed along the edge direction. Hence, they are bad keypoints since they are not very distinctive, i.e., wherever you move in a neighborhood it looks the same. In the case of corners, we observe a significant gradient change in all directions, which makes them distinctive and invariant to translation, rotation and illumination. That was the idea behind Harris Corner Detector, looking for the regions in images which have maximum variation when moved (by a small amount) in all regions around it. Therefore, a small window around each pixel is considered. Then, we move the window by a small amount in the direction (u, v) and compute the Sum of Squared Difference (SSD) between the intensities in each pixel of the window. Formally,

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \tag{2.1}$$

The window function $w(x, y)$ could be represented by a rectangular window or a Gaussian window which gives different weights to pixels. $I(x, y)$ denotes the intensity value of the pixel located at coordinates (x, y) and $E(u, v)$ is the SSD for a deviation in direction (u, v).

In order to maximize the function $E(u, v)$, one needs to maximize $I(x + u, y + v)$. Using first order Taylor Expansion, we get the following,

$$I(x + u, y + v) \approx I(x, y) + uI_x + vI_y \tag{2.2}$$

where I_x and I_y are the image derivatives in x and y directions, respectively.

Then, we get the function $E(u, v)$ written in the matrix form,

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \quad , where \quad M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_y I_x & I_y I_y \end{bmatrix} \tag{2.3}$$

To measure the corner response at each pixel, a function R is defined by the expression

$$R = det(M) - k(tr(M))^2 \tag{2.4}$$

where k is a parameters that was empirically determined constant in the range $[0.04, 0.06]$. It influences trading off precision and recall. The determinant of M could be computed by the product of its eigenvalues and the trace by their sum. Therefore, when $|R|$ is small, which happens when both eigenvalues are small, the region is flat. If $R < 0$, which happens when one of the eigenvalues is much greater than the other, the region is a edge. Finally, when R is large, which happens when both eigenvalues are large, the region is a corner.

Finally, its results in an array similar to the image with the corner response at each pixel. Using non-maximal suppression the maxima of corner pixels in every local area is found and the rest are suppressed. The decision about considering or not a corner is based on a pre-defined threshold on corner response values.

Later, J. Shi and C. Tomasi proposed a small modification to the Harris Corner detector in [2]. Instead of the corner measure in Harris, they proposed the following scoring function,

$$R = min(\lambda_1, \lambda_2) \tag{2.5}$$

where λ_1 and λ_2 are the eigenvalues of the matrix M. The same way, when R is greater than some threshold, the point is considered a corner. This modification has shown better results with the incremental cost of having to compute the eigenvalues.

Figure 2.1 can represent the differences in the scoring function and the corner classification.

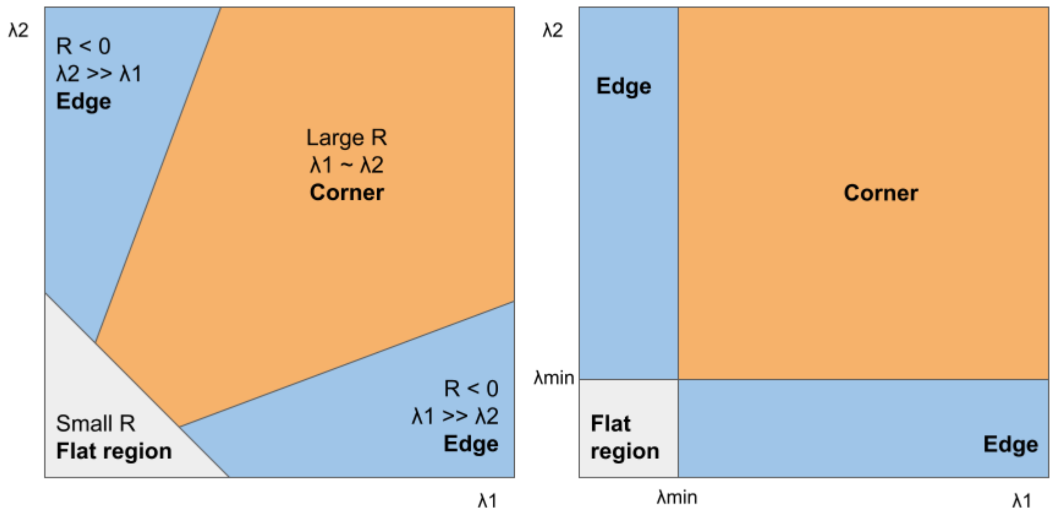


Figure 2.1: Corner classification as a function of the eigenvalues. Left: Harris, Right: Shi-Tomasi

After detecting features, we must assign an identification to each one of them, so that we could search for the same feature in the next frame or in another image which represents the same scene seen by other perspective. For video sequences, simple error metrics, such as Sum of Squared Differences (SSD) or normalized cross-correlation (NCC), can be used to directly compare the intensities in small patches around each feature point, assuming the local motion is mostly translational.

2.1.2 SIFT & SURF

The methods described in section 2.1.1 fail when there are large scale or rotation changes. Then, SIFT was presented in [3] and [4], which stands for Scale-Invariant Feature Transform. SIFT remains one of the most popular and widely used algorithms for feature detection and descriptors.

Points with different scales cannot be detected using the same window. We need larger windows to detect larger corners. For this reason, *D. Lowe* proposed to use Laplacian of Gaussian (LoG) for the image with various σ values, which act as a scaling parameter. Gaussian kernel with low σ gives high value for small corner, while gaussian kernel with high σ fits well for larger corner. This way, it is possible to look for 3D (space+scale) maxima and keypoints are represented by space coordinates and scale factor.

In order to reduce costs, SIFT approximates LoG with Difference of Gaussians, which computes the difference between Gaussian bluring of an image with different σ. The process is done in a pyramid of octaves, generated from downsampling the original image. Each octave's image size is half the previous one, as can be seen in the figure 2.2.

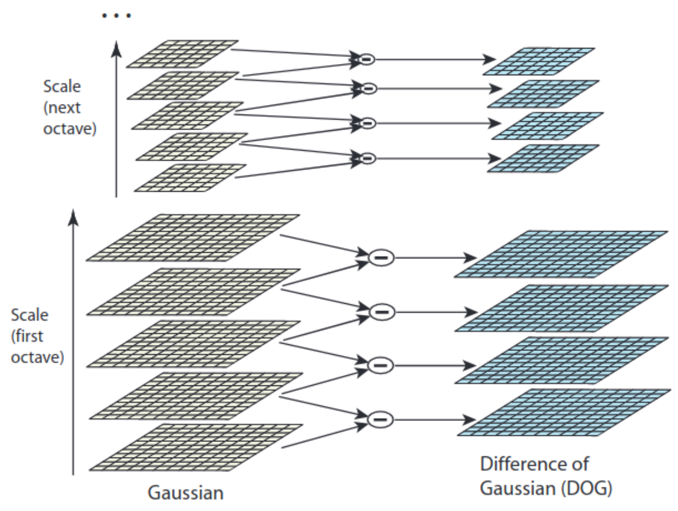


Figure 2.2: For each octave of scale space, the initial image isrepeatedly convolved with Gaussians toproduce the set of scale space images shown on the left. Adjacent Gaussian images are subtractedto produce the difference-of-Gaussian images on the right.After each octave, the Gaussian image isdown-sampled by a factor of 2, and the process repeated. From [4]

After generating the scale space and computing the Difference of Gaussians, pixels are compared with its 8 neighbors as well as 9 pixels in the next scale and 9 pixels in previous scales. If that pixel is a local extrema, it is a potential interest point at that scale. This process of finding potential keypoints is described in the figure 2.3.

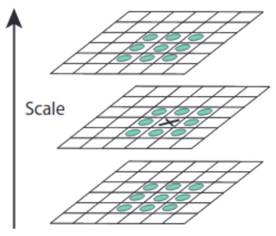


Figure 2.3: Local extrema of the Difference of Gaussian images detected by comparing a pixel to its 26 neighbors in 3x3 regions at the adjacent scales. From [4]

Then, potential features are refined using Taylor series expansion of scale space and then a measure similar to Harris Corner Detector is used to eliminate edges. They rely on the 2x2 Hessian matrix computed at the location and scale of the keypoint.

This way, we guarantee legitimate and stable keypoints with scale invariance. Next, an orientation is assigned to each keypoint to assure rotation invariance. A local neighborhood is taken and the gradient is calculated in that region. An orientation histogram with 36 bins is created covering 360 degrees. Each

10-degree bin gets the magnitude of the gradient in that direction and the highest peak is taken. Other peaks above 80% of the highest peak are considered and keypoints with same location and scale but different orientations are created.

Finally, each keypoint gets its own descriptor vector with 128 values that is highly distinctive and invariant as possible to changes in viewpoint and illumination. The computation of this descriptor is described in figure 2.4

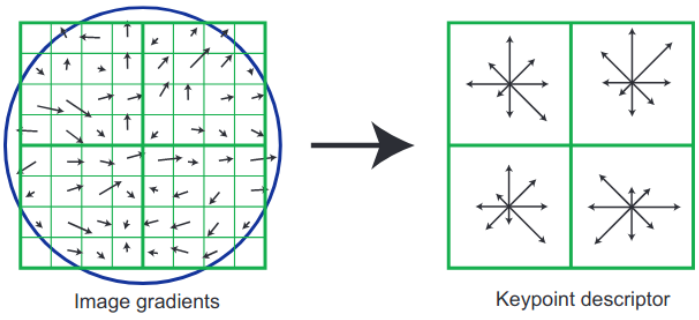


Figure 2.4: Descriptors are created by computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location. These are weighted by a Gaussian kernel window, as indicated by the circle. Then, the samples are accumulated into orientation histograms summarizing the contents over 4x4 regions, with the length of each arrow corresponding to the sum of gradient magnitudes near that direction with the region. Here, we see a 2x2 descriptor array computed from 8x8 set of samples, whereas the experiments in [4] use 4x4 descriptors computed from a 16x16 sample array. From [4]

SIFT has achieved great performance. However it is consider computationally expensive. Hence, a new algorithm was proposed in [5], called SURF, which is a speeded-up version of SIFT.

Its fast computation results from approximating LoG with Box Filter, whose convolution can be efficiently done by integral images. Besides that, SURF relies on determinant of Hessian matrix for both scale and location rather than using a different measure for selecting the location and the scale because of its good performance in computation and accuracy.

The integral image $I_{\sum}(x, y)$ with the same dimensions of the input image I, is computed in such a way that each entry results from the sum of all input image pixels within a rectangular region delimited by the pixel at the coordinates (x, y) and the origin. The Hessian operator can be approximated by convolving the kernels, shown in figure 2.5, resulting in a series of additions and subtractions os sums of rectangular areas.

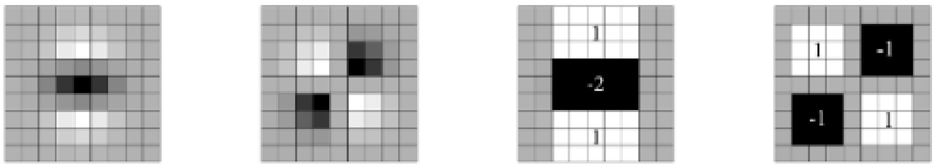


Figure 2.5: Discretised and cropped versions of Gaussian second order partial derivatives in y-direction and xy-direction (left), and the approximations using box filters (right). From [5]

Scale spaces are usually implemented as image pyramids. The images are repeatedly smoothed with a Gaussian filter and subsequently sub-sampled to achieve a higher level of the pyramid. The authors propose, instead, to apply filters with increasing size to the original image at exactly the same speed and even in parallel. Then a non-maximum suppression in a $3 \times 3 \times 3$ neighborhood is applied.

Like SIFT, SURF calculates the orientation of each feature and determines its descriptor vector. The orientation is determined by computing the first order derivatives across a 6σ area, σ being the scale of the point, similar to the one described in SIFT, using Haar wavelets to take advantage of integral images.

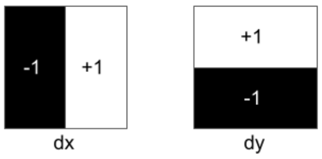


Figure 2.6: Haar wavelets filters

The orientation is determined by "scanning" the weighted scatter plot of d_x and d_y values with a arc of 60 degrees, finding the direction with largest values. To get the descriptor, a $20\sigma \times 20\sigma$ neighborhood is oriented along the orientation vector and divided into 16 cells. Within each cell, a vector with 4 elements is calculated by $(\sum d'_x, \sum d'_y, \sum |d'_x|, \sum |d'_y|)$, result of the Haar kernel convolutions. Then, these vectors are concatenated into a 64-element descriptor vector.

To sum up, SURF improves the speed in every step, becoming 3 times faster than SIFT with a comparable performance. It is good at handling bluring and rotation, but not so good at handling viewpoint and illumination change.

2.1.3 FAST, BRIEF & ORB

SIFT and SURF got good results, but they were patented for several years and people were supposed to pay to use them. The US patent on SIFT held by the University of British Columbia expired as of March 7, 2020.

Considering real-time applications, some feature detectors are not fast enough. So, a new algorithm called FAST was proposed in [6] and later revisited in [7]. FAST, which stands for Features from Accelerated Segment Test, is a feature detection method. The most promising advantage is its computational efficiency. FAST is also one of the first approaches of using machine learning to derive a feature detector and the paper claims that this detector significantly outperforms existing feature detectors in repeatability. Repeatability is one of the most important properties of a feature detector: whether or not the same real-world 3D point is detected in more than one image of the same scene viewed from two different positions.

The method consists of considering a circle of 16 pixels (Fig. 2.7) around the corner candidate p, whose intensity is denoted by I_p. If there exists a set of n contiguous pixels in that circle whose intensity is above I_p plus some threshold t or below $I_p - t$, the pixel p is considered an interest point. In the first version, $n = 12$, which allows a high-speed test to ignore non-corners. First compare pixels 1, 5, 9 and 13 of the circle and at least 3 of these should satisfy the threshold criterion. If it passes this test, then

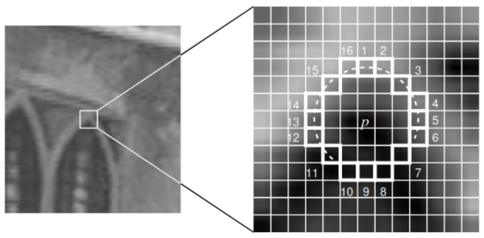


Figure 2.7: Accelerated Segment Test - 16 pixels circle around corner candidate. From [6]

check the other pixels and see if 12 contiguous fall into the criterion. Then, repeat the process for the pixels in the image.

This method has some limitations. For $N < 12$, the algorithm does not work very well due to the number of detected points being very high. Second, the order in which the 16 pixels are tested influences the speed of the algorithm.

In order to overcome these problems, the authors proposed a machine learning approach. First, selecting a set of training images and running the algorithm do detect interest points by taking one pixel at a time and evaluating all the 16 pixels around. For each p in all the images, store the 16 pixels as a vector P. Each value x in this vector can have one of 3 states (d for darker, when the intensity is smaller than $I_p - t$; b for brighter, when the intensity is higher than $I_p + t$ and s for similar, when the intensity is between those values). Then, the ID3 algorithm (decision tree classifier) will query the 16 pixels in such a way that the true class is found (interest point or not) with minimum number of queries, i.e., selecting the pixel x which has the most information about p. The order of querying learned can be used for faster detection in other images.

Detection of several points adjacent to one another is also a problem of the initial version, which can be dealt by applying non maximal suppression. This algorithm is faster than other detectors, but is not robust to high levels of noise. Likewise, there is a growing need for local descriptors that are fast to compute, fast to match, and memory efficient. SIFT uses floating point numbers, for a 128-dimension vector, which takes 512 bytes. Computing descriptors for thousands of features takes lots of memory which is not feasible for resource-constraint applications. Larger memory usage leads to slower matching.

BRIEF descriptor was proposed in [8]. BRIEF converts image patches into a binary feature vector, which means that it only contains ones and zeros. Hence, each keypoint descriptor takes 128-512 bits of memory depending on the dimension of the vector. First, the image patch is smoothed by a Gaussian kernel and then a set of pixel pairs are selected in an unique way. Then, pixel intensity comparisons are done on these pairs and depending on the result, 1 or 0 is assigned to that pair. After doing this for N pairs, we get a N-dimensional descriptor vector. Besides taking low memory, binary strings could also be matched using Hamming Distance, which is just applying XOR and bit count. These operations are very fast in modern CPUs and is another advantage of this approach.

From the "OpenCV Labs", came out ORB in [10]. ORB, which stands for Oriented FAST and Rotated BRIEF, is a fusion of FAST detector and BRIEF descriptor with some modifications on both. First, ORB uses a multi-scale image pyramid to achieve partial scale invariance. The image pyramid consists

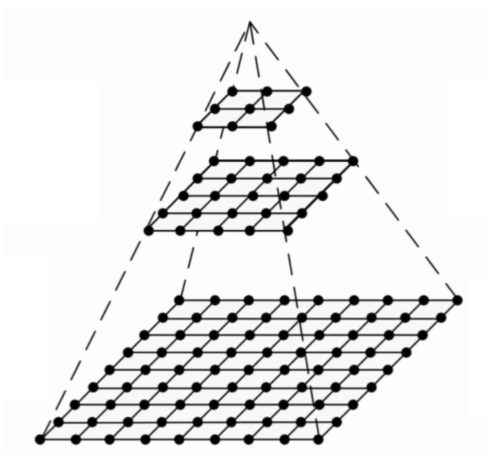


Figure 2.8: Image pyramid. Each level has half the resolution (width and height),and hence a quarter of the pixels, of its parent level. Adapted from [9]

of sequences of images at different resolutions at different levels in a shape of a pyramid (Fig. 2.8. ORB detects points at all these resolutions of the image. After that, ORB assigns orientation to each feature. This orientation is calculated by the direction of the vector from the point to the intensity weighted centroid of the patch. With this modification, ORB aims to achieve rotation invariance.

Since BRIEF fails with rotation, the authors proposed to rotate it according to the orientation computed for each keypoint. The coordinates of the pairs used in BRIEF are concatenated into a matrix which is then rotated to get the steered version of them. These changes made ORB a very good substitute to SIFT and SURF which were patented and computationally more expensive.

A lot more efforts were done and a lot more hand engineered detector and descriptors have been made during the years.

2.2 Feature Tracking

Images of the same scene from different perspectives should have common parts. Therefore, after detecting interest points in each image, one must find which ones are common between them, i.e., the ones that should represent the same 3D world point in order to extract useful information. In a video sequence, feature points can be tracked along frames.

With this purpose, interest point detectors must be robust and repeatable, the algorithm should be able to detect the same features of the same scene under variety of viewing conditions, independent of scaling, shifting, rotation, illumination variations and noise. In the same way, feature descriptors must be discriminative, i.e., descriptors of different regions are different, and invariant, that is, descriptors of the same object part from different images should be similar.

Features can be tracked along a video sequence using one of two methods, described in the next sections.

2.2.1 Optical Flow

Optical flow is the apparent motion of the objects in an image between two consecutive frames caused by the movement of one with respect to the other. It is represented by a 2D vector field with vectors indicating the displacement from one frame to the next. It relies on the assumptions that pixel intensities of an object do not change between consecutive frames and that neighboring pixels have similar motion. Therefore, optical flow methods rely on the minimization of the brightness constancy, which can be written as:

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{2.6}$$

Assuming small displacements, the image intensity can be approximated by a first order Taylor polynomial,

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t \tag{2.7}$$

The equation above results into the Optical Flow equation,

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{2.8}$$

where (u, v) are the components of the velocity, which are unknown and cannot be computed as such. To find the optical flow, another set of equations is needed, given by some additional constraint. Several methods have been proposed to solve this equation.

One of the most popular is proposed in [11], called Lucas-Kanade method. It assumes that the flow in a local neighborhood around the corner is constant and solves the optical flow equation (Eq. 2.8 for each pixel in that patch, by the least squares criterion. This method fails in large motions. The solution to that is again to use pyramids, because as we go up in the pyramid, small motions disappear and large motions become small motions. The vector (u, v) that results from least squares is used to estimate the next position of each interest point.

Other methods are proposed that estimate optical flow for all the points in the image, instead of sparse feature points. This is called dense optical flow and it is significantly more expensive.

Optical flow methods are more suited to image sequences than image pairs from different views and, due to the assumptions made, the main problems are large motions, occlusion, strong illumination changes and changes of the appearance of the objects.

2.2.2 Feature Matching

Feature matching uses the feature descriptors to match features with one another by a nearest neighbor search in the feature space. One keypoint in one image is considered to match another keypoint in the other image if they are close enough.

The most simple method is Brute-Force matching. Every descriptor in the first image is compared to every descriptor in the second image according to some distance measure and the closest one is returned. The first approach is to use the Sum of Squared Differences (SSD) over the entries of the descriptor vectors. Similar to that is to use the L2-Norm, which is simply the square root of the SSD. The Hamming distance can be used with binary descriptor vectors, such as BRIEF.

This process will lead to “incorrect matches” because some descriptors could be ambiguous or because some features that appear in the first image could not be present in the second. One way to overcome this problem is to set a threshold on the distance measure and accept only matches below that threshold. However, the choice of the threshold depend on the application and influences the proportion of “false positives” and “false negatives”.

Another approach to get more “good matches” is to use Lowe’s distance ratio test proposed in [4]. Besides the best match, one must save the second best match too and then it only considers a “good match” the ones whose error of the best match is less than 70% the value of the error of the second best match for a particular pair. The efficiency of this measure relies on the fact that similar features will have both descriptors and than will not be considered a “good match”, as it forces to only accept those whose nearest descriptor is significantly nearer than the second one.

A good alternative to the ratio test is to perform a two-way nearest neighbor[1]. A “good match” is only considered when some descriptor of the second image is the best match for one in the first image and vice-versa. That is, the two features in both sets should match each other, which provides consistent results.

Feature matching algorithms are far better if there is a perspective difference between the images, or the frames, or when the transformations are large, e.g., for a wide interval between frames. They are scale and rotation invariant and are robust to changes in illumination, as those caused by shadow or different contrast.

2.3 3D Motion from 2D Image Transformations

A vision-based system must have the ability to extract 3-dimensional world information from the 2-dimensional images, using the information acquired from the cameras to determine the relations between the objects in the 3D world.

First, one must estimate the transformations between two 2-dimensional images, e.g, between different frames. Then, one must understand the projective relation between the world frame and the image frame itself, in order to transfer the knowledge about image transformations into world coordinates transformations.

2.3.1 2D Image Transformations

Two images of an object from the same camera in different positions are related by a 2D projective transformation. There are several transformations that relate the pixel coordinates from the images, depending on the degrees of freedom, as can be seen in Fig. 2.9.

[1]OpenCV Documentation (*crossCheck* FLAG in `https://docs.opencv.org/3.4/d3/da1/classcv_1_1BFMatcher.html`)

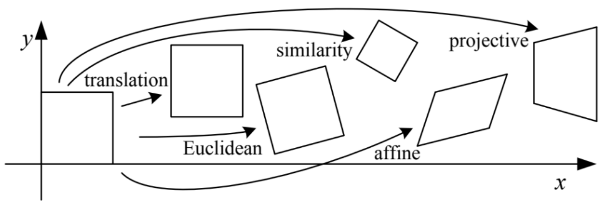


Figure 2.9: 2D Planar Transformations. From from [9]

Complex transformations can be performed by a sequence of simple transformations such as translation, rotation and scaling. Rotation of 2-dimensional vectors can be made by a 2×2 matrix multiplication and translation by adding a 2-dimensional vector. To facilitate this process, we use Homogeneous Coordinates, so that every transformation can be represented by a multiplication of a 3×3 matrix.

Homogeneous coordinates • Any cartesian point can be converted to homogeneous coordinates adding another dimension, w, into existing coordinates. Therefore, a point with pixel coordinates (u, v) can be transformed into homogeneous coordinates, becoming $(u, v, 1)$. The same way, points in homogeneous coordinates can be converted back to cartesian coordinates simply by dividing the first two coordinates by the last one. A point (u, v, w) in homogeneous coordinates becomes $(u/w, v/w)$ in cartesian coordinates. Hence, homogeneous coordinates are scale invariant and allow to represent points at infinity. Formulas involving this system of coordinates are often simpler and more symmetric than those using the Cartesian ones.

Next, several transformations will be presented sequentially increasing the degrees of freedom.

Translation • 2D translation can be represented in Cartesian coordinates as $x' = x + t$ or in Homogeneuous Coordinates as

$$\bar{x}' = \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} \bar{x} \tag{2.9}$$

where $\bar{x}$ represents the point x in homogeneous coordinates. Using the full-rank $3x3$ matrix, it is possible to chain transformations by multiplying matrices and also to compute inverse transformations.

Translations have 2 degrees of freedom and they preserve lengths, angles, orientation, parallelism and straight lines.

Rotation + Translation • This transformation, composed by a rotation followed by a translation, also called 2D Euclidean transformation, can be written as

$$\bar{x}' = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \bar{x} \tag{2.10}$$

where

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{2.11}$$

is an orthonormal rotation matrix with $RR^T = I$ and $|R| = 1$. This transformation introduces a new degree of freedom (θ, that represents the rotation angle), summing up to 3. It preserves the same properties as translation except for the orientation.

Similarity Transform • Also known as scaled rotation, it can be written as

$$\bar{x}' = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix} \bar{x} \tag{2.12}$$

where s represents the scale factor. So, it has 4 degrees of freedom and it preserves parallelism and angles between lines.

Affine • The Affine transformation can be represented as

$$\bar{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \bar{x} \tag{2.13}$$

This transformation has 6 degrees of freedom and parallel lines remain parallel. The angles between lines are not preserved.

Homography • This transformation is also known as a perspective transform or projective transform and it can be written as

$$\bar{x}' = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \bar{x} \tag{2.14}$$

Since the coordinates are written in homogeneous coordinates, this transformation is written up to a scale. The resulting point $\bar{x}'$ must be normalized after the transformation to get the Cartesian coordinates x, that is,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}} \tag{2.15}$$

Matrices that differ only by a scale represent the same transformation. Therefore, often the homography matrix is divided by h_{22} so that the value in that position becomes 1 and one can easily see that this transformation has 8 degrees of freedom. Homographies only preserve straight lines.

All the transformations above can be multiplied by one another to chain transformations, which is of great importance when applied to images. The next table summarizes the most important properties of these transformations.

2.3.2 Projective Homography Estimation

Let $p_i = (u_i, v_i, 1)$ be the vector containing the homogeneous coordinates of a point in the first image and let $p_i' = (u_i', v_i', 1)$ be the homogeneous coordinates of the same point in the second image. The projective homography matrix transforms p_i into p_i', up to a scale factor and can be solved using the Direct Linear Transform (DLT) algorithm, as explained in chapter 4.1 in [12].

Transformation	Matrix	#DoF	Preserves
Translation	$\begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix}$	2	Orientation
Euclidean	$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$	3	Lengths
Similarity	$\begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix}$	4	Angles
Affine	$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix}$	6	Parallelism
Homography	$\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$	8	Straight Lines

Table 2.1: Hierarchy of 2D Transformations. Adapted from [9]

This type of homography can be calculated using the pixel locations of corresponding points in the pair of images. These corresponding points are obtained from “good matches”, as explained in the section 2.2.

Each pair of corresponding points allows to write 2 equations, as stated in Eq. 2.15. Given that homography matrix has 8 unknowns, at least 4 sets of corresponding coplanar points are needed (at least 3 of them must be non-collinear). We now have at least 8 equations that we can stack and create the homogeneous system,

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1u'_1 & -v_1u'_1 & -u'_1 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1v'_1 & -v_1v'_1 & -v'_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2u'_2 & -v_2u'_2 & -u'_2 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2v'_2 & -v_2v'_2 & -v'_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3u'_3 & -v_3u'_3 & -u'_3 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_3v'_3 & -v_3v'_3 & -v'_3 \\ u_4 & v_4 & 1 & 0 & 0 & 0 & -u_4u'_4 & -v_4u'_4 & -u'_4 \\ 0 & 0 & 0 & u_4 & v_4 & 1 & -u_4v'_4 & -v_4v'_4 & -v'_4 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ 1 \end{bmatrix} = 0 \tag{2.16}$$

The matrix A has rank 8, and thus has a 1-dimensional null-space which provides a solution for h, up to a non-zero scale factor.

In general, there are many more than four corresponding points in a pair of images. Each additional pair of points adds two rows to the matrix in 2.16. In this case, the system becomes over-determined.

If all point locations are exact, then the matrix A still has rank 8 an there will be a single homogeneous solution. However, in practice, there is always some uncertainty, the points will not be exact and there will not be an exact solution. In this case, one attempts to find an approximate solution that minimizes a suitable cost, known as the Homogeneous Linear Least Squares problem. It is solved using Singular Value Decomposition (SVD). One takes the singular vector that corresponds to the smallest singular value. This is the solution, h, which contains the coefficients of the homography matrix that best fits the corresponding points.

The problem in this over-constrained case is the presence of outliers that will degrade the solution from the Least Squares. Techniques that reject these outliers must be implemented to get a more accurate estimate of the homography.

2.3.3 Random Sample Consensus

Also known as RANSAC, Random Sample Consensus is an iterative method to estimate the parameters of a model which contains outliers. The algorithm was published in [13]. RANSAC allows more robust estimations since it is able to reject outliers.

RANSAC can be used to estimate homography using the re-projection error as a distance measure to classify corresponding point pairs as inliers or outliers. RANSAC is an iterative procedure which is described in the algorithm 1.

Algorithm 1 RANSAC algorithm for Homography Estimation

```
1: procedure HOMOGRAPHY($src, dst, inlierDistThreshold, inlierRatio, maxIter, confidence$)
2:     $numberIter \leftarrow maxIter$
3:     $iterCount \leftarrow 0$
4:     while $numberIter > iterCount$ do
5:         $x, y, u, v \leftarrow random(4)$      ▷ Randomly select 4 correspondences between src and dst
6:         $h \leftarrow DLT(x, y, u, v)$      ▷ Estimate homography using DLT
7:         $d \leftarrow ReprojDist(src, dst, h)$      ▷ Compute Reprojection Error
8:         $inliers \leftarrow d < inlierDistThreshold$      ▷ Compute inliers
9:         $numInliers \leftarrow sum(inliers)$      ▷ Compute number of inliers
10:        if $numInliers > sum(maxInliers)$ then      ▷ Check if it is the best estimate
11:            $maxInliers \leftarrow inliers$
12:            $finalH \leftarrow h$
13:        end if
14:        if $sum(maxInliers > \#points * inlierRatio$ then      ▷ Inlier ratio higher than suggested
15:            $break$
16:        end if
17:        $numberIter \leftarrow max(IterationsFromConfidence(confidence, outlierRatio), maxIter)$
18:        $iterCount \leftarrow iterCount + 1$
19:    end while
20:    $finalH = DLT(inliers)$
21:    return $finalH, inliers$
22: end procedure
```

Normalization • In chapter 4.4 from [12], the authors presented that the DLT algorithm is dependent on the origin and scale of the coordinate system, which is not desirable for the stability of the algorithm. To solve this problem, they propose to normalize the coordinates of the points by subtracting the mean and dividing by the standard deviation, as represented by the transformations below,

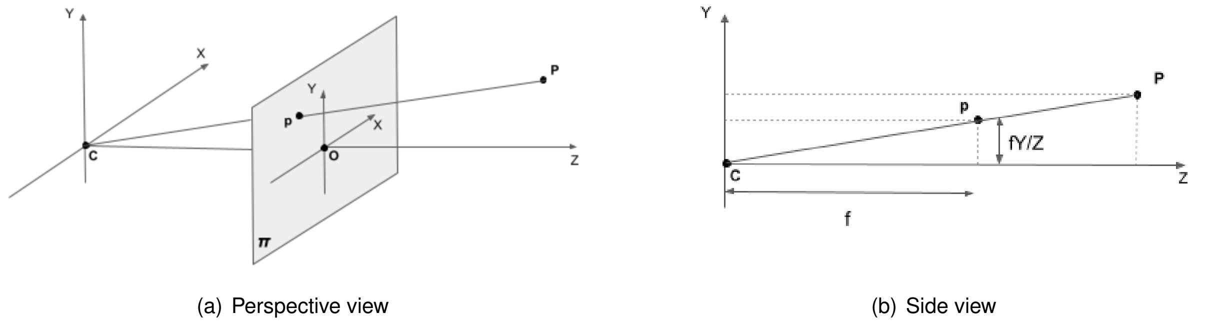
$$\tilde{p}_i = Tp_i \quad \text{and} \quad \tilde{p}'_i = T'p'_i \tag{2.17}$$

where T and T' represent the normalization. Then, estimate the homography $\tilde{H}$ from the normalized points $\tilde{p}_i$ and $\tilde{p}'_i$. Finally, compute the result homography using matrix multiplication,

$$H = T'^{-1}\tilde{H}T \tag{2.18}$$

2.3.4 Pinhole Camera Model

The information from the 3D world is projected into a 2D plane when a camera acquires an image. The simplest way of representing this transformation is the pinhole model, described in Fig. 2.10.



(a) Perspective view (b) Side view

Figure 2.10: Pinhole camera model representation

Pinhole Model • The points in the 3D world are projected to a point, the *camera centre*, represented in Fig. 2.10 by the letter C. One can construct an Euclidean coordinate system with origin in that point. The *image plane*, represented by π, is located in the plane $Z = f$, f being the focal length of the camera. A point in the world $P = (X, Y, Z)$ is mapped into the projection plane to a point $p = (x, y)$, which is the intersection of the projection plane and the projection line that contains the point P and the camera centre, C. As one can see in the side view in Fig. 2.10, similar triangles allow to calculate the coordinates of the projected point,

$$p = (x, y) = \left(f\frac{X}{Z}, f\frac{Y}{Z}\right) \tag{2.19}$$

The line from the camera centre perpendicular to the image plane is called *principal axis* or *principal ray*, and the intersection with the image plane is the *principal point*, represented as O.

Pinhole model using homogeneous coordinates • Once again, homogeneous coordinates can simplify the transformations between points. Here, if the world and image points are represented by homogeneous coordinates, this transformation can be represented by a linear mapping, computed using matrix-vector multiplication as

$$\bar{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.20}$$

which can be described in a more compact way as

$$\bar{p} = diag(f, f, 1)[I|0]\bar{P} \tag{2.21}$$

where $\bar{p}$ and $\bar{P}$ represent the homogeneous coordinates of the point in world and in the camera plane, respectively. In real cameras, this simple approach needs some modifications due to some assumptions made.

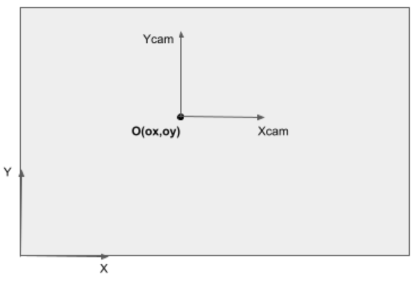


Figure 2.11: Principal point offset. Image and camera coordinate frames

Principal point offset • The first assumption is that the origin of the coordinates coincides with the principal point, which may not be true as can be seen in Fig. 2.11. A more general approach is to take into consideration this offset,

$$p = (x, y) = \left(f\frac{X}{Z} + o_x, f\frac{Y}{Z} + o_y \right) \tag{2.22}$$

where (o_x, o_y) are the coordinates of the principal point in the image frame. This transformation can also be represented by matrix multiplication in homogeneous coordinates,

$$\bar{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} fX + Zo_x \\ fY + Zo_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & o_x & 0 \\ 0 & f & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.23}$$

The same way, can be written in a concise form as,

$$\bar{p} = K[I|0]\bar{P} \tag{2.24}$$

where K is called the *camera calibration matrix* or **intrinsics matrix** and can be written as

$$K = \begin{bmatrix} f & 0 & o_x \\ 0 & f & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.25}$$

The world point P is represented in the *camera coordinate frame*, whose origin is the principal point and Z axis coincides with the principal axis. This coordinates frame is partially represented in Fig. 2.11 as x_{cam} and y_{cam}.

Digital cameras • The pinhole model described until now assumes that the image coordinates are Euclidean coordinates having equal scales in both axial directions. However, digital cameras acquire images forming an array of pixels. Therefore, one must take into account the quantization process and there is the possibility of having different scale factors in each direction. To measure the image coordinates in pixels, one must multiply each coordinate by the number of pixels per unit distance in both directions, which can be represented as m_x and m_y.

The intrinsics matrix needs to be redefined as,

$$K = \begin{bmatrix} \alpha_x & 0 & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

where $\alpha_x = fm_x$ and $\alpha_y = fm_y$ represent the focal length of the camera in pixel dimensions. Similarly, the principal point needs to be converted to pixels, using $u_0 = \alpha_x o_x$ and $v_0 = \alpha_y o_y$.

Sometimes, x and y axis are not perpendicular, the pixel is not rectangular, which introduces a new parameter, the *skew* factor s and the intrinsic matrix becomes,

$$K = \begin{bmatrix} \alpha_x & s & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.27}$$

Lens distortions • The model described above obey a linear projection model where straight lines in the world remain straight lines in the image, as a consequence of linear matrix operations. However, in many wide-angle lenses a curvature appears in the projection of straight lines. This effect is named *radial distortion* and needs to be compensated when using image processing techniques in order to get more accurate results in 3D reconstructions.

2.3.5 Euclidean Homography

World points can be represented with respect to different frames. One can convert the representation of a point from a coordinate frame to another by applying rotation and translation, as the equation suggests,

$$P_2 = RP_1 + T \tag{2.28}$$

where P_1 and P_2 represent the coordinates of a point P in the world according to frame 1 and frame 2, respectively. $R = R_1^2$ is the rotation matrix from frame 1 to frame 2 and $T = t_{2\to 1}^2$ is the translation vector of the frame 2 to frame 1 represented in the frame 2.

When all points lie on a plane, as in Fig. 2.12, we have another constraint,

$$N^T P_1 = n_x X + n_y Y + n_z Z = d \tag{2.29}$$

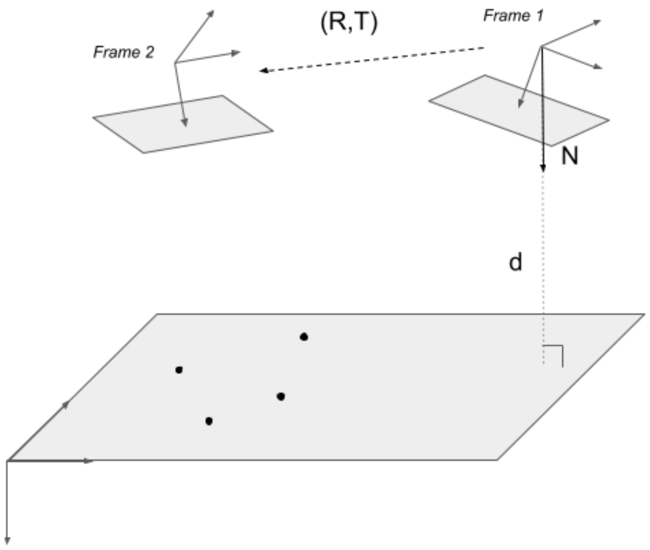


Figure 2.12: Representation of a 2D plane in the world and respective notation

where N is the unit normal of the plane represented in frame 1 and d is the distance from frame 1 origin to the plane along the direction of N. The equation above can be expressed in a more compact way as,

$$\frac{1}{d}N^T P_1 = 1 \quad \text{for points} P_1 \text{in the plane} \tag{2.30}$$

Substituting Eq. 2.30 in Eq. 2.28, the transformation becomes,

$$P_2 = RP_1 + T\frac{1}{d}N^T P_1 = HP_1 \tag{2.31}$$

where

$$H = R + \frac{1}{d}TN^T, \quad H \in \mathbb{R}^{3x3} \tag{2.32}$$

H is known as *Euclidean Homography* or *planar homography matrix*. If the camera is placed at the origin of each frame in different instants of time, the euclidean homography allows to compute the relative pose between the two cameras, or between the same camera at different instants of time.

Points in the world represented in camera frame can be converted up to a scale into the image frame using the camera matrix described in section 2.3.4 by the equations:

$$\bar{p}_1 = \alpha_1 K_1 \bar{P}_1 \quad \text{and} \quad \bar{p}_2 = \alpha_2 K_2 \bar{P}_2 \tag{2.33}$$

where $\bar{p}_i = (u_i, v_i, 1)$ are the homogeneous pixel coordinates of image plane from camera i and $\bar{P}_i = (X_i, Y_i, Z_i, 1)$ are the homogeneous coordinates of the world point in the reference frame of camera i. α_i are constants.

Assuming the same camera in both frames, $K = K_1 = K_2$, and we can derive the relation between the homogeneous pixel coordinates of common points in both images.

$$\bar{p}_1 = \gamma K \left(R + \frac{1}{d} T N^T \right) K^{-1} \bar{p}_2 \tag{2.34}$$

where $\gamma = \dfrac{z_1}{z_2}$ is the scale factor and z_1 and z_2 are the z coordinates of P in each camera frame.

Now, we can relate the projective homography (described in section 2.3.2 estimated by the image processing techniques with the euclidean homography,

$$H_{proj} = \gamma K H_{euc} K^{-1} \tag{2.35}$$

Therefore, the camera motion affects the projective homography in a predictable manner and the euclidean homography gives a physical meaning to the homography. Then, the homography estimated using feature matching and DLT algorithm gives information about the poses between the camera in different instants of time, and consequently, about its motion.

Chapter 3

Deep Learning

Artificial Intelligence has been witnessing a considerable growth in bridging the gap between the human capabilities and machines. Machine Learning is the study field responsible for the AI systems to learn and improve from previous experience and data, without or with little explicit human interference. ML algorithms acquire data and build models that represent this data and specially, that generalize well to new entries. In [14], the author defines a learning problem as a computer program that learn from experience E, with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Machine learning algorithms are classified into three categories: **supervised learning**, **unsupervised learning** and **reinforcement learning**. Supervised learning needs datasets with labels to find a deterministic function that maps future input observations to predictions. Unsupervised learning systems do not use labeled datasets. They investigate similarities between input pairs of objects and derive some structure. Reinforcement learning do not have a fixed dataset, but a feedback loop between the system and its experiences. The final goal is to map situations from the environment to actions with the objective of maximizing rewards.

Looking at the recent literature, one can note that a big focus is being oriented towards deep learning. Deep learning (DL) is a very powerful framework, as deep models appeared as an alternative to linear models. They are able to deal with more complex representations of the data with the introduction of more layers, more units within a layer and non-linear functions. Although the concepts of DL with neural networks (section 3.1) have long existed, recent advances in computation and research enabled these techniques to match or exceed state-of-the-art in many problems. Deep models efficiently derive the function that maps inputs to outputs, due to **backpropagation** algorithm, described in section 3.2.

The advancements in Computer Vision with Deep Learning arise due to a particular type of architectures - **Convolutional Neural Networks**, described in section 3.3. The problem of detecting feature points in images can also be tackled with deep architectures, section 3.4.

3.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are the core architecture in DL. As described in [15], **feedforward neural networks** aim to approximate some function f^* that maps an input x to an output y, $y = f^*(x)$. This network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

Neural networks are composed of several *perceptrons* or *neurons*, which are the most basic component. It consists of weighted sum of all the inputs and some scalar (*bias*) and then some function, called *activation function*, is applied to this sum in order to achieve non-linearity.

$$y = f_{act}(w^T x + b) \tag{3.1}$$

where y represents the scalar output of one neuron, x represents the vector of inputs to that neuron, w is the vector of weights, b is the bias term and f_{act} is the activation function. Non-linear activation functions are necessary to add the complexity required for solving otherwise intractable problems.

Nowadays, the most common activation function is the Rectified Linear Unit (ReLU), expressed as:

$$f_{relu}(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases} \tag{3.2}$$

This function is simple to implement, fast to compute, and avoids diminished gradients as operations are chained together, while achieving the intended non-linear complexity.

Artificial Neural Networks are designed like the human brain, with neuron nodes interconnected with each other. They are called networks due to their representation by composing together many functions. They are organized by layers of neurons, where the outputs of layer $i - 1$ act as the inputs to layer i. Then, all process can be done by matrix multiplications, getting a vector of values in each layer, instead of the scalar y in equation 3.1. The first layer is the vector of inputs to the model, called **input layer** and the final layer of a feedforward network is called **output layer**. The layers in between are the **hidden layers**. The number of layers, or the length of the chain of functions, gives the **depth** of the model and the dimensionality of the hidden layers gives the **width** of the model.

Deep architectures are not restricted to feedforward neural networks where the information flows from the input layer to the output one, passing through the computations in the hidden layers. Some models include feedback connections from the output to the input. These are called **Recurrent Neural Networks (RNNs)**, which are used in temporal analysis applications.

The weights of the network are learned during the training process, which is composed of two phases. The first one is the forward pass, where each input instance is passed through the ANN and the result is compared to the desired output using some cost function. Then, it comes the backward pass, where the error computed by the cost function is propagated back to the network, using gradients, to update the weights. This process is described in the next section. To sum up, the training process can be defined as an optimization problem, whose goal is to minimize the cost function.

3.2 Backpropagation

The non-linearity of ANNs causes loss functions to become non-convex. Therefore, they are trained by an iterative process, using gradient-based optimizers to drive the cost function to a very low value, rather than linear equation solvers used in linear regression models or the convex optimization algorithms. There are several algorithms to compute this optimization, but all rely on the ideas of gradient descent.

Gradient-based optimization algorithms require the computation of the gradient of the loss function with respect to each parameter of the network, in order to update its value. This can be done using the backpropagation algorithm. The gradient with respect to the final layer is calculated first and then partial computations of the gradient are reused in the computation of the gradient for the previous layer until it gets to the first layer. This backwards flow of the error information relies on the chain rule of calculus by computing products between Jacobians and gradients, using matrix-vector multiplications for each layer and neuron of the network. After being propagated back to all the parameters in the network, the gradient of the error function is used to update the weights and biases at each iteration of the gradient descent, according to

$$\theta^{k+1} = \theta^k - \alpha \frac{\partial L(f(X;\theta), y)}{\partial \theta} \tag{3.3}$$

where θ represents some parameter of the network, k is the iteration of gradient descent and $L(f(X;\theta), y)$ is the loss function computed from the prediction of the network with that parameter and the ground truth.

Backpropagation makes the use of gradient methods for training neural networks feasible. It is considered a very efficient algorithm, whose implementation can take advantage of parallel computations in GPUs to further improve the performance.

3.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are currently the most prominent type of architectures for deep learning applications in image data. Whereas for traditional machine learning algorithms relevant image features have to be extracted manually, deep learning uses the entire image as input and learns the essential features needed for the application.

This group of networks was inspired by the organization of the Visual Cortex in biological processes. Individual neurons respond to stimuli only in a restricted region of the visual field. The visual fields of different neurons partially overlap such that they cover the entire visual area.

Another motivation to replace fully connected networks, described in section 3.1, is to avoid the massive amount of weights and computation to process 2D images. A small image of dimension 128×128 would need $16,384$ weights and would take that enormous amount of multiplications to calculate the activation of each single neuron of the first hidden layer. That would lead to very large networks that take extremely long time to train, require excessive amounts of memory and also can suffer from *overfitting*. Overfitting appears due to the massive amount of parameters that make the model extremely complex and fitted to the training data, but that fails with new data, which leads to poor generalization ability.

In [16], one of the first CNNs, that propelled the field of Deep Learning, the author explains that fully connected layers should not be used as first layers, because images are highly spatially correlated, and using individual pixels as separate features do not take advantage of these correlations. **Convolutional layers** are the major building block in CNNs. Convolution is the operation of applying a filter to an input image to extract certain features from it. These filter are also called *kernels* and they are matrices that slide across an image and are multiplied by each single patch of the image, getting some activation that is higher when the patch has properties similar to those of the filter. Repeated application of the same filter to the entire image results in a map of activations called *feature map*. Since these filters are substantially smaller than the entire image, the number of weights decreases significantly.

The advantage of CNNs is the ability to learn a large number of filters in parallel, which are specific to the training dataset and the type of problem. In traditional algorithms, these kernels were hand-engineered. This makes CNNs independent from prior knowledge and human effort in feature design and results in highly specific features that can be detected anywhere on input image. Consequently, these networks are **space invariant**, which means that the filter is able to detect some object independently of its position in the image.

The size of each convolutional filter specifies the receptive field of the filter, i.e., the number of neighborhood pixels that are taken into account to compute the activation in a specific location. Typically, kernels have dimensions 3×3, 5×5 or 7×7. Since an RGB image is a 3D volume, these filters become also 3D tensors, taking into account the *depth* of the input. Tensors are multi-dimensional arrays. For example, one kernel with receptive field 3×3 to be applied to a tensor with depth D_i, gets the shape $3 \times 3 \times D_i$.

Each convolutional layer is composed of several (D_o) of those filters, each one applied to an input 3D volume of dimensions $W_i \times H_i \times D_i$, and producing the respective feature map, resulting in a 3D volume of dimensions $W_o \times H_o \times D_o$. W_i and H_i are the width and height of the input tensor to the layer and W_o and H_o are the width and height of the output tensor, or feature map of the layer. Therefore, the number of weights for this layer becomes $K \times K \times D_i \times D_0$, where K represent the size of the kernels.

Similar to the other neural networks, CNNs also have activation functions to introduce non-linearity. They are applied to the feature maps, which are the result of the convolution layers. **Pooling layers** are also introduced in CNNs. They are responsible for dimensionality reduction in the feature maps, leading to a decrease in computational power required to process the data. This nonlinear downsampling also helps to extract high level features. Pooling can be done by returning the maximum value in the patch covered by the kernel, which is called *max polling*, or by averaging the values in that patch, *average pooling*. In some applications, such as image classification, fully connected layers are also introduced at the end of the network.

CNN architectures vary in the number and type of layers implemented according to the specific application. However, in general, they are composed of alternately stacked convolutional layers and pooling layers. Pooling layers contribute to the downsampling task so that models become less complex and more computationally efficient. Convolution layers are responsible for feature extraction. First layers extract low level features, such as, corners, edges, color or gradient orientation. As one goes deep in the network, the layers start to combine those low-level features and capturing high-level ones, such as polygons, faces or objects. In figure 3.1, one can see the architecture of LeNet-5, one of the first to use convolution layers, back in 1998.

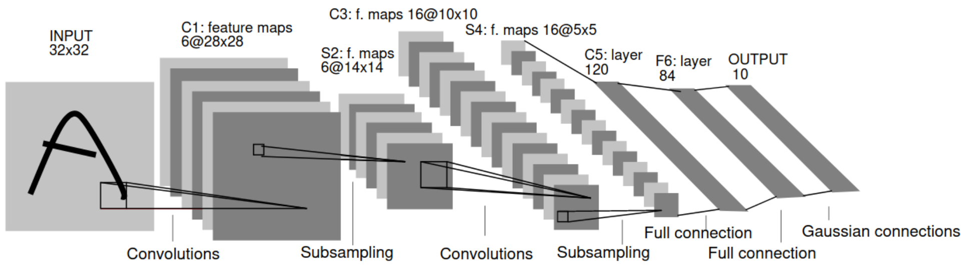


Figure 3.1: Architecture of LeNet-5. From [16]

Between 1998 and 2010, neural networks were in incubation. However, the increasing data available, the increasing CPU power and GPUs becoming a general-purpose computing tool allowed the neural network progress in Computer Vision. Since then, several architectures have been proposed every year and the tasks tackled by CNNs are becoming more and more interesting.

3.4 Deep Interest Point Detection

As mentioned in section 2.1.3, FAST introduced the idea of using machine learning for interest point detection. The success of convolutional neural networks in general object detection and other computer vision problems motivated research community to explore the performance of these networks for the keypoint detection task.

To recap the feature detection problem, there are two types of keypoints in common use in computer vision. **Semantic keypoints** are interest points with semantic meaning for objects in the image, such as the left shoulder of a person, the back left tire hub of a car or the left tip of a mouth. Deep learning has dominated state-of-the-art semantic keypoint detection. These algorithms are supervised learning and require extensive and expensive human annotation to create datasets labelling this semantic keypoints.

In order to build CNNs for the semantic features extraction process, a stacked hourglass architecture has been proposed in [17] and [18], and other architectures such as the U-net [19], or variants of the hourglass have been tested in recent years.

The Stacked Hourglass Network can learn a generalized pattern of human poses, and predict joints location accordingly. It was first introduced in 2016 in [17] to recognize human poses and it is still one of the most important networks in pose estimation area, and widely used in several applications.

Hourglass modules downsample the images and then upsample them to the initial size, allowing to capture and consolidate information across all scales. In the final layer, they produce heatmaps. Human pose data has lots of variances, which makes it hard to converge if one just simply regress the joint coordinates. Heatmaps allow the network to express its confidence over a region rather than regressing

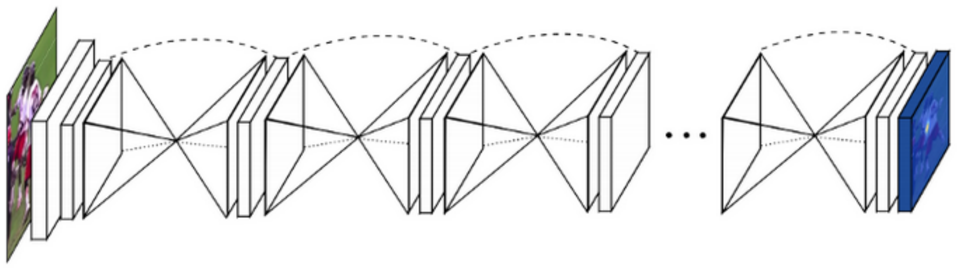


Figure 3.2: Stacked Hourglass Architecture. From [17]

a single (x, y) position for a keypoint. The dataset must be labelled with the coordinates of a fixed number of semantic keypoints, e.g., left ankle or right shoulder of a person.

In [18], the authors also use a stacked hourglass architecture, but to estimate the 6-DoF object pose. The model is trained with objects from several classes. In each class, keypoints are manually defined on 3D models and projected to the images yiealding ground-truth keypoint locations in 2D for training the network.

The employment of convolutional neural networks for monocular pose estimation in space using semantic keypoints has already become an attractive solution in recent years. One of the main advantages of CNNs over traditional feature-based methods for relative pose estimation is the increase in robustness under adverse illumination condition, as well as a reduction in the computational complexity. In [20], the authors propose a network for spacecraft pose estimation, also based on the detection of semantic keypoints. These networks also have the advantage that the trainable features are selected offline prior to the training, so the matching of the extracted feature points with the features of the wireframe model can be made with no need of a long search over the image-model correspondences, which normally characterizes most of the edges/corners-based algorithms. The authors introduced the idea of not only using the heatmap's peak location into the pose estimation solver, but also the statistical distribution around the peak to allow reliable covariances and a robust navigation performance. The network is trained with the image coordinates of feature points, computed offline based on camera intrinsics and feature coordinates in the target body frame, which are extracted from 3D models prior to training. During training, the network is optimized to locate a fixed number of manually chosen features of the spacecraft in question.

In many other applications, such as UAV vision-based navigation, the terrain surface does not have well defined objects and the keypoints have no semantic meaning, thus the need of another type of keypoints arises. **Interest points** are low-level points that usually do not have clear semantic meaning, such as corners. When compared to semantic tasks, such as those explained above, the notion of interest point detection is ill-defined and thus human annotator cannot reliably and repeatedly identify the same set of interest points to create datasets. Therefore, it is non-trivial to formulate the task of interest point detection as a supervised learning problem.

Consequently, the task of detecting interest points using CNNs must rely on **self-supervised learning** or **unsupervised learning**. Unlabeled data is being generated all the time. One must try to make use of this much larger amount of unlabeled data, setting the learning objectives properly so as to get supervision from data itself. The basic idea of self-supervised learning is to get labels automatically for the unlabeled data and train unsupervised dataset in a supervised manner.

During the last years, several approaches were proposed to tackle this task. The focus is to the ones that jointly learn the detector and descriptor. In 2016, a novel deep network, called **LIFT**, was introduced in [21]. It implements the full feature point pipeline, that is, detection, orientation estimation and feature description. Their architecture relies on three CNN-based components that feed into each other: Detector, Orientation Estimator and Descriptor. All these components come from previous articles and are shown to perform well in their individual functions. The authors mesh them together using Spatial Transformers to create and end-to-end, differentiable network. They found impossible to learn the full architecture from scratch and introduced a problem-specific learning approach that involves learning the descriptor component first, then the orientation estimator and finally using both to train the detector. LIFT stays close to the traditional patch-based detect then describe recipe and requires supervision from a classical Structure-from-Motion system.

SuperPoint [22] comes from a self-supervised framework for training interest point detectors and descriptors. The authors proposed a fully-convolutional neural network architecture that operates on a full-sized image and produces interest point detection and fixed length descriptors in a single forward pass, as opposed to patch-based methods. Most of the network's parameters are shared between the two tasks, which differs from traditional tasks and contributes to an efficient architecture that can be used in real-time applications. The paper presents a self-supervised solution using self-training. They prove that it is possible to transfer knowledge from a synthetic dataset onto real-world images. The pipeline starts by creating a synthetic dataset of simple geometric shapes with locations of interest points locations. Then, a simpler network (using only the detector part) is trained on those synthetic images and it is called *MagicPoint*. Since MagicPoint misses many potential interest point locations in real images when compared to classical interest point detector, the authors came with a multi-scale, multi-transform technique called *Homographic Adaptation*. This approach boosts interest point detection repeatability and performs cross-domain adaptation. It is responsible for the self-supervised learning, as it warps the input images multiple times to help the interest point to see the image from many different viewpoints and scales. Using Homographic Adaptation together with MagicPoint detector allows to generate pseudo-ground truth interest points that are more repeatable in real images, supervised by the interest point itself, rather than a large-scale human annotation effort. Finally, the whole network, now including a descriptor subnetwork, is trained on this self-annotated dataset, resulting in the *SuperPoint*. This system works well for geometric computer vision matching tasks and gives rise to state-of-the-art homography estimation results on *HPatches* [23] when compared to LIFT, SIFT and ORB.

LF-Net article [24] proposed a novel deep architecture and a training strategy to learn a local feature pipeline from scratch, using images without the need for human supervision. The authors use image pairs for which they know the relative pose and corresponding depth maps. It is proposed to create a virtual target response for the network, using the ground truth geometry. Specifically, they run the detector on the first image, find maximum response locations and optimize the network parameters so that when run on the second image it produces a response map with sharp maxima at the right locations. The points are warped using the ground truth, which guarantees a large pool of ground truth matches to get the descriptors. For training, they divide the problem into two branches, each containing identical copies of the network. One is used to generate supervision, created by using ground truth warpings (non-differentiable) and the other is used to optimize the network. They use a dataset containing video sequences and LF-Net performs worse than SuperPoint for large frame differences.

Key.Net [25] uses a combination of handcrafted and learned CNN features to produce a keypoint detector. They also propose a novel multi-scale loss and operator for detecting and ranking stable keypoints across scales and a multi-scale feature detection with shallow architecture. Key.Net produces only keypoint locations, it does not describe them for future matching.

D2-Net [26] is a close approach to SuperPoint as it also shares a deep representation between detection and description. However, here the network shares all the parameters between detection and description and uses a joint formulation that simultaneously optimize for both tasks. The authors propose to postpone the detection to a later stage, which make keypoints more stable. The network is trained using pixel correspondences extracted from readily available large-scale SfM reconstructions, without any further annotations and it adresses the problem of finding reliable pixel-level correspondences under difficult imaging conditions. The method performs worse than SuperPoint for stricter matching thresholds on HPatches, because the latter uses detectors firing at low-level blob-like structures, which are inherently better localized than the higher-level features used by this approach.

R2D2 [27] introduced the idea of reliability apart from repeatability. The authors argue that salient regions are not necessarily discriminative, and so can harm the performance of the description. Furthermore, they claim that descriptors should be learned only in regions for which matching can be performed with high confidence, in order to avoid ambiguous areas. The article contributes with novel unsupervised losses to learn keypoint detector and descriptor that are both repeatable and reliable. At test time, they run the trained network multiple times on the input image at different scales and keep a shortlist of the best descriptors over all scales.

The problem of efficiently and accurately detect and describe interest points using CNNs for higher level computer vision applications is still an open field of research these days.

Chapter 4

Implementation

In this Thesis, we evaluate the performance of several image processing algorithms and we aim at proposing a more accurate monocular vision-based relative navigation system, using an homography-based approach.

The monocular downward-looking camera takes an image sequence of the ground scene during flight (Figure 4.1). Since the distance to the ground is high, the ground surface is assumed to be planar and the landing approach mission can be tackled using the homography approach under this assumption.

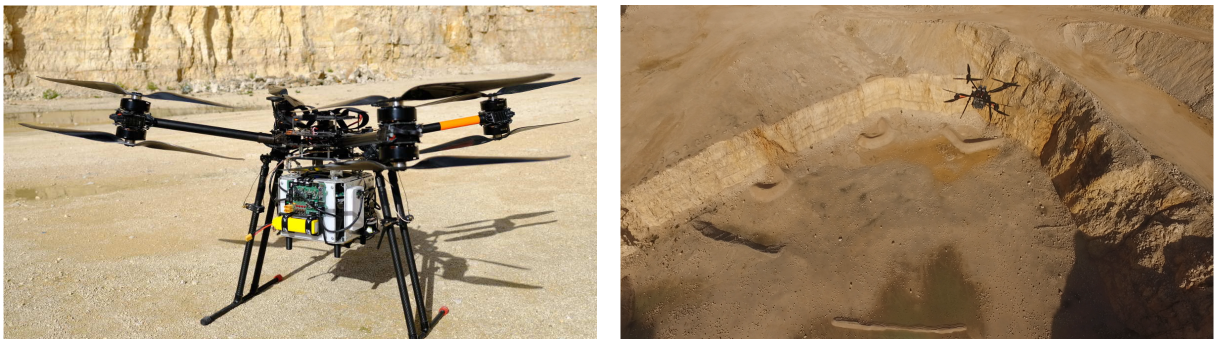


Figure 4.1: Spin.Works UAV from one landing mission on a quarry. On the left, we see the UAV with a monocular camera mounted underneath the electronic systems, pointing to the ground. On the right, we see it flying over a surface that looks similar to Mars, using a vision-based navigation system. The geometry of the problem is depicted on Figure 2.12. The images taken during this mission were extensively used during the performance evaluations made in this thesis. The dataset is described in section 4.2.2

Our focus gets to the feature detection algorithms and the improvements gained by substituting classical approaches to the ones involving deep learning. Therefore, all our implementations and evaluations use two classical algorithms as reference and one using AI for comparison tests. The reference algorithms chosen were Harris Corners and SIFT. Harris Corners is the first feature detection algorithm and still used in industry, e.g., in the solution proposed by Spin.Works to the problem tackled by this Thesis. SIFT is one of the most popular methods due to its high accuracy even when competing with learning approaches. The algorithm chosen to test the performance of DL in feature detection was SuperPoint since the results demonstrated by the paper appear to be successful and it outperforms the competitors in homography estimation metrics.

In this chapter, we present the whole pipeline built to perform evaluations. In section 4.1 we present the software tools adopted, then in 4.2, the datasets are described. In sections 4.3 and 4.4, we define the implementation of the feature detection algorithms and the homography-based motion reconstruction, respectively. Finally, in section 4.5, the evaluation metrics are outlined.

4.1 Software Tools

In order to solve the problem addressed by this Thesis, a software pipeline was implemented in *Python* language to represent to whole process of using camera information to estimate the motion of the aircraft in landing missions.

Since we are working with images and many matrix calculations are needed, the implementation makes use of several Python libraries that are optimized for real-time computer vision and image processing applications such as OpenCV[1] and Numpy[2], and that are optimized for neural networks computations in both CPU and GPU, such as PyTorch[3].

Numpy is an open source Python library used for working with arrays, linear algebra, and matrices. The operations are vectorized, which describes the absence of any explicit looping or indexing in the code. These things are ”behind the scenes” in optimized, pre-compiled C code, which makes Numpy a lot faster than regular Python operations. In this implementation, all mathematical operations are made using Numpy.

OpenCV is a cross-platform library used to develop real-time computer vision applications. It mainly focuses on image processing, video capture and operations for feature detection, object detection and tracking and other optimized state-of-the-art computer vision and machine learning algorithms. We make use of it for reading, resizing and writing images and to implement classical feature detectors, such as Harris Corner and SIFT.

PyTorch is a library that facilitates building and training deep learning projects. It is very similar to Numpy but with strong GPU acceleration. It was the chosen framework to train and apply DL algorithms to the role of feature detection.

Matplotlib[4] is a plotting library for Python. Matplotlib's collection of functions, called *pyplot*, was the API chosen to plot the results in this implementation. It makes matplotlib work like MATLAB.

4.2 Evaluation Datasets

4.2.1 HPatches

One of the most popular benchmarks for evaluating local image descriptors is HPatches [23]. The dataset contains 116 scenes with 696 unique images and it is divided into two groups. The first 57

[1]`https://opencv.org/`
[2]`https://numpy.org/`
[3]`https://pytorch.org/`
[4]`https://matplotlib.org/`

scenes exhibit large changes in illumination while maintaining the viewpoint and the other 59 scenes have large viewpoint changes. Each scene is composed by 6 images, being the first considered the reference, and 5 homography matrices that represent the transformation between the reference image and the respective one. Examples are shown in Figure 4.2.

The images are used by the tested detection algorithms and the homography matrices are the ground truth needed to evaluate their performance in metrics such as repeatability of keypoints or homography estimation.

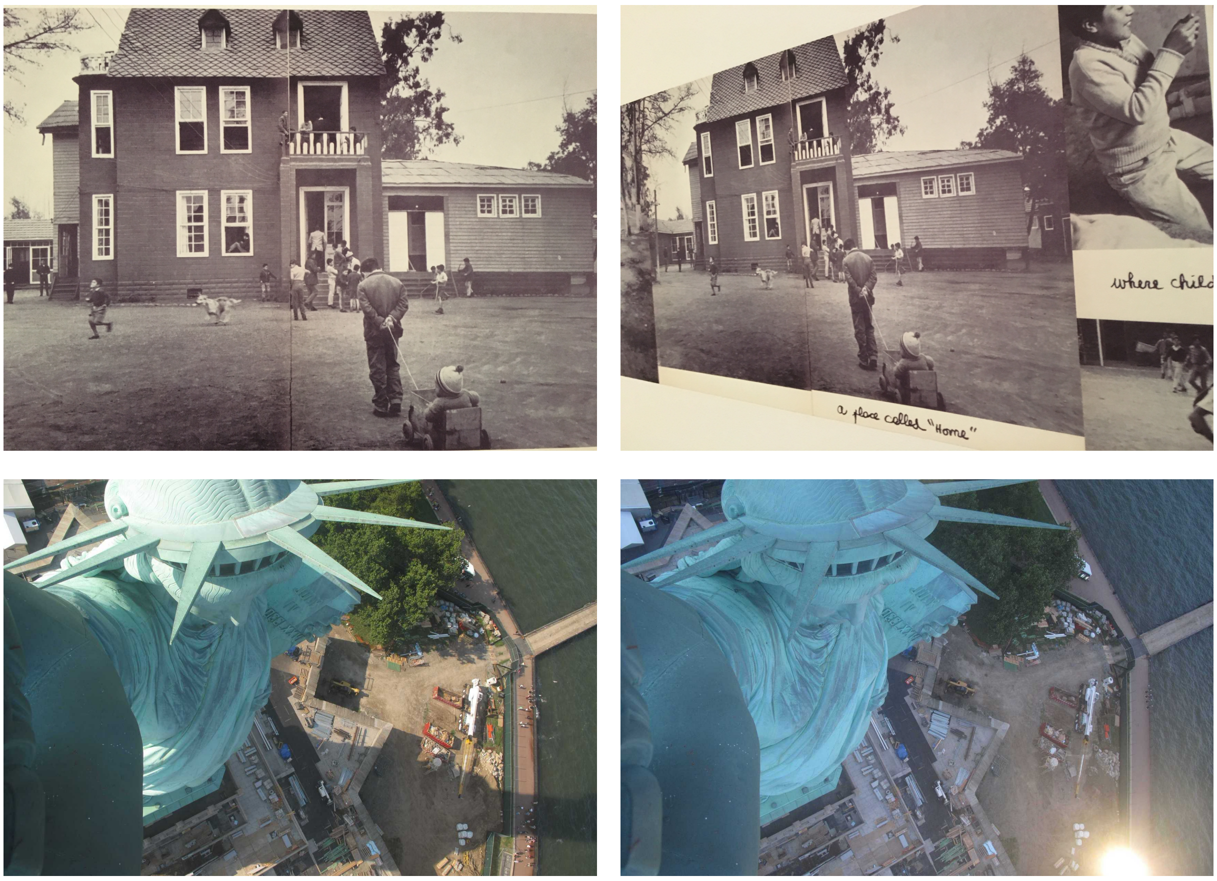


Figure 4.2: Example of HPatches images. Above there are two images of the same scene with viewpoint changes and, below, images exhibit large changes in illumination while maintaing the viewpoint

4.2.2 Spin.Works UAV Landing Mission Dataset

Since the motivation of our work is to evaluate a vision-based navigation framework for lunar and planetary landing missions, Spin.Works ran several experiments of landing missions using UAVs in representative terrain surfaces such as a quarry.

From one of those experiments resulted a video that was highly used during performance evaluations in this Thesis. This video sequence has a total of 384 frames, similar to those in Figure 4.3, acquired by a camera attached to the UAV looking downwards during the landing phase in a quarry.

Spin.Works used some software to perform Structure from Motion (SfM), which uses the first 249 frames to compute the orthophotograph of the terrain surface, the position and attitude of the cam-

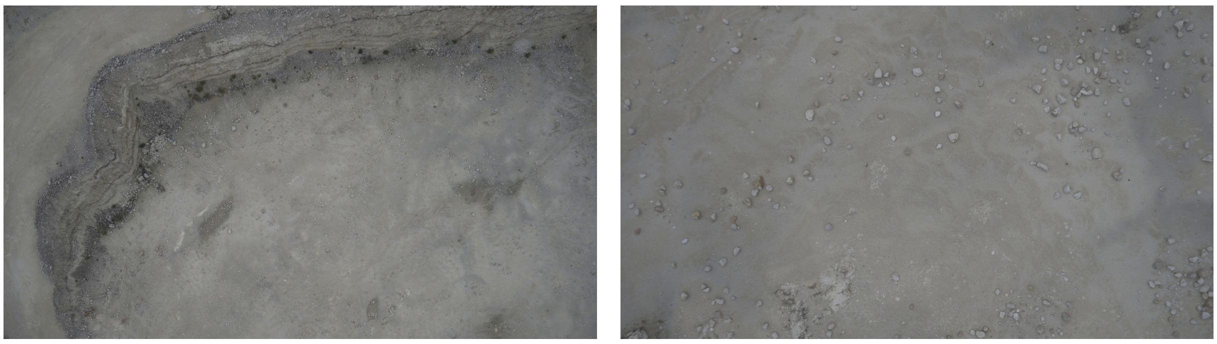


Figure 4.3: Example images taken by the downward-looking camera during the Spin.Works UAV Landing Mission at a quarry

era/aircraft, an estimation of the model of the camera used and the point cloud of the terrain.

The attitude is described by the euler angles (roll, pitch, yaw) and the position by GPS coordinates in the World Geodetic System 1984 (WGS84). In order to make use of this data, the position must be converted from geodetic coordinates to cartesian coordinates. First, we converted them to the ECEF system (*earth-centered, earth-fixed*) that represents positions as X, Y and Z coordinates. The origin is defined as the center of mass of Earth, the z-axis points to the true north and the x-axis intersects the sphere of the earth at the equator and the prime meridian in Greenwich.

For navigation purposes, we like to use local tangent plane reference coordinates, which are a geographical coordinate system based on the tangent plane defined by the local vertical direction. In aircraft navigation, most objects of interest are below the aircraft, so the positive direction is often defined pointing down. This reference frame is called NED, which stands for *north, east, down*, since x-axis points to north direction, y-axis to east and z-axis down. It is more convenient for navigation as it is a local frame, the numbers involved are relatively small and the axes are more intuitive. Coordinates in ECEF can be converted to NED by choosing a reference position and using its X, Y and Z coordinates and its latitude and longitude. We chose its origin to be at the surface of the earth geoid. Now, we have the camera positions with respect to the NED frame and the terrain surface is at an altitude $z = -120m$, calculated from the point cloud. Hence, it is possible to compute the altitude above ground of the aircraft.

The Euler attitude, expressed in $[roll, pitch, yaw]$, corresponds to a rotation ZYX, such that, starting by overlapping the reference of the body with the local reference frame NED, the body rotates the yaw value about the z-axis, then it rotates the total pitch about the new y-axis and, finally, the roll angle about the current x-axis, always using the right hand rule, resulting in the Body reference frame.

All these data is assumed as ground truth for performance evaluation of the algorithms. From that, we extract the transformations between the surface plane and the image plane, the homographies between different frames, and the position and attitude at each instant to evaluate the motion reconstruction done by the image processing techniques.

4.2.3 Spin.Works Moon Landing Mission Simulation

Since getting real datasets with the ideal conditions and representation of the problem is a difficult task, we have attempted to create and evaluate the algorithms' performance in simulation datasets that reproduce the planetary landing mission.

Spin.Works generated some of these simulations using PANGU[5], which is a proprietary software from STAR-Dundee[6] often used by ESA. PANGU is a powerful set of tools for modelling the surfaces of planetary bodies such as Mars and the Moon. It can generate camera images from any position and orientation. Hence, they created some Moon landing trajectories and the software produced a collection of synthetic images, as the ones in Figure 4.4 reproducing the environment one would expect to see when in a landing phase in a Moon entry mission, forming a video sequence similar to the real one described in section 4.2.2.

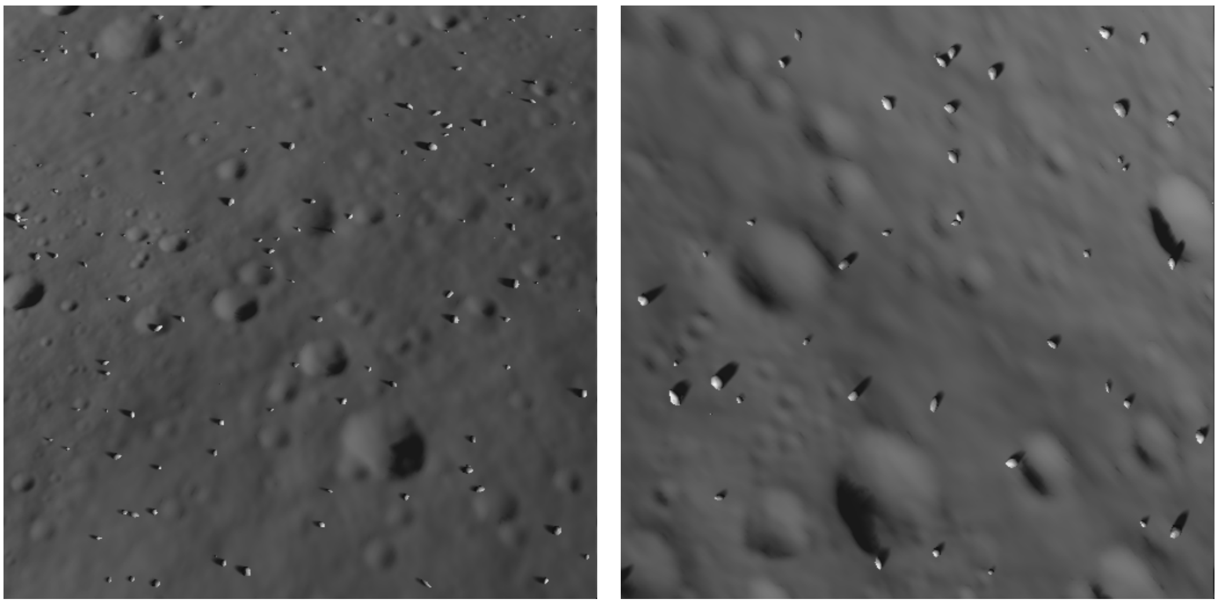


Figure 4.4: Example images from Spin.Works Moon Landing Mission Simulation

We have a completely controllable environment where the camera parameters, the motion states and the 3D of the surface is perfectly known. The attitude of the aircraft/camera is expressed in euler angles and the position is expressed in the ENU reference frame, which stands for *east, north, up*. ENU is also a local tangent plane frame with a different convention for the axis. The east axis is labeled with x, the north y and the up z. One can easily convert to NED coordinates to be coherent with the representation on the real dataset. The x and y-axis must be swapped and the sign of the z-axis must be changed.

[5]https://pangu.software/
[6]https://www.star-dundee.com/

4.2.4 Perseverance Rover's Descent and Touchdown on Mars (Official NASA Video)

During Perseverance rover's descent on February 18, 2021, terrain relative navigation was used to improve the knowledge about position and to choose a safe landing site. NASA posted online many images and videos from the mission, including the video from the rover's descent and touchdown seen by a down-looking camera. It can be watched in `https://www.youtube.com/watch?v=4czjS9h4Fpg`.

It is a great opportunity to validate and compare the algorithms using real images, exactly from the domain we are investigating on this Thesis. That is the most recent and representative dataset available for our problem. Therefore, we have extracted the frames from the YouTube video, at 10 Hz, and cut the regions we care about. Then, we resized each frame to 512×512 dimension, forming a set of 1310 images of the surface from Mars during rover's descent, similar to the ones in Figure 4.5.

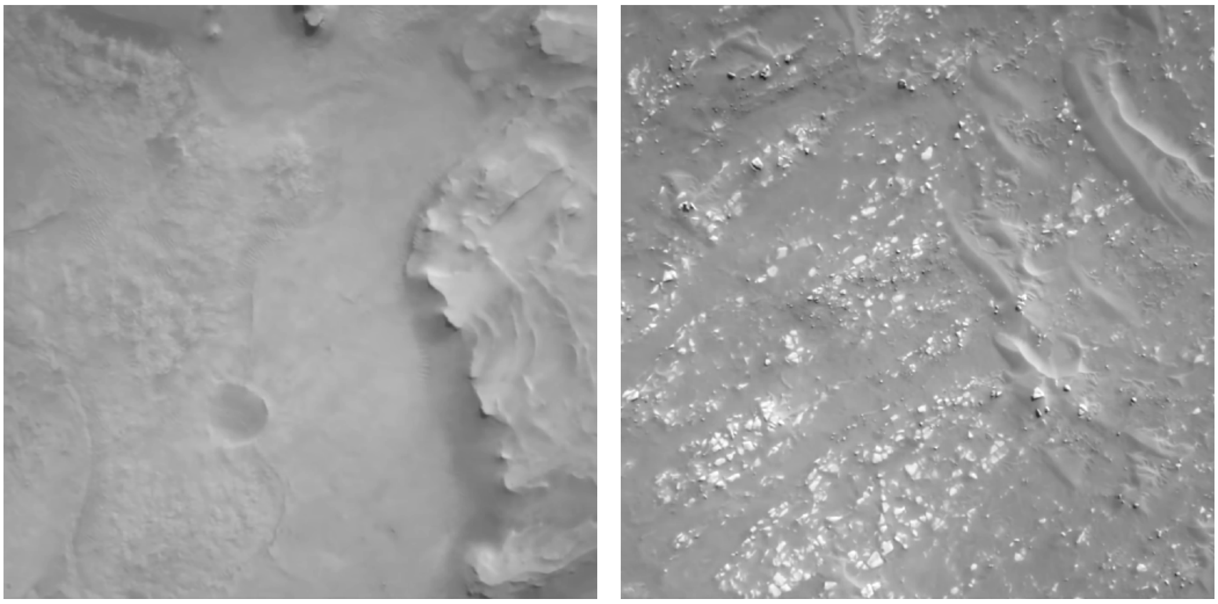


Figure 4.5: Example images from Perseverance Rover's Descent and Touchdown on Mars (Official NASA Video)

NASA did not publish navigation data for us to compare our estimates. However, at Spin.Works, they did a similar approach from the UAV dataset and used the SfM software to get orthophotomaps of the terrain, as well as position, attitude and model of the camera from the sequence of images.

Positions are defined up to a scale and attitudes may not be defined with respect to the surface frame. Hence, we need to identify ground control points to get information about the scale factor in position and the normal of the surface. On one hand, we have rotated data so that the last frame is vertical to the terrain surface. On the other hand, we know from information published by NASA, that backshell separation occurs at about 1.3 miles (2.1 kilometers). That frame is visible on the sequence, with a white puff. So, we could scale position using that information, resulting in a trajectory starting at 11 kilometers from the surface of mars, which is consistent with the EDL diagram published by NASA, where the heat shield separation occurs at about 7-11km, represented into the first frames of the video. From the camera model, we used the focal length and defined an approximation by a linear model.

Finally, we get data for the 6 degrees of freedom during descent, which we use as reference.

4.3 Feature Detection Algorithms

4.3.1 Harris Corners

We have implemented Harris detector using the function *goodFeaturesToTrack* from OpenCV library with the flag *useHarrisDetector* set to True and a fixed number of best features, quality level and minimum distance between features. The function returns a list of coordinates for the features identified.

For each of these features we computed a descriptor as the patch around the pixel location. We have selected a window size of 15 by 15 pixels centered at the keypoint location and saved the intensity values in that patch. The result matrix of intensity values is flattened to a vector of dimension 225. Finally, in order to get illumination invariance of the descriptor we normalized that vector by subtracting its mean and dividing by its standard deviation.

Next, we defined a similarity measure to compare descriptors from different measures. From several options we chose the *normalized cross-correlation* which is defined as

$$NCC = \frac{\sum_i \sum_j (f_1(i,j) - \mu_1)(f_2(i,j) - \mu_2)}{\sqrt{[\sum_i \sum_j (f_1(i,j) - \mu_1)^2][\sum_i \sum_j (f_2(i,j) - \mu_2)^2]}} \tag{4.1}$$

where f_1 and f_2 represent the intensity values in the window patches in image 1 and 2, respectively and μ_1 and μ_2 are the mean values of that patches.

Score values range from 1 (perfect match) to -1 (completely anti-correlated), so the higher the score, the better the match. Since we are treating the patches as normalized vectors, they are unit vectors. Therefore, the correlation becomes the dot product of the descriptors. In order to disambiguate the descriptor matching, we use *cross-check* or *two-way nearest neighbor* described in section 2.2.2.

4.3.2 SIFT

SIFT algorithm was also implemented using OpenCV. In this case, the function returns keypoint locations and SIFT descriptors. The default parameters were used, we also chose the maximum number of features to be similar to the other algorithms during performance evaluations.

The similarity metric adopted was the *L2 norm*. This distance measures the square root of the sum of the squared differences between the descriptors from different images. As in the other methods, we also implemented *brute-force* matching using *cross-check* to distinguish ambiguities.

4.3.3 SuperPoint

SuperPoint was implemented and trained from scratch using PyTorch framework, following the procedure described in the paper [22], but with slight differences. SuperPoint training is divided into 3 phases as expressed by the Figure 4.6

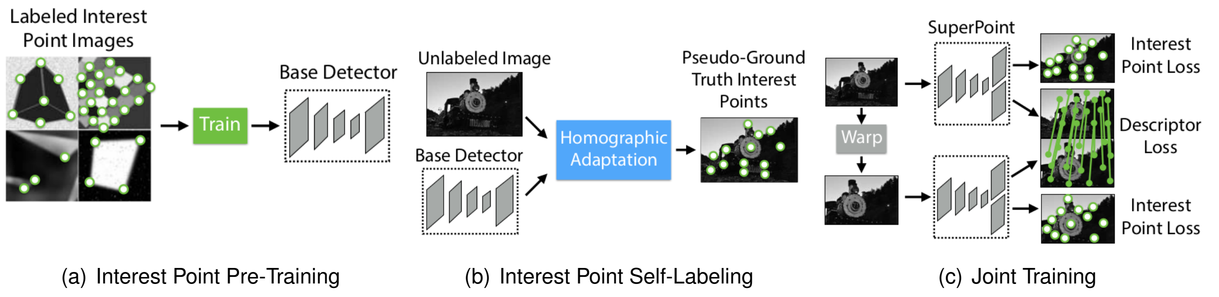


Figure 4.6: **SuperPoint training overview.** First, an initial detector network is trained using synthetic data. Then, that network produces pseudo-ground truth points on MS-COCO images. Finally, SuperPoint is trained using the labels generated before. Adapted from [22]

The initial detector model, called MagicPoint, is firstly trained with synthetic images generated on-the-fly. In each iteration, the program renders one batch of 32 240×320 images with synthetic shapes such as triangles, squares or chessboard-like structures and the respective ground truth locations of the corners, junctions of lines and other points of interest. MagicPoint network is similar to the SuperPoint one, shown in Figure 4.7, but with no descriptor decoder.

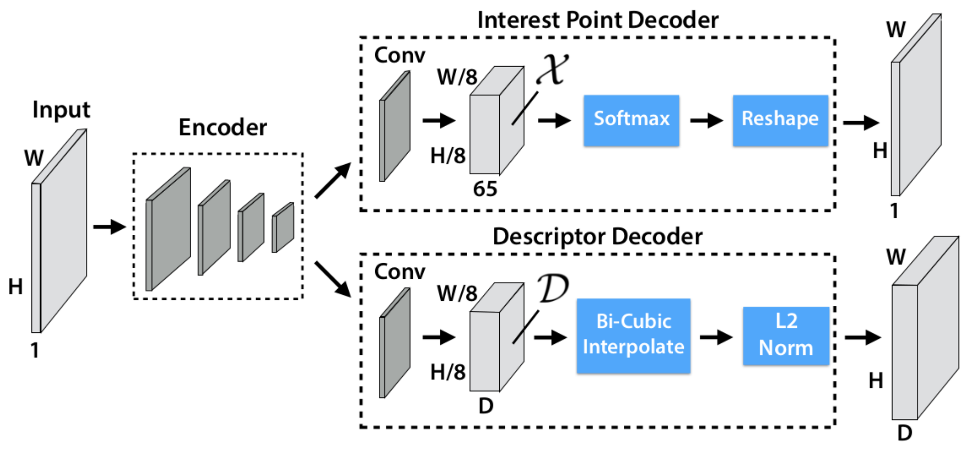


Figure 4.7: **SuperPoint Architecture.** Decoder with VGG-like architecture with 3x3 convolutional layers sized 64-64-64-64-128-128-128-128. Every two layers there is a 2x2 max pooling layer. Detector Decoder with a a single 3x3 convolutional layer of 256 units followed by a 1x1 convolutional layer with 65 units and finally a channel-wise softmax layer. Descriptor Decoder composed by a 3x3 convolutional layer of 256 units followed by a 1x1 convolutional layer with 256 units. All convolution layers in the network are followed by ReLu non-linear activation and BatchNorm normalization. From [22]

The second phase comprises the generation of pseudo-ground truth labels using the MS-COCO 2014 [28] training dataset split which has about 80,000 images and the MagicPoint model trained earlier. The images are resized to $240x320$ and converted to grayscale and the labels are generated using **Homographic Adaptation** method described in the paper and represented by the Figure 4.8. During Homographic Adaptation a total of 100 random homography matrices are generated as the composition of simpler transformations, such as translation, scaling or rotation as described in section 2.3.1. The points that are detected in most transformations are the ones considered as pseudo-ground truth interest point locations for each image.

The joint training of SuperPoint is done on the grayscale MS-COCO images with the labels generated

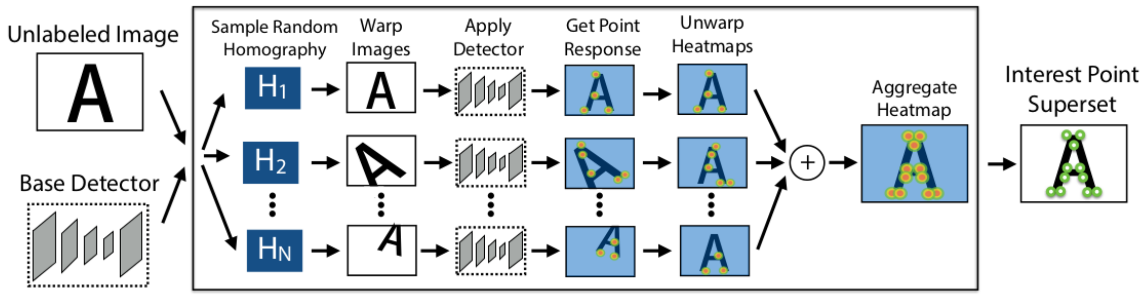


Figure 4.8: **Homographic Adaptation.** Homographic Adaptation is a form of self-supervision for boosting the geometric consistency of an interest point detector. From [22]

from the Homographic Adaptation with batch size equal to 2. For each training image, a homography is randomly sampled using the same method described earlier, but with more restrictive parameters than during Homographic Adaptation, such as, less-extreme rotations. Both the image and the pseudo-ground truth labels are transformed by this homography and the detector and descriptor losses are optimized simultaneously. The interest point detector loss is a cross-entropy loss over the downsampled tensor and the descriptor loss is a hinge loss over the possible descriptor correspondences. They are well described in section 3.4 of SuperPoint paper. To increase the repeatability of the keypoints detected, another round of Homographic Adaptation is done before the joint training of SuperPoint, resulting in a new labelled COCO dataset.

The descriptor loss was slightly modified from the original paper inspired by the paper [29] (section 3.2 paragraph Matching layer) and the *Tensorflow* implementation in the repository `https://github.com/rpautrat/SuperPoint`. The descriptor loss is now computed on L2 normalized descriptors and, after computing the distance matrix between the descriptors of both images, we have applied ReLu activation to eliminate obvious non matches (similarity of the descriptor below 0). Then, the distance matrix is renormalized to help disambiguate when several descriptors are very close. Since the distance matrix has been normalized, the positive matches have a higher activation than all the negative ones, so the parameters proposed in the paper needed to be slightly changed. The descriptor loss balancing term λ_d becomes 0.05 and the balacing factor between the two losses λ becomes 10,000. The other parameters are kept equal to the ones proposed in the paper.

All training was done using 2 NVidia GeForce GTX 1070 GPUs and the optimizer was ADAM with default parameters of $lr = 0.001$ and $\beta = (0.9, 0.999)$. Data augmentation techniques such as gaussian blur, gaussian noise, speckle noise, and random contrast and brightness changes were implemented to improve the model's robustness to lighting and viewpoint changes.

At inference time, the method can be described by the algorithm 2

Upsampling layers tend to add a high amount of computation, thus the authors propose this decoder with no parameters, also known as "sub-pixel convolution" [30] or "pixel shuffle in PyTorch, in order to get an output with the same dimensions has the input. The heatmap expresses at each pixel the confidence of being an interest point. The threshold we use is equal to 0.015, as suggested in the paper. It happens that sometimes the heatmap probability is somewhere distributed by the neighbor pixels and after applying threshold, we get multiple detections of the same point at close coordinate pixels. The

Algorithm 2 SuperPoint Detection and Descriptors

```
1: $model \leftarrow SuperPoint(weights)$    ▷ Create model using pre-trained weights
2: procedure DETECTANDCOMPUTE($image, NMSDist, threshold, numPoints$)    ▷ Grayscale image
3:     $heatmap, descriptorsRaw \leftarrow model(image)$    ▷ Spatial dimensions of H/8xW/8
4:     $heatmap \leftarrow pixelShuffle(heatmap)$    ▷ Upsampling heatmap
5:     $heatmap \leftarrow NMS(heatmap, NMSDist)$
6:     if $heatmap(x, y) > threshold$ then    ▷ For every location (x,y) in heatmap
7:         $coordinates = (x, y)$
8:         $score = heatmap(x, y)$
9:     end if
10:    $coordinates \leftarrow bestFeatures(score, numPoints)$    ▷ Save only fixed number of best points
11:    $descriptors \leftarrow BilinearInter(descriptorsRaw, coordinates)$
12:    $descriptors \leftarrow L2Norm(descriptors)$
13:    return $coordinates, descriptors$
14: end procedure
```

solution is to apply Non-maximum Suppression (NMS) to the heatmap. In our implementation, we firstly identify the keypoint locations by thresholding the heatmap and then we create a patch around it whose score is the confidence value. We run a box NMS function from *torchvision* library that selects from intersecting patches the one that has the highest score. Now, we get the coordinates of the keypoints identified by the model after eliminating the multiple close detections.

The descriptor decoder outputs a semi-dense grid of descriptors (one every 8 pixels), because learning the descriptors semi-densely rather than densely reduces training memory and keeps the run-time tractable. Then, the paper proposes to perform bi-cubic interpolation of the descriptor and then L2-normalization. However, we used bi-linear interpolation instead as it is faster and the results are similar. We interpolate only the descriptors at the keypoint locations rather than interpolate the dense descriptor map to get lower the computation time.

As in SIFT, we have implemented the L2-norm as a similarity measure for the descriptor matching. All the operations above mentioned were implemented with the possibility of using GPU acceleration due to the parallelization of most computations, which enables SuperPoint to be used for real-time applications.

Sub-pixel Refinement

SuperPoint corner locations are defined with pixel coordinates, while other detectors like SIFT have sub-pixel precision. Some of the experiments we made got better results with SIFT due to this fact. So, we have implemented a method to refine corner locations from SuperPoint. After getting the corner locations, we apply the function *cornerSubPix* from OpenCV that iterates to find the sub-pixel accurate location of corners as described in [31].

The function gets the grayscale image, the corner locations from SuperPoint and three more parameters that we use default values from OpenCV documentation.

4.4 Homography Estimation & Motion Reconstruction

4.4.1 Ground Truth Homographies

HPatches dataset provides ground truth homographies between different images of the same scene. Datasets provided by Spin.Works do not have explicit ground truth homographies between the frames along the video sequences. Nevertheless, it is possible to compute the reference homographies using the camera model and the motion parameters described in sections 4.2.2 and 4.2.3. Our approach is to compute the transformations between the ground surface plane and the image plane, as expressed by the algorithm 3.

Algorithm 3 Ground truth homographies from navigation states and camera model

```
1: $pixelCorners \leftarrow computeCoordinates(camImageDims)$   ▷ 4 image corners in pixel domain
2: $cameraVersors \leftarrow computeCoordinates(FoV)$   ▷ 4 image corners in homogeneous coordinates
3: procedure SURFACETOIMAGEHOMOGRAPHY($position, attitude$)   ▷ Ground to image transformation
4:     $Body2NED \leftarrow angle2dcm(attitude)$   ▷ Direction Cosine Matrix Body to NED
5:     $cameraVersors \leftarrow multiplication(Body2NED, cameraVersors)$   ▷ Rotate versors to NED
6:     $groundCorners \leftarrow project(cameraVersors, position)$   ▷ 4 image corners in pixel domain
7:     $homography \leftarrow DLT(groundCorners, pixelCorners)$
8:     return $homography$
9: end procedure
```

Image corners in homogeneous coordinates, which we call *camera versors*, are computed from the *field of view* (FoV). When we do not have access to the field of view of the sensor, it is trivial to compute it from focal length and image dimensions in pixels, using trigonometric relations,

$$FoV_i = 2arctan(\frac{d_i}{2f_i}) \tag{4.2}$$

where FoV_i, d_i and f_i represent the field of view, the image dimension and the focal length, respectively, along each of the dimensions.

Camera versors and image corners are properties that only depend on the camera, thus they are independent of the camera position and do not change along the frame sequence. Camera versors are defined in homogeneous coordinates in the Body frame, so we can easily convert them to the NED navigation frame by a matrix-vector product. The rotation of the Body with respect to the NED frame is expressed in Euler angles. The rotation matrix can be computed from the Euler angles as,

$$\begin{aligned} R(\phi,\theta,\psi) = R_x(\phi)R_y(\theta)R_z(\psi) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cons\phi \end{bmatrix} \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} cos\psi cos\theta & sin\psi cos\theta & -sin\theta \\ cos\psi sin\theta sin\phi - sin\psi cos\theta & sin\psi sin\theta sin\phi + cos\psi cos\theta & cos\theta sin\phi \\ cos\psi sin\theta cos\phi + sin\psi sin\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\theta cos\phi \end{bmatrix} \end{aligned} \tag{4.3}$$

The corners of the region of the surface ”seen” by the camera, in meters, are computed projecting camera versors, using similar triangles relations and the camera position. DLT algorithm, described in 2.3.2, computes the homography transformation between the terrain surface plane and the current frame plane, using 4 corners in meters and in pixel domain.

The ground surface plane remain constant along the frame sequence. Therefore, it is possible to calculate any homography by matrix operations followed by normalization. If the evaluation procedure requires inter-frame homographies, we compute them using the formula,

$$H_i^j = H_j H_i^{-1} \tag{4.4}$$

where H_i^j represents the homography between frames i and j, and H_i and H_j are the transformations between the world surface and the image plane, respectively.

4.4.2 Test Set Homographies

In the last section, we have seen how to compute the homographies we use as the reference set, from navigation states and camera model. We have to estimate the same homographies through image processing techniques to create our test set.

The procedure to compute these transformations is independent of the algorithm chosen for the feature detection, description and matching. The homography matrix is computed from the correspondences between features from different frames using the DLT algorithm. However, not all correspondences are correctly identified and we have implemented the RANSAC algorithm to reject outliers and compute the estimate only with correspondences that are considered inliers using a threshold on a geometric distance between estimated points and original points. Estimated points are transformed from the source image using the estimate of the homography matrix and the L2-norm distance is calculated with respect to the correspondent points in the destination image. The correspondences whose distance is under some defined threshold are considered inliers to the model.

RANSAC was implemented in our framework, following the algorithm described in section 2.3.3. The parameters were carefully chosen to obtain the best results. We have defined a reprojection error threshold of 1 pixel, a confidence of 0.999 and an acceptance inlier ratio of 0.9 for a maximum of 2000 iterations.

4.4.3 Non-Linear Least Squares

We have shown, in section 2.3.5, that the coordinates of points from the same plane are related by a homography transformation. But when we use image processing techniques to estimate the homography we get a matrix that relates positions in different frames, in pixel units. As shown before, it is possible to use the camera intrinsics matrix to get this transformation in meters.

Homographies are used to estimate camera positions and attitude in order to reconstruct spacecraft motion. To evaluate the performance of the algortihms in that task, we used the datasets described in 4.2. In the case of the real dataset, images are resized to reduce computational cost. Therefore, the sensor parameters need to be resized, namely the focal length, e.g., if we show one pixel per two from the original images, focal length, in pixel units, need to be divided by two. Recalling the definition of focal length from section 2.3.4, $\alpha_x = fm_x$, where α_x represents the focal length in pixel units, f the physical focal length of the camera and m_x the number of pixels per unit distance in x-direction (inverse of pixel size), if we resize the images by half, it means pixel size is double, so $m'_x = \frac{m_x}{2}$. Also, the principal point, in pixel units, needs to be also scaled to the new image size.

The image processing community often defines the pixel image reference frame with center at the upper left corner of the image, x-axis horizontal and pointing to the right and y-axis vertical and pointing down, while in body reference frame it is useful to define x-axis vertical and pointing up and y-axis horizontal and pointing right to be consistent with NED frames, euler angles and rotation matrices. Therefore, it is trivial to convert from camera sensor frame to body frame, multiplying by the orthogonal matrix

$$D = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Homographies computed from image processing techniques must be converted to euclidean homographies, as described in section 2.3.5, by the formula,

$$H_{euc} = \frac{K^{-1}DH_{proj}D^TK}{\gamma} \tag{4.6}$$

where γ is the scale factor that corresponds to $\frac{z_f}{z_i}$ and z_f and z_i are the z-coordinates of some point in the terrain surface with respect to cameras (i) and (f). However, it is not trivial to found this coordinate values for common points between the frames. In somes cases, using the altitude or the range to ground are good approximations of the values z_f and z_i, but as we move from the reference frame, estimates start getting worse, since z-values refer to different points in plane. The correct solution can be found in the article [32]. If we notice that the median of the singular values of H_{euc} is equal to 1, then we can compute the scale factor γ as follows:

$$\gamma = median(svd(K^{-1}DH_{proj}D^TK)) \tag{4.7}$$

where svd returns the singular values of H_{euc}, in ascending order. SVD is implemented using the function *svd* from *numpy linalg* library.

After applying equation 4.6 to every single homography matrix computed by the image processing techniques, we get the estimates of the euclidean homographies between the intervals of frames considered, so we have information about the relative pose of the camera between those intervals. That information is implicit and can be recovered if we define the model

$$H_i^f = R(\phi, \theta, \psi) + \frac{t(t_x, t_y, t_z)n^T}{d} \tag{4.8}$$

where H_i^f represents the euclidean homography between camera (i) and camera (f), R the rotation matrix, function of 3 parameters (ϕ, θ, ψ), t the translation vector $(t_x; t_y; t_z)$ between the two cameras, considering that camera (i) is at the origin (no translation) and with no rotation, n is the normal vector of the ground plane, expressed in the camera (i) reference frame and d the distance to the ground plane, or altitude above ground of camera (i).

Our goal is to produce an estimate of the vector of parameters, x, which we call *state vector*, that represents the position and orientation of the aircraft at each time frame and can be defined as

$$x = [\phi, \theta, \psi, t_x, t_y, t_z] \tag{4.9}$$

The estimate of x must be the one that better approximates the data represented by the vector y, which in our case, denotes the estimated euclidean homography matrix described above, H_i^f, re-organized from 3x3 to a 9x1 vector, assuming the model $y = f(x)$, specified in equation 4.8. $n[3 \times 1]$ and d are constants computed, a priori, and related with the pose of camera (i) with respect to the ground plane.

The function f is non-linear, which complicates the problem. However, we can assume that given an initial rough condition for our state vector x, it is possible to produce some cost function where our state space is locally convex and, then, where it is possible to converge to a local minimum that matches the optimal estimate of x. This problem can be described as a local optimization problem, and can be solved iteratively, using **non-linear least squares** (NLLS).

The linearization of the homography function becomes

$$H_i^f(x) = H_i^f(x_0) + \frac{dH_i^f}{dx}(x_0)[x - x_0] \tag{4.10}$$

where $H_i^f(x)$ represents the estimate of the euclidean homography at state x, $H_i^f(x_0)$ the result of applying the homography model at initial state vector, x_0, $\frac{dH_i^f}{dx}(x_0)$ denotes the Jacobian of the model at x_0, explicitly defined as $\left[\frac{dH}{d\phi}, \frac{dH}{d\theta}, \frac{dH}{d\psi}, \frac{dH}{dt_x}, \frac{dH}{dt_y}, \frac{dH}{dt_z}\right]$, a 9x6 matrix and, finally x stands for an improved estimate of the actual state vector (rinse repeat).

At each iteration of the NLLS algorithm, the state vector is updated by solving the system of linear equations specified in equation 4.10. Since the problem is represented by an overdetermined system, i.e, we have more observations than unknowns $(9 > 6)$, linear least squares method is used to get the solution that minimizes the Euclidean 2-norm $||b - A\Delta x||$, where b is defined as $H_i^f - H_i^f(x_0)$, $A = \frac{dH_i^f}{dx}$ and Δx represents $x - x_0$.

The solution uses the **Moore-Penrose inverse** to allow the inversion of the problem, and is expressed by the formula

$$\Delta x = (A^T A)^{-1} A^T b \tag{4.11}$$

We implement this step using the function *lstsq* from *numpy linalg* library, which operates in a similar way as the *backslash* operator (\) in *Matlab*.

We make use of the linearization and recursivity due to the fact that the equations are highly non-linear, which forces the estimate to be done in small steps through the steep gradient and linearizing again at each iteration.

A key frame must be given, a camera whose navigation motion states are known: position (altitude above ground) and attitude. The rotation matrix $R(\phi, \theta, \psi)$ is replaced by $R(\phi, \theta, \psi)R_0$ in the model, where R_0 is the rotation matrix from the key frame camera to the (ground) local vertical reference frame.

The problem is somewhat badly conditioned, because the attitude in radians is very small and translation in meters is very large, and the function is highly non-linear. We could perform a scaling operation to normalize the state variables, but we decided to scale down the Δx, by a weight factor and increase the number of iterations to get a slow but stable convergence. Using a scale factor of $w = 0.009$ and a number of iterations $N = round(20/w)$, it has a good behavior.

At the end of each convergence (between any two frames), we store the estimated attitude (ϕ, θ, ψ) and translation $t = (t_x, t_y, t_z)$. The translation vector needs to be first transformed back to the reference frame of the key-camera-frame, through: $t = -R(\phi, \theta, \psi)^T \times t$, before storage.

4.5 Evaluation Metrics

The objective of this Thesis is to evaluate the performance of image processing algortihms to reconstruct spacecraft motion in landing missions. Therefore, our evaluations are conducted in three phases. We start by testing the repeatability of the points, then using an homography estimation metric and finally we get to a more advanced metric where we compare the motion states directly. All evaluation metrics are described below.

Repeatability measures the detector's ability to identify the same features despite variations in the viewing conditions. Using ground truth homographies to get reference locations of the keypoints detected in the first image, in the second one, we compute the distance between points detected in the second image and that reference points. If distance is under some threshold ϵ, the point is considered to be re-identifed and we define repeatability as the number of points that are re-identified over the number of points detected.

We evaluate the ability of algorithms to estimate the homography relating a pair of images, by comparing the matrix estimated by the methods described in the last sections to the ground truth homography. Since different entries of the matrix have different scales, it is not straightforward to compare matrices directly. Hence, we use the **Reprojection Error**, where we compare the performance of the homography in estimating the location of the four corners of one image into the other. We characterize the corners as c_i with $i = 1, 2, 3, 4$ and then we apply the ground truth homography to get the reference location of them in the second image, c'_i, and the estimated homography to get the test locations, $\hat{c}'_i$. Reprojection error is the mean of the L2 distances from the four reference corners and the four test corners. We define a threshold ϵ in pixels to denote the reprojection error acceptable to consider a correct homography. The percentage of correct homography estimation is the ratio between the number of pairs whose error is under the threshold and the number of pairs evaluated.

The most important metric to our study is the **Motion Reconstruction Errors**. Here, we evaluate position and attitude estimates by comparing them with ground truth navigation parameters provided, as described in datasets section. Consequently, we plot Euler angles and translation values along the frame sequence and estimation errors for each motion state variable.

We use HPatches dataset only as a starting point to evaluate metrics such as repeatability and homography estimation. Then, we focus on navigation datasets where we test those two metrics and also the comparison of motion states. There we performed several experiments with different conditions. Essentially we have two different pipelines.

On one hand, we detect points on every frame and compute estimates of the homography relating pairs of successive frames. The transformation from the world plane to the i^{th} frame plane can be estimated as a sequence of frame-to-frame homographies multiplication as follows:

$$H_w^i = H_{i-1}^i H_{i-2}^{i-1} ... H_0^1 H_w^0 \tag{4.12}$$

where H_w^0 defines the transformation from the ground plane to the reference frame plane chosen where we assume to know the motion states. The other H_{i-1}^i are the homography transformations between successive frames. Consequently, we get the transformation between the ground surface and every frame from the video sequence using composition of homographies estimated from image processing techniques. We tested different conditions, such as changing the reference frame and the *delta* between frames, i.e., the interval between frames we use to estimate homographies using the procedure described in section 4.4.2.

On the other hand, we choose the key frame, detect keypoints and save descriptors. Then, we perform directly the estimation of the homography between the key frame and the current frame. For each frame i we get the expression

$$H_w^i = H_0^i H_w^0 \tag{4.13}$$

where H_w^0 is still the transformation from the world plane to the key frame, which is computed assuming that we know motion states of the frame we choose as reference. These experiments allows to evaluate the strength of features detected by different algorithms so we can perceive the ones that keep on being detected during longer periods of time. This procedure contributes to eliminate cumulative errors caused by composition of homographies, as described above.

Chapter 5

Results

In this Chapter, we describe our experiments following the framework described in the implementation chapter and present results achieved by the algorithms in testing. We start using HPatches dataset to evaluate metrics related to detections and homography estimation and then we move to navigation datasets. First, we use a video sequence of a landing mission of a UAV in a representative surface. Then, we test on moon synthetic images from a simulated landing trajectory and finally, we perform experiments using the video of the latest Mars landing mission from NASA.

5.1 HPatches

We started our evaluations using the HPatches dataset. As stated in section 4.2.1, the dataset is divided in two groups: one with images that exhibit illumination changes and other with images that are transformed by viewpoint changes. Therefore, our tests are divided into *Viewpoint* and *Illumination*.

5.1.1 Repeatability

We compare the ability to detect the same keypoints in different images of the same scene, using the repeatability metric. Repeatability is computed at 240×320 resolution with 300 points detected in each image and we use a correct distance threshold of $\epsilon = 3$ pixels. Results are summarized in table 5.1.

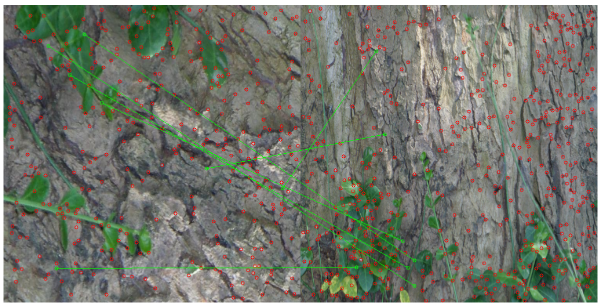
	Illumination	Viewpoint
Harris	0.601	**0.593**
SIFT	0.487	0.476
SuperPoint	**0.640**	0.508

Table 5.1: **Detector Repeatability on HPatches.** Repeatability measures the probability that a point is detected in the second image. SuperPoint is the most repeatable under illumination changes, while Harris Corners exhibits better performance under viewpoint changes.

Our implementation of the SuperPoint model outperforms both classical detectors under illumination changes. However, Harris Corners is the most repeatable detector under viewpoint changes. SIFT reveals poor performance on this metric.

5.1.2 Homography Estimation

The problem we aim to solve with this Thesis relies on an accurate estimate of the homography matrix between different images of the same scene. So, we perform our evaluations using the re-projection error defined in section 4.5, comparing performances of three algorithms under discussion.


(a) **Incorrectly estimated homography.** Re-projection error higher than 10 pixels, meaning that it is considered incorrect under all thresholds between 1 and 10.

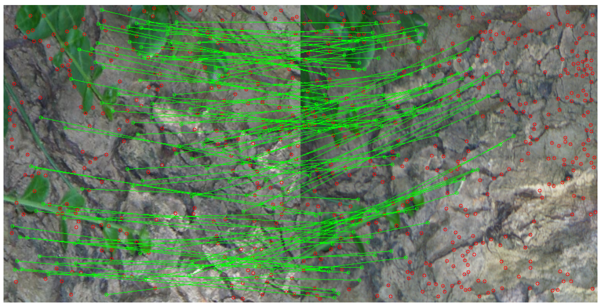
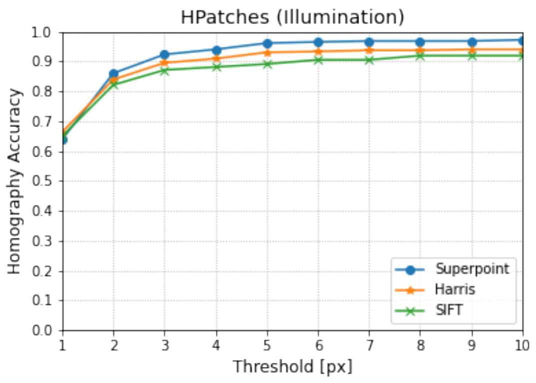
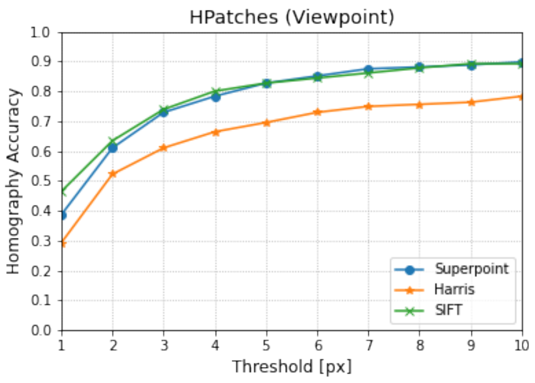

(b) **Correctly estimated homography.** Re-projection error between 1 and 2 pixels, meaning that it is considered correct under all thresholds from 2 to 10 and incorrect using threshold equal to 1.

Figure 5.1: HPatches example of homography estimation using matches from SuperPoint interest points. Green lines represent matches that are consider inliers on homography estimation. Red points are detections that were not matched or whose matches were considered outliers.

We evaluated the set with illumination changes and the set with viewpoint changes separately. We computed a maximum of 500 points for all algorithms at a 480×640 resolution. The re-projection error is calculated for each pair of images in each set and if it is lower than the threshold ϵ, we consider it correctly estimated (example in Figure 5.1). Then, the ratio of correctly estimated pairs by the total number of pairs is the homography accuracy in the plots. We performed these tests with different thresholds, ranging from 1 to 10, and the plots below show the comparison between the algorithms.



(a) Illumination Changes



(b) Viewpoint Changes

Figure 5.2: Homography estimation accuracy curves on HPatches. The accuracy represents the ratio between correctly estimated homographies and the total number of image pairs, for re-projection error thresholds ranging from 1 to 10.

SuperPoint outperforms both classical methods in the homography estimation task for image pairs with illumination changes. Also, it performs comparably to SIFT and considerably better than Harris in the case of viewpoint changes. These results suggest that SuperPoint could be a more accurate alternative to classical methods to solve the problem addressed by this Thesis.

However, we need to prove that these results remain when it comes to images of lunar and planetary terrain taken by a spacecraft in a landing phase. Also, we need to understand the influence of this low level metric in the estimate of motion states to better distinguish the algorithms' performance in the high level task of estimating the relative motion of the aircraft descending to the surface. Therefore, next we perform evaluations in datasets that are more representative of the problem at hand.

5.2 Spin.Works UAV Landing Mission

The Spin.Works UAV landing mission dataset, described in detail in section 4.2.2, was our first approach to compare classical algorithms versus the deep learning one in an environment that looks similar to what is expected in a lunar or planetary landing mission. The terrain surface from the quarry could be a good approximation of what one camera would see when approaching lunar or planetary surface, so results here should be more reliable than using an all-purpose dataset, such as HPatches.

We also used the homography estimation metric, but we go further and compare motion states estimates to results from the SfM software to evaluate algorithms' performance on the task we really care about.

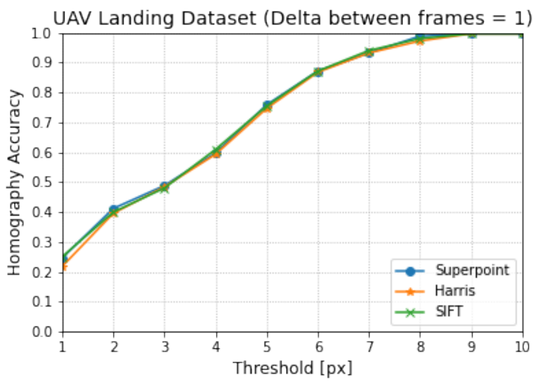
Original images taken frame-by-frame from the video sequence have dimensions 1080×1920. In order to balance performance and processing time and resources, images were resized to 512×512 for all performance tests. Ground truth homographies are calculated from navigation states provided by the SfM software, using the procedure described in section 4.4.1.

5.2.1 Homography Estimation

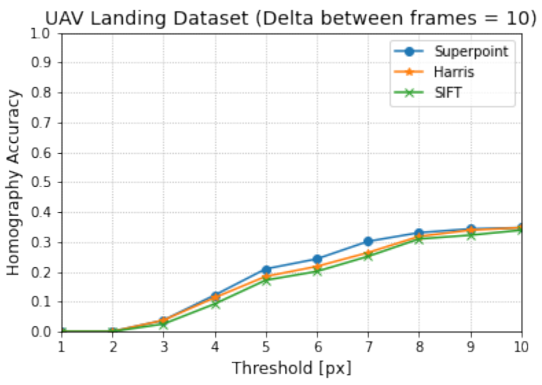
We started by evaluating the performance of the algorithms in the task of estimating homographies that represent image transformations between different frame transitions. First, we tested all transitions between consecutive frames and then, we have successively increased the interval between frames. Plots below express the accuracies, with thresholds ranging from 1 to 10, for homography estimates at intervals of 1, 10 and 20 frames, which we call *delta between frames*.

Results indicate that SuperPoint performs comparably to classical methods in the case of intervals of consecutive frames and slightly better when we increase the delta between frames, specially when compared to SIFT. This consequence suggests that SuperPoint could be a powerful alternative for transitions that are more spaced in time, which make us believe that SuperPoint keypoints and descriptors are more invariant to viewpoint and illumination changes and so, the detected corners remain being correctly matched along longer time sequences.

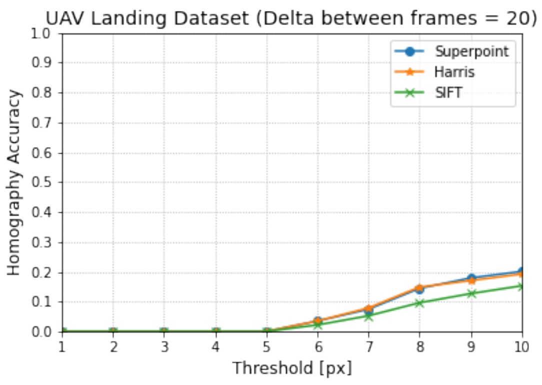
However, this metrics are insufficient to demonstrate that SuperPoint keypoints and descriptors are advantageous over Harris Corners and SIFT at the high level task of estimating motion states for the



(a) Delta between frames equal to 1



(b) Delta between frames equal to 10



(c) Delta between frames equal to 20

Figure 5.3: Homography estimation accuracy curves on UAV landing dataset from Spin.Works. The accuracy represents the ratio between correctly estimated homographies and the total number of image pairs, for re-projection error thresholds ranging from 1 to 10.

vision-based navigation problem that we are investigating with this Thesis.

5.2.2 Qualitative Results

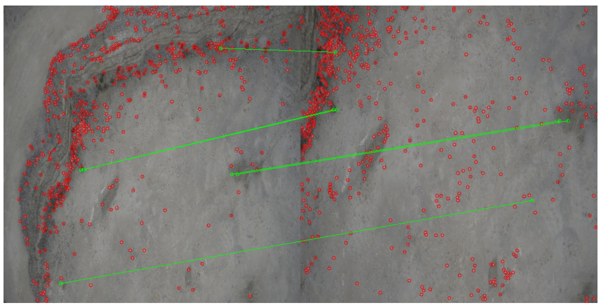
Homography estimation highly depends on the precision of the features detected and quality of the descriptors that contributes to correct matches between keypoints detected in different images of the same scene. One of the most important attributes of detectors is to provide repeatable and distinguishable features that remain being detected and correctly matched along the frame sequence, in order to provide stability to the vision-based navigation system.

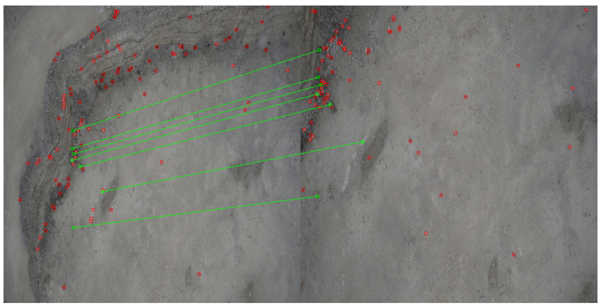
We have done a qualitative evaluation of this aspect, visualizing the features detected and correctly matched in two frames separated by increasing periods of time. We run the algorithms, implemented as described in section 4.3, to detect and describe a maximum of 500 keypoints on each frame image resized to dimensions 512×512. Then, descriptors are compared between frames, as reported in the same section. Finally, RANSAC uses correspondences to estimate the homography matrix that relates those frames and provides the set of inliers (section 4.4.2).

Inliers are correspondences whose re-projection error using the matrix estimated is below the threshold value parameterized into the implementation of RANSAC. We use the same values for the three algorithms: re-projection error threshold of 1 pixel, maximum of 2000 iterations and a confidence level of 0.999.

Qualitative results are shown by the images, where green circles and lines represent keypoint locations and correspondences, respectively, that belong to the inlier set, and red circles are keypoints that were detected, but not matched between frames, or whose match was considered an outlier.

In Figure 5.4 we tested the ability to find correspondences between features in images separated by a period of 80 frames, taking as reference the first frame of the video sequence. SuperPoint outperformed classical algorithms, specally Harris Corners, having a large set of reliable correspondences between frames, while Harris correspondences clearly fail. As we increased the frame interval, the number of correspondences was getting lower, namely when using Harris Corners or SIFT. SuperPoint is still visually having a good performance after 80 frames, where we see considerable changes in scale, which also supports our conviction that SuperPoint is a more stable and reliable detector for a vision-based navigation system. It is also interesting to notice the ability of SuperPoint to detect points in the region of the terrain outside the cliff, contrarily to Harris Corners or SIFT, which is important in the homography estimation task due to the assumption that the terrain is planar.


(a) Harris Corners - 6 Inliers / 71 Matches


(b) SIFT - 8 inliers / 42 Matches

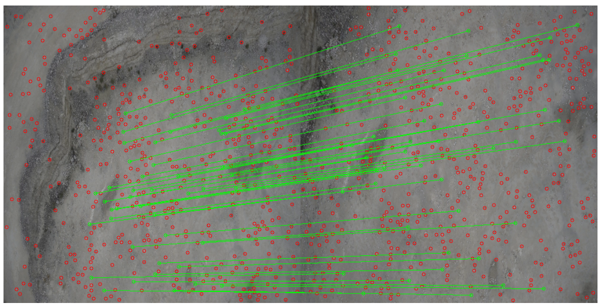
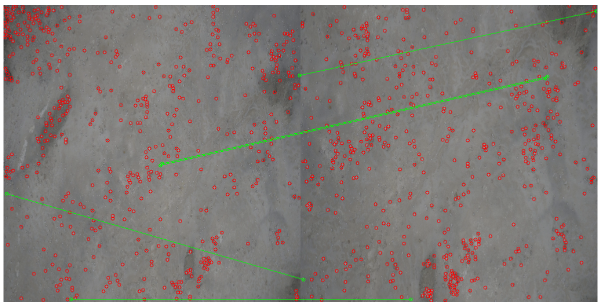
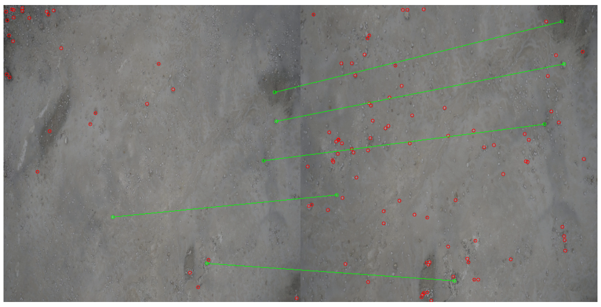

(c) SuperPoint - 44 Inliers / 161 Matches

Figure 5.4: Feature correspondences between frame 1 and frame 81 of the UAV landing video sequence

Another example can be seen in Figure 5.5, where the transition also expresses a significant scale change and where apparently the region has some repetitive texture. Besides that, SuperPoint keeps being the most promising of the three in the task of estimating the transformation between the frames.

Even with promising results for the deep learning alternative, it is not sufficient to clarify if it is capable of detecting points good enough to retrieve motion states of the spacecraft along the landing into a planet, so that it could be integrated into a system for vision-aided navigation.


(a) Harris Corners - 5 Inliers / 84 Matches


(b) SIFT - 5 inliers / 18 Matches

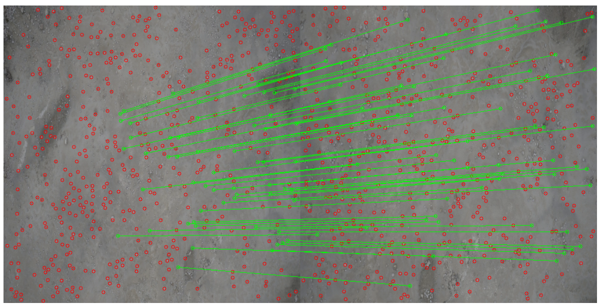

(c) SuperPoint - 56 Inliers / 143 Matches

Figure 5.5: Feature correspondences between frame 101 and frame 151 of the UAV landing video sequence

5.2.3 Navigation States Estimation

Using the procedure described in section 4.4.3, we retrieve motion states (position and attitude) from homographies estimated by image processing techniques. The SfM software gave us ground truth values of position and attitude that we assume as reference (shown in Figure 5.6). A pinhole model is defined, using focal length from the SfM software, to construct the intrinsics matrix.

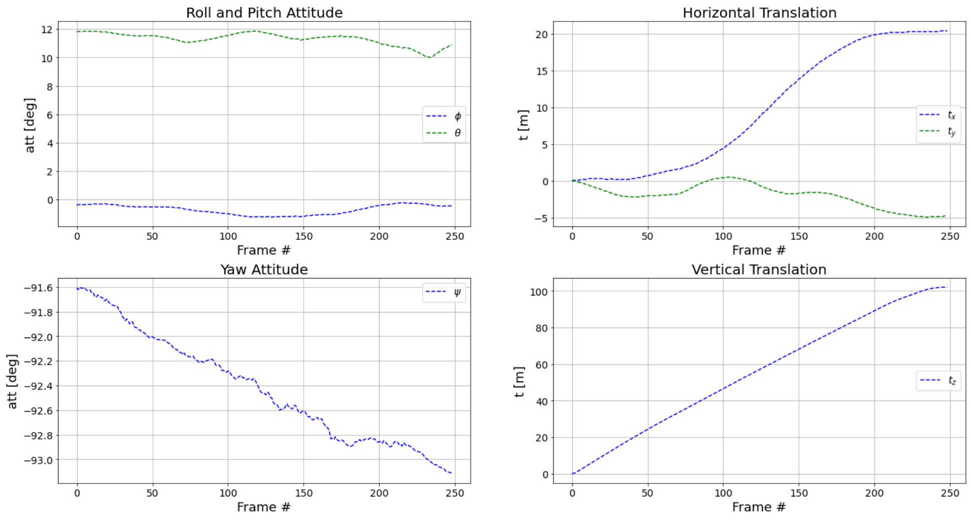


Figure 5.6: Reference navigation data (translations and attitude) for the UAV landing mission dataset from Spin.Works, using NED navigation frame.

Our implementation allows us to vary some conditions, such as to choose the frame we use as reference (frame where we assume to know the pose) and the delta between frames where algorithms detect keypoints and estimate homographies. First, the motion recovery algorithm is tested using a control set of homographies, where we are using the SfM navigation data to generate the homographies and then use the homographies to move backwards and reconstruct the navigation states, hoping to get the same results. This procedure allows us to verify that the algorithms and functions are correctly implemented and evaluate how much information is lost in the process when using controlled, noise free data. Consequently, we discard causes of error, such as recursive least squares not converging, etc. Some deviations still occur between the original navigation dataset and the reconstructed dataset. These deviations are plotted in Figure 5.6. The magnitude of these deviations shows a maximum of 0.009 degrees in attitude (occurring in pitch), about 5 cm in horizontal displacement ($\sqrt{3.5^2 + 3.5^2}$) and sub-milimeter deviations in altitude. These deviations can be interpreted as a ”noise-floor” for the performance of the algorithm, i.e. deviations of this magnitude are interpreted as best achievable performance using the proposed system. Therefore, estimated deviations can be used as an evaluation metric of the quality of the homographies produced by different methods.

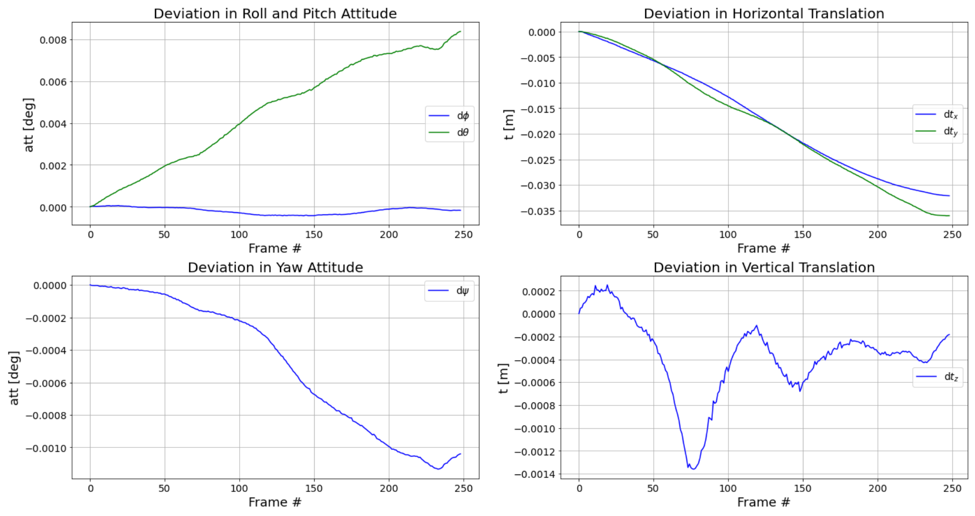
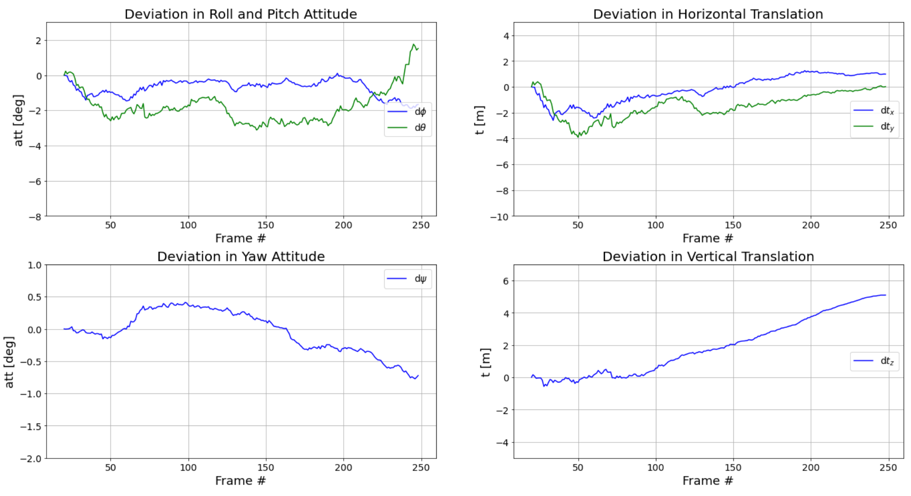


Figure 5.7: UAV landing pose recovery deviations, using homographies calculated directly from navigation data instead of image processing techniques. Frame 1 is used as reference.

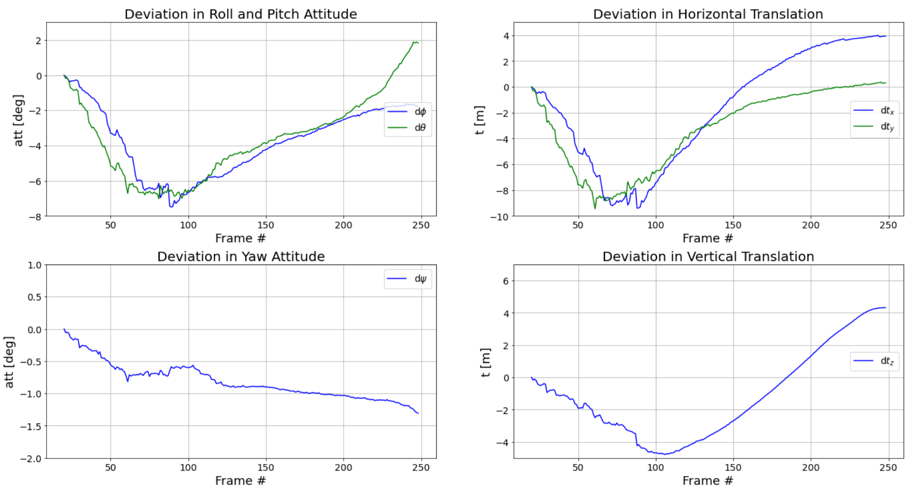
As explained in section 4.5, our tests are divided in two different pipelines. On one hand, we estimate homographies between a fixed interval of frames and compose transformations to get the matrix that relates the reference frame and the current one. On the other hand, we increase the interval between frames and estimate directly the homography between the reference and the current frame. Results are presented by plotting deviations of the estimates of the 6 degrees of freedom (3-axis translation and Euler angles) from the reference values (plotted in Figure 5.6).

Reference frame: 20, Delta between frames: 1

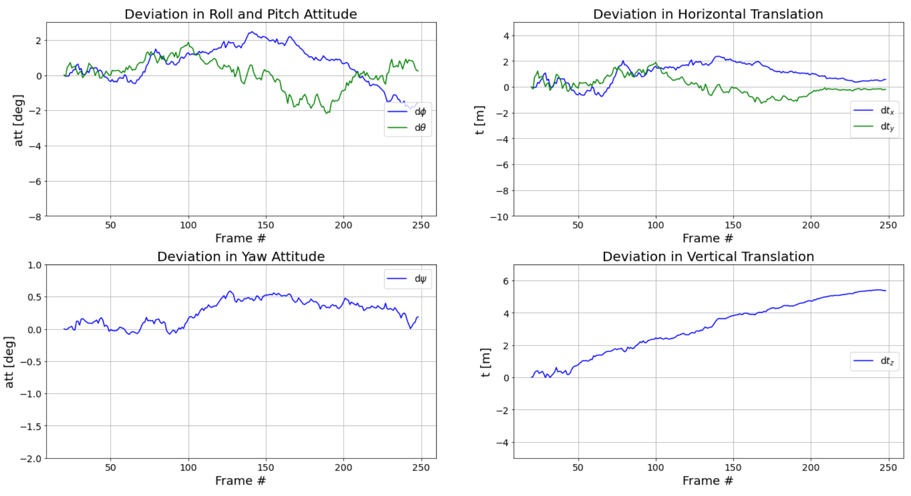
At a primary stage of evaluations, we defined one reference frame from the beginning of the sequence, e.g. frame 20, and the smallest interval possible, delta between frames equal to 1. The deviations between the estimated translations and attitudes by the three methods and the reference values are plotted in Figure 5.8. We can visually notice in plots that SIFT exhibits the worst performance among



(a) Harris Corners



(b) SIFT



(c) SuperPoint

Figure 5.8: UAV landing pose recovery deviations, using homographies calculated from image processing techniques with different features detectors. Frame 20 is used as reference and delta between frames is equal to 1. Inter-frame homography matrices are multiplied to get the transformation of the current frame with respect to the reference frame.

the three methods, since estimates done using SIFT keypoints are much deviated from the reference values. Harris Corners and SuperPoint perform similarly, although SuperPoint seems to be slightly more accurate, namely in pitch and y-translation.

	t$_x$	t$_y$	t$_z$	Roll	Pitch	Yaw
Harris	**0.942**	1.451	**1.962**	**0.706**	1.871	0.272
SIFT	3.885	3.308	2.589	3.751	3.736	0.811
SuperPoint	1.079	**0.572**	3.062	1.042	**0.692**	**0.270**

Table 5.2: Mean of the absolute value of the absolute deviations at each time step for UAV pose recovery using different feature detection methods. The values representing translations (t_x, t_y, t_z) are expressed in meters and the ones for attitude (Roll, Pitch, Yaw) are represented in degrees

	t$_x$	t$_y$	t$_z$	Roll	Pitch	Yaw
Harris	**0.577**	0.968	1.703	**0.455**	0.762	0.178
SIFT	2.618	3.015	1.415	1.938	2.041	0.276
SuperPoint	0.620	**0.547**	**0.177**	0.667	**0.547**	**0.177**

Table 5.3: Standard Deviation of the absolute value of the absolute deviations at each time step for UAV pose recovery using different feature detection methods.

	t$_x$	t$_y$	t$_z$	Roll	Pitch	Yaw
Harris	2.599	3.902	**5.096**	**1.884**	3.119	0.773
SIFT	9.396	9.447	4.785	7.492	7.025	1.307
SuperPoint	**2.371**	**1.902**	5.419	2.473	**2.182**	**0.588**

Table 5.4: Maximum Deviation of the absolute value of the absolute deviations at each time step for UAV pose recovery using different feature detection methods.

Quantitative results for deviations from reference of each state variable and detection method are summarized into tables 5.2, 5.3 and 5.4. Each column represents each of the variables that compose the state (3 for translation and 3 for attitude). The values in the table express the mean, standard deviation and maximum value, respectively, of the absolute value of the absolute deviations from the reference. The ones representing translations are expressed in meters and the ones for attitude are represented in degrees.

Results in the tables support our qualitative evaluation from the plots. SIFT deviations are o lot higher than SuperPoint and Harris Corners. The last two algorithms perform comparably with these conditions, but SuperPoint demonstrates lower values of maximum deviations in general. This conclusions comply with the first goal of our work, which was to introduce artificial intelligence into a vision-based system for lunar and planetary landing missions. We have seen that the method using deep learning reveals at least the same performance of classical algorithms, which indicates that it is worth to investigate this field. The following tests will focus on proving our conviction that SuperPoint's strength shows up, specially, when we use longer intervals of frames.

Reference frame: 1, Delta between frames: 20

We performed several experiments changing the reference frame and increasing the delta between frames until we get to the current condition, where frame 1 becomes the reference and algorithms only detect keypoints and estimate the homography between images that are separated by a period of 20 frames, starting from the key frame, until the end of the video sequence. Inter-frame homography matrices are multiplied to get the transformation of the current frame with respect to the reference frame before estimating the motion states, as in the last condition.

Results are presented in plots from figure 5.9. It is trivial to see that, using SuperPoint, the estimates are clearly better. Both Harris and SIFT fail badly on the final transitions of 20 frames, making estimates completely impractical. To maintain the readability of the plots, we have cut the last estimates from both classical algorithms and that is why the x-axis is shorter than in SuperPoint where the estimate remain accurate until the end of the sequence, since the deviations stay much lower than using Harris or SIFT. Besides that, we can compare the accuracy of the predictions in the first frames' transitions and the difference is visually noticeable.

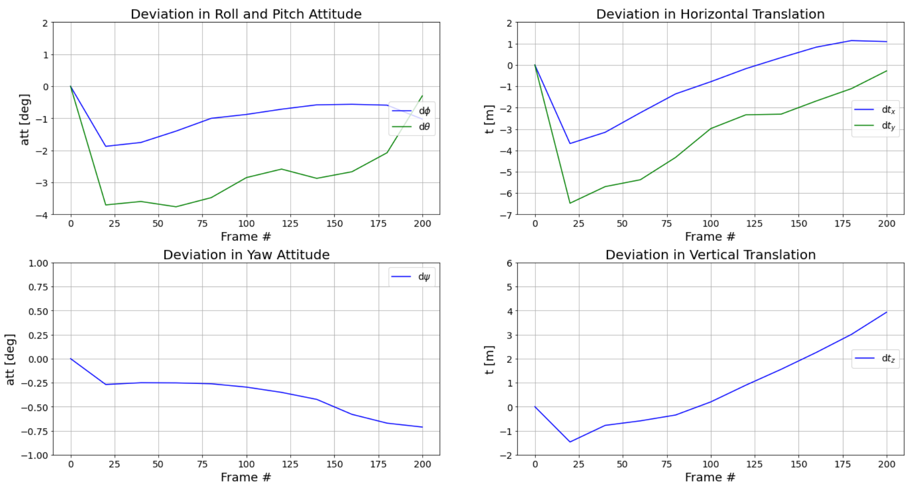
	t_x	t_y	t_z	Roll	Pitch	Yaw
Mean	**0.355**	**0.323**	**2.880**	**0.365**	**0.517**	**0.139**
Standard Deviation	**0.240**	**0.188**	1.732	0.670	**0.467**	0.229
Maximum	**0.689**	**0.570**	5.288	2.601	**1.715**	0.919

Table 5.5: Statistics for the absolute value of the absolute deviations during UAV pose recovery, using SuperPoint detector with delta between frames equal to 20. Results should be compared to the last line in Tables 5.2, 5.3 and 5.4. Bold means that values are better than using delta between frames equal to 1 from the last tables.

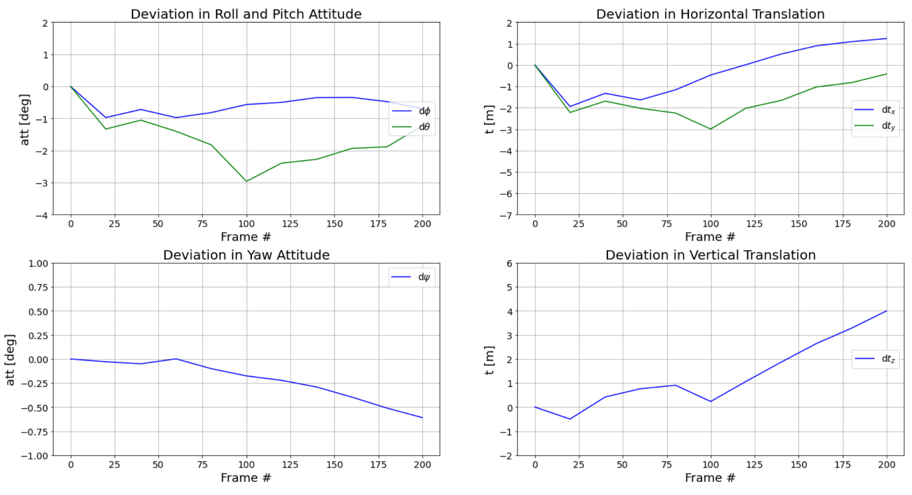
This time, we only show metrics of the absolute deviations for SuperPoint (table 5.5). Due to the reason mentioned above, the values of the deviations for the classical methods were enormous in the final transitions, which influenced a lot the statistical analysis and the values of the mean and standard deviation became worthless.

This method of relative navigation, similar to dead reckoning, is subject to cumulative errors. The composition of the homographies accumulates errors that are introduced in individual inter-frame estimations. For long periods of time, cumulative errors can lead to obsolete information about position and attitude of the aircraft. For that reason, it is important to increase the delta between frames, which reduces the number of estimations made and, consequently, the accumulation of errors. It can be used as an additional source of information to the perception system, like another sensor to introduce into the navigation filters to help rectify the estimates of pose state variables.

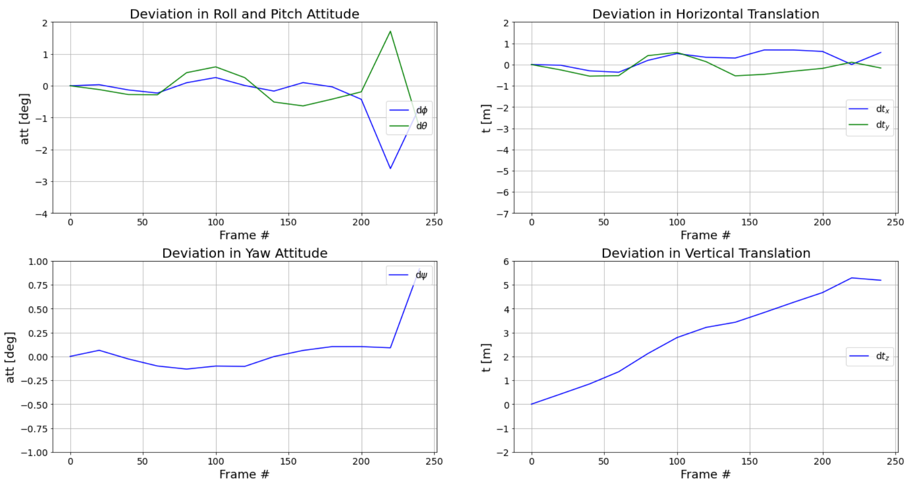
When we compare table 5.5 with the results of SuperPoint from tables 5.2, 5.3 and 5.4, we see that, in this condition of intervals, the absolute deviations of the estimates are clearly lower, with maximum values of less than 1 meter in horizontal translation, which supports the above description of the cumulative errors. Our conviction that the power of SuperPoint lies on the fact that its matchings stand valuable for longer periods of time into the video sequence is once again supported by this results.



(a) Harris Corners



(b) SIFT



(c) SuperPoint

Figure 5.9: UAV landing pose recovery deviations, using homographies calculated from image processing techniques with different features detectors. Frame 1 is used as reference and delta between frames is equal to 20. Inter-frame homography matrices are multiplied to get the transformation of the current frame with respect to the reference frame. Last estimates cut from both classical algorithms, as Harris and SIFT fail badly on the final transitions.

Limit for delta between frames

At second pipeline of our experiment, we aim to evaluate the longest period of frames in which the same features remain being correctly detected and matched, producing accurate estimates of pose states. Here, we do not rely on composition of inter-frame homographies. Instead, we perform directly the homography estimation between the reference frame features and the ones from the current frame. We perform experiments using different reference frames and count the number of frames until when deviations start exploding and the estimation become very unstable. Table 5.6 shows results for the three algorithms for different key frames. The numbers represent the period of frames starting from each reference, where estimations are accurate and stable.

Reference Frame	1	45	90	150	200
Harris Corners	70	53	44	29	17
SIFT	70	55	15	**44**	24
SuperPoint	**100**	**84**	**61**	42	**26**

Table 5.6: Number of frames until degradation of motion states estimates for UAV landing video sequence, from different reference frames

Once again our conviction is confirmed. SuperPoint actually shows better results for longevity of detected features and pose estimation between long periods between frames is significantly more stable and accurate using SuperPoint than classical methods, which indicates that the learned detector and descriptors are more robust and invariant to scale, translation and rotation changes in environments similar to lunar or planetary surface. Figure 5.10 expresses estimations with SuperPoint for the first 100 frames, using this method. As it was said before, this evaluation is very important because since we do not compose homographies, it does not contribute to the existence of cumulative errors and, once again, it could be another information, or another sensor, to help rectify the predictions to achieve the goal of building a precise vision-based navigation system for landing missions.

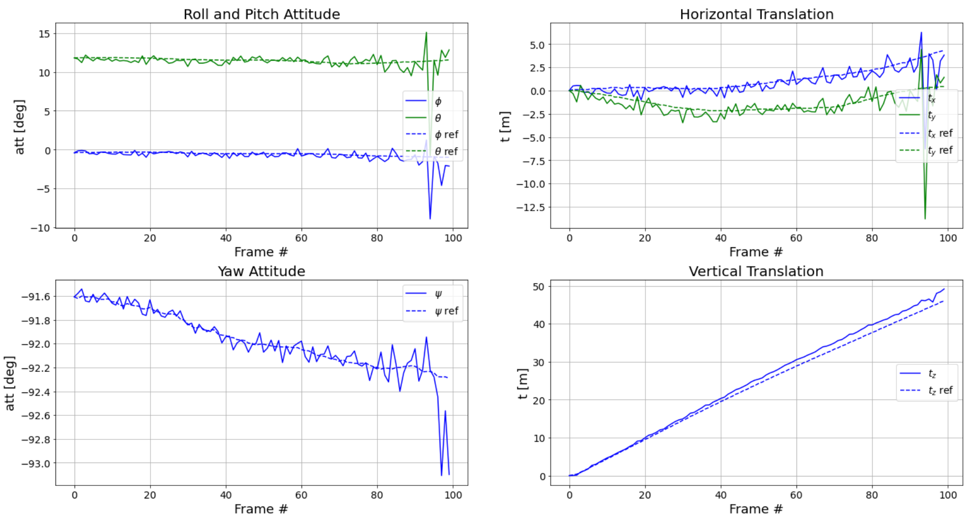


Figure 5.10: UAV landing mission - Pose estimation from direct SuperPoint matchings from reference frame (frame 1) until frame 100

The table exhibits one interesting detail. As the reference advances in the sequence, the number of frames until degradation of results becomes lower. This consequence can be explained by the area reduction over time. The region of the surface seen by the camera changes along the sequence of frames, so we decided to compute the ratio of the intersection between the region seen by the reference camera and the region seen by the current camera over the area of the region from the reference camera. This region is computed using the reference navigation data, projecting the camera versors into the surface and getting coordinates, in meters, of the four corners. The intersection ratio decreases as we go further from the reference. However, this decrease is a lot faster when the reference is further from the beginning of the video, which means, in that conditions, the region of the reference frame seen by the next frames decreases a lot faster and, as a consequence, it becomes harder to detect the same features and obtain valuable matches. Table 5.7 expresses these ratios, in percentage, using the same reference frames from table 5.6 and current frames by summing the number of frames until degradation of SuperPoint to the respective reference frames.

Reference / Current	1 / 100	45 / 129	90 / 151	150 / 192	200 / 226
Ratio	35%	35%	40%	36%	37%

Table 5.7: Ratio of the intersection over the reference, for reference and current frames calculated from SuperPoint results of table 5.6

It is worth to realize that SuperPoint starts producing unvaluable results when the ratio is below the values expressed in table 5.7, which are all similar. This information can be valuable for parameterizing the vision-based navigation system, as it allows us to predict, a priori, when the algorithm would fail to estimate reliable homographies and decide the delta between frames not by a fixed interval, but by thresholding this area ratio.

To sum up, SuperPoint exhibits the best performance among the three. We based our experiments on two assumptions: the camera model is linear and the surface is planar, which may justify the small deviations in the estimations. It is possible to take into account possible distortions from the real camera in order to try to reduce those errors. However, it was not explored on this Thesis, as our goal was to compare several detection algorithms under the same conditions and assumptions. Besides that, the proposed homography-based navigation system can be integrated into navigation filters so that the errors would easily shrink.

5.3 Spin.Works Moon Landing Mission Simulation

As stated in section 4.2.3, we also performed experiments in simulation datasets for lunar landing missions. The use of simulations arises due to the difficulty of getting real datasets for this problem. The main advantage is that we can easily vary the conditions, such as the trajectory, the landing site or the camera model and we have a completely controllable environment where we known for sure the motion states of the aircraft. Consequently we can evaluate our algorithms and tune their parameters simulating the conditions of a real landing mission in order to prepare the vision-based navigation system for real applications.

We have position and attitude of the camera with respect to the ground with a time-step of 0.1 seconds, plotted in Figure 5.11. Synthetic images with size 512×512 are generated using a pin-hole camera model, at 10Hz, for all the descent, which forms a dataset with 950 frames.

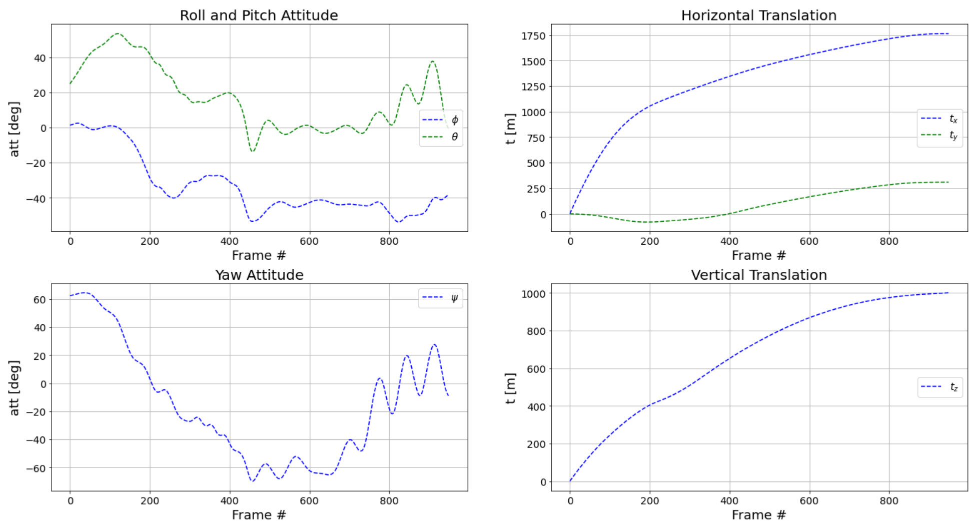


Figure 5.11: Reference navigation data (translations and attitude) for the Moon landing simulation dataset from Spin.Works, using NED navigation frame.

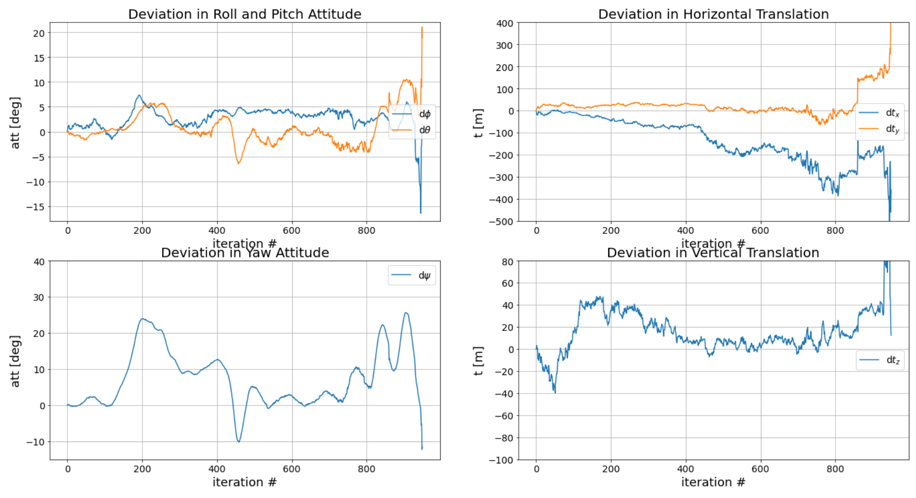
5.3.1 Navigation States Estimation

We skip our experiments to the ones involving the estimation of pose during the video sequence that represents the trajectory of an aircraft on a moon landing scenario, as it is the ultimate focus of this investigation. We also started by evaluating the estimates of the motion states from the homographies computed from the states themselves, which gave us residual errors, such as a maximum deviation of 3 meters in a 1.5 kilometer horizontal translation. This result validates our implementation and we proceed the evaluations using homographies calculated by the image processing techniques. Once again we change conditions, such as the reference frame and the delta between frames.

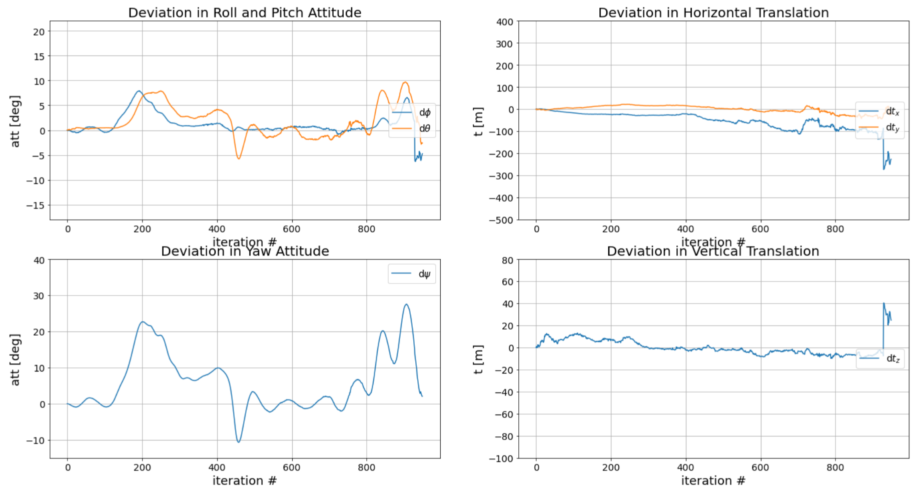
Reference frame: 1, Delta between frames: 1

Once again, we started by testing the methods with the simplest scenario of using the first frame as reference and composing the inter-frame homographies (calculated with a delta between frames equal to 1) to obtain the relation between the current and reference frames. We also the deviations of the estimated translations and attitude from the real values defined by the simulated trajectory of an EDL mission to the Moon, Figure 5.12.

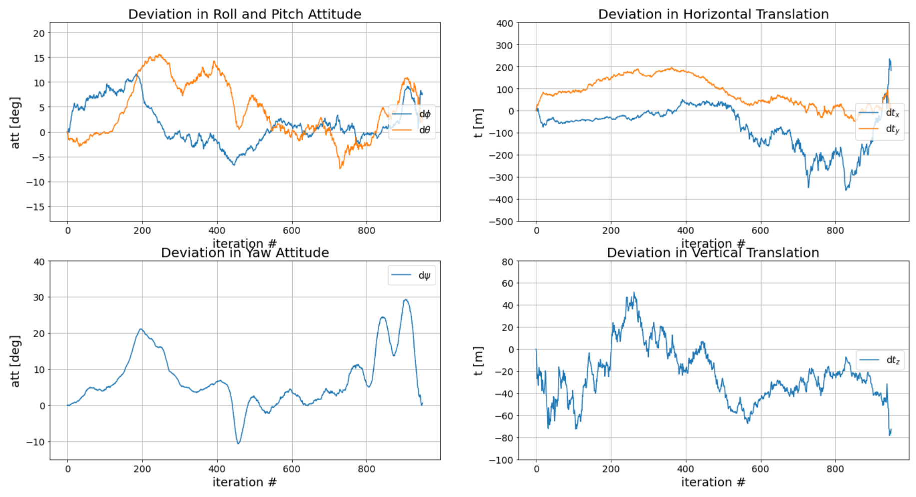
Results show that all three methods are able to retrieve motion states fairly well. However, SIFT exhibits the best performance among all, as the estimates are far closer to the references, namely the translation ones. SIFT detections have sub-pixel precision, while both Harris Corners and SuperPoint keypoints are defined at integer coordinate locations. Since this dataset has a larger number of frames,



(a) Harris Corners



(b) SIFT



(c) SuperPoint

Figure 5.12: Moon landing simulation pose recovery deviations from reference states, using homographies calculated from image processing techniques with different features detectors. Frame 1 is used as reference and delta between frames is equal to 1. Inter-frame homography matrices are multiplied to get the transformation of the current frame with respect to the reference frame.

there are a lot more transitions to estimate, which may result result in more cumulative errors as we go further from the reference. Besides that, translations are a lot higher than the ones from the UAV dataset. Sub-pixel precision may reduce the errors, which gives SIFT a great advantage and supports the results.

To tackle this problem and make SuperPoint competitive to SIFT, we tested to introduce the function *cornerSubPix* from OpenCV to the corners detected by SuperPoint in order to refine corner locations and allow sub-pixel precision. Results are presented in figure 5.13. As we expected, estimates are far better now and they can reproduce well both attitude and translation until the end of the trajectory, demonstrating less accumulated errors at the end than SIFT, namely in the x-axis from horizontal translation.

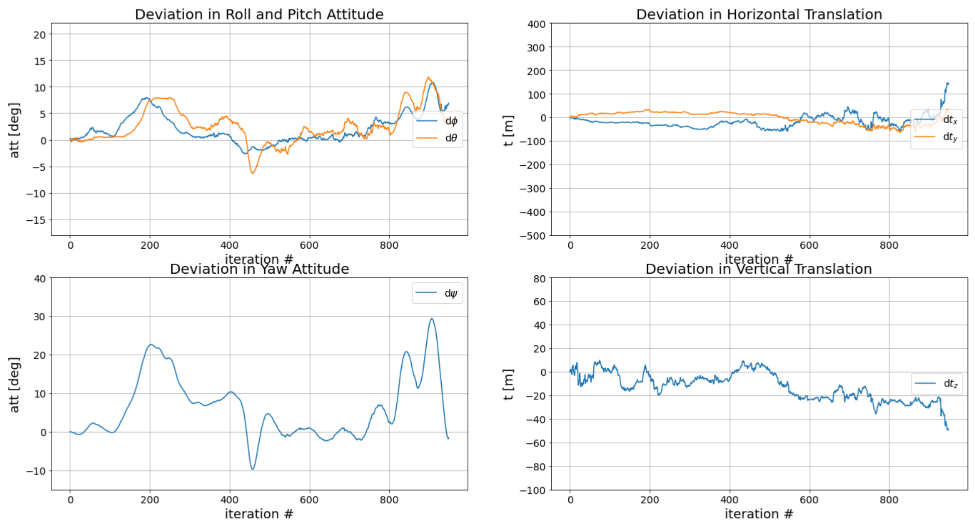
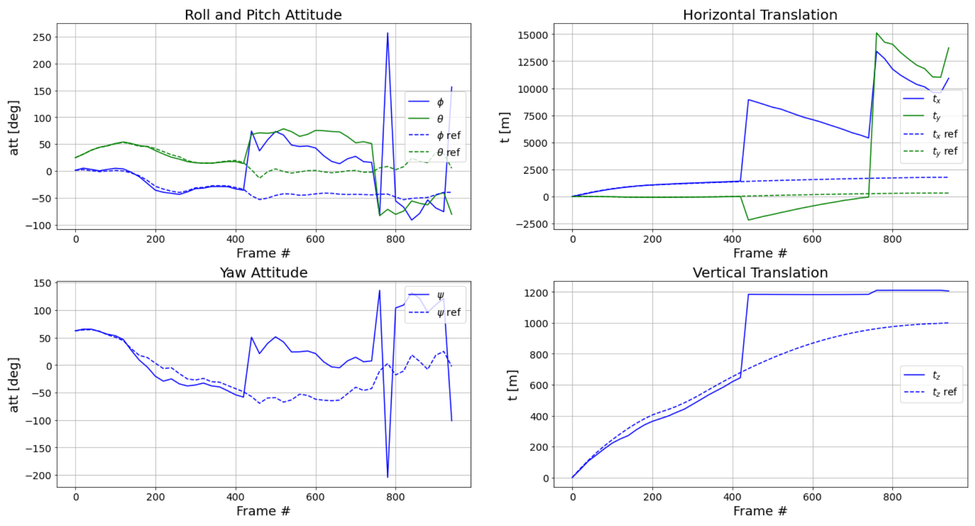


Figure 5.13: Moon landing simulation pose recovery deviations from reference states, using homographies calculated from SuperPoint-cv.cornerSubPix detections. Frame 1 is used as reference and delta between frames is equal to 1. Results should be compared to Figure 5.12

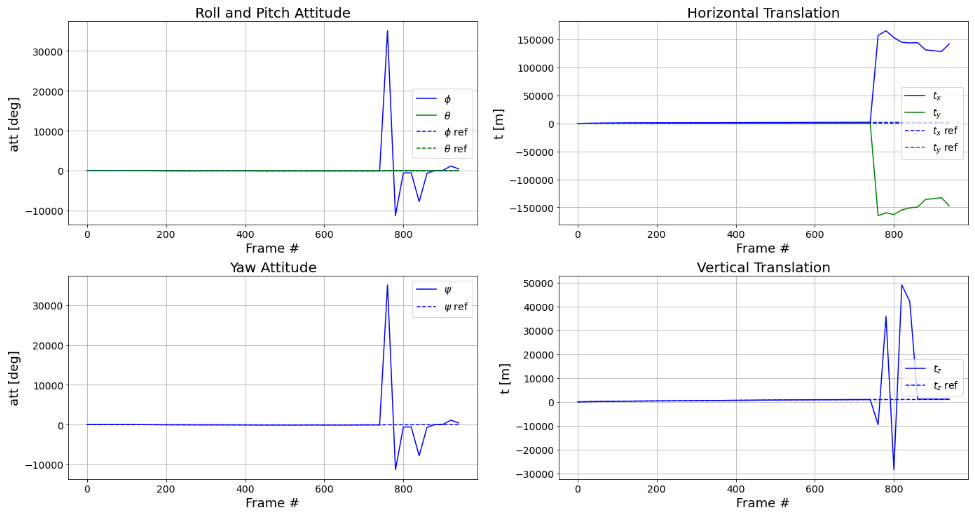
As we know from the last dataset, SuperPoint does not bring considerable advantages when we evaluate estimations from consecutive frames or small intervals between frames, but it is also important to notice that at least it performs comparably to the classical algorithms under these conditions and for the real application of developing a vision-based system for space missions.

Reference frame: 1, Delta between frames: 20 / 30

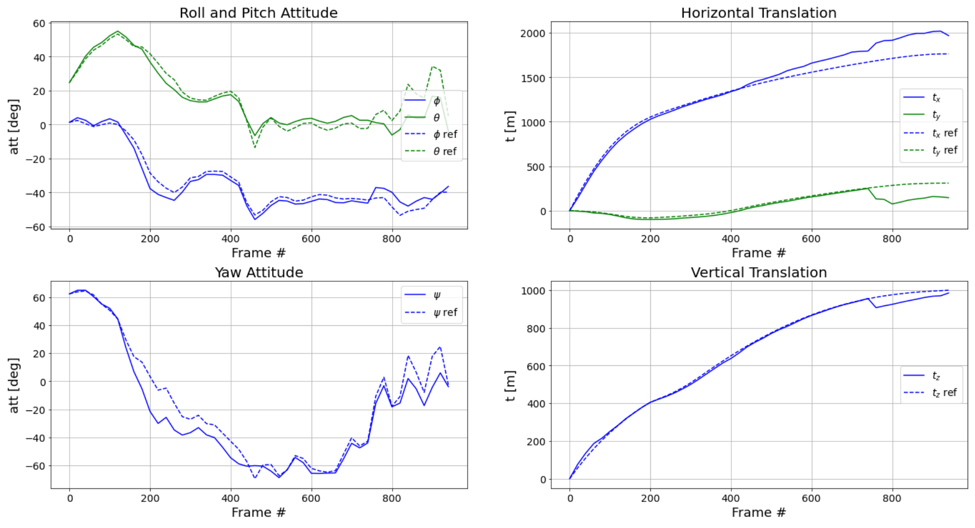
Next, our experiments concern larger intervals between frames. We have increased delta and significant differences arise for intervals of 20 or 30 frames. Results are represented in figures 5.14 and 5.15 for delta equal to 20 and 30, respectively. Once again, we prove our conviction that SuperPoint brings enormous advantages when we increase the period between the frames, as its detections and descriptors are stronger and more invariant than those using classical methods. Estimates using SuperPoint remain valuable until the end of the sequence using both delta equal to 20 and 30. The same does not happen with Harris Corners or SIFT. SIFT performs well until frame 750. From then on, it clearly fails to estimate the homography, which is reflected by the big deviation in the final transitions and both figures.



(a) Harris Corners

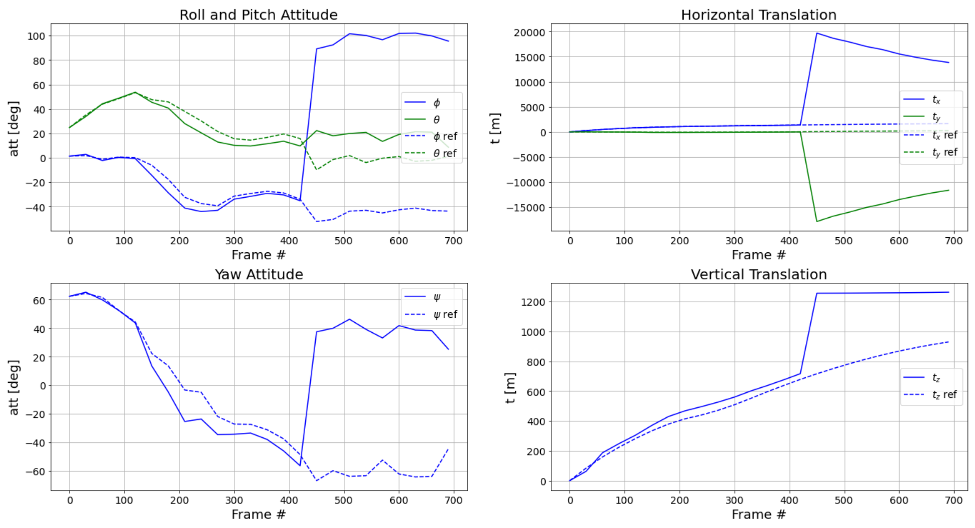


(b) SIFT

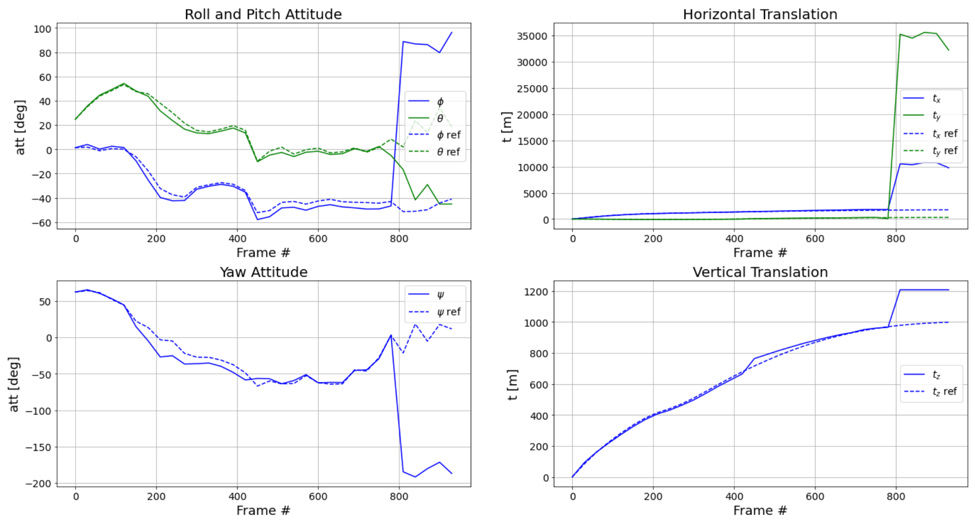


(c) SuperPoint + cv.cornerSubPix

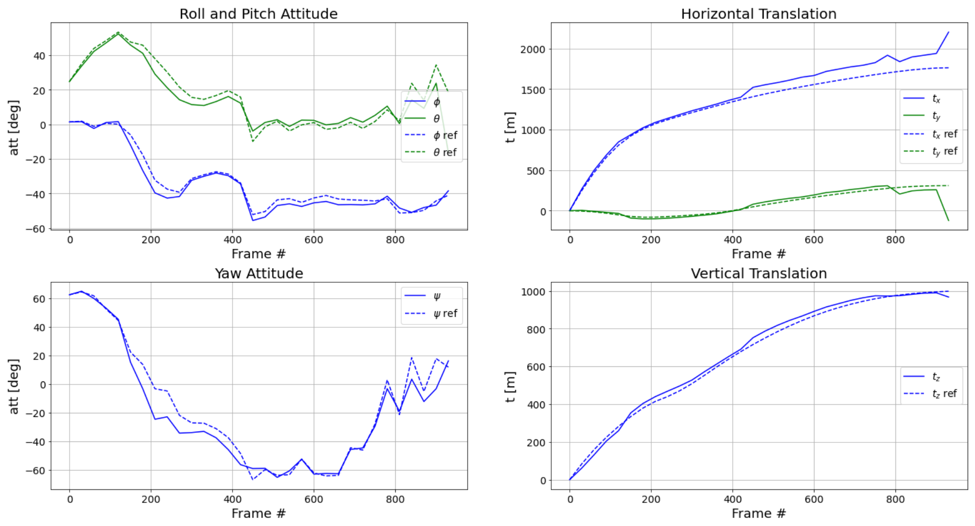
Figure 5.14: Moon landing simulation pose recovery, using homographies calculated by image processing techniques, with frame 1 as reference and delta between frames equal to 20



(a) Harris Corners



(b) SIFT



(c) SuperPoint + cv.cornerSubPix

Figure 5.15: Moon landing simulation pose recovery, using homographies calculated by image processing techniques, with frame 1 as reference and delta between frames equal to 30

Harris Corners fails a lot earlier in the sequence. Figure 5.14 shows an enormous deviation after frame 400 and another at about frame 750. The plot was cut to maintain readability, but deviation ”steps” keep on happening and getting bigger until the end of the video. We can confirm that SuperPoint is a lot more stable, as estimations did not explode in any condition from these experiments.

At this time, we did not present the results for the limit for delta between frames because it is not much higher than 30 with several references, specially from the beginning. This trajectory is very different from the UAV one. Here, we have a more accelerated scenario, horizontal translations are higher and the aircraft incidence angle changes a lot during the simulation flight, which makes the intersection area ratio to decrease a lot faster, making it impossible to find the same features, since camera stops being looking to the same region a lot faster. This reduction in the intersection ratio is expressed in figure 5.16, using frame 1 and 400 as reference.

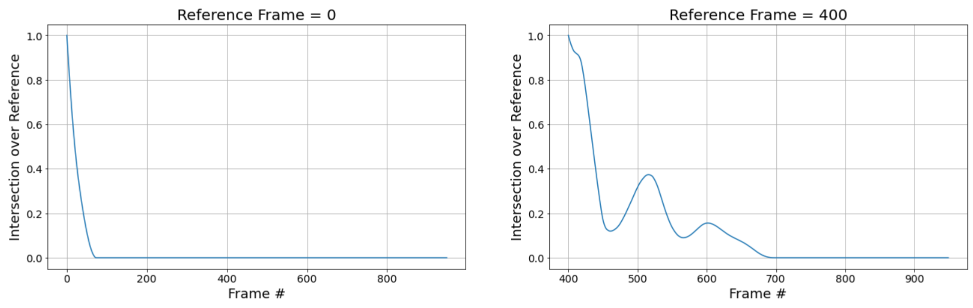


Figure 5.16: Area reduction from reference frame region seen by the camera

Results from simulation sequences are important because they are much easier to create and to change conditions in order to prepare the algorithms for the environment and dynamics the aircraft will come across when on a real mission. It allows to predict algorithm’s behavior a prior and to parameterize the system to the specific conditions of the mission. Despite our initial doubts about the performance of the vision based navigation, since simulation terrain texture could be less embossed and harder for feature detection, SuperPoint also proves to be very accurate and a good alternative to classical methods, specially for matchings in longer intervals of frames.

5.4 Perseverance Rover’s Descent and Touchdown on Mars

TRN has two distinct parts: first, the algorithm tries to do terrain matching with a surface map stored in memory, which results in an absolute navigation solution; second, after getting that solution, it moves to relative navigation until it reaches the landing site. Our work addresses the relative vision-based navigation. We do not have the actual navigation data, and the reduced set of information regarding the camera specifications, the reference altitude associated with the key frames and the whole video sequence as a whole, forced us to make several educated guesses during the process of reconstructing the navigation states using SfM methods. Moreover, the quality markers of the SfM results are not so good, indicating that the SfM process found a lot of inconsistencies in the data and, therefore, the quality of the results is questionable. All this uncertainty makes the available navigation results from SfM only valid for qualitative analysis and not quantitative. Still, the novelty of the data justified it being included here.

Reference frame: 50, Delta between frames: 10

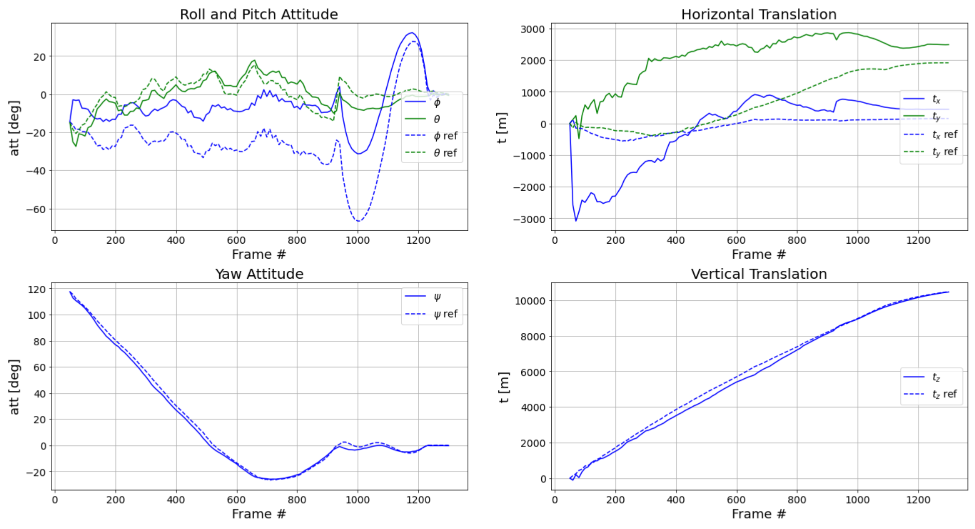


Figure 5.17: Perseverance landing pose recovery, using homographies calculated by SuperPoint keypoints, with frame 50 as reference and delta between frames equal to 10

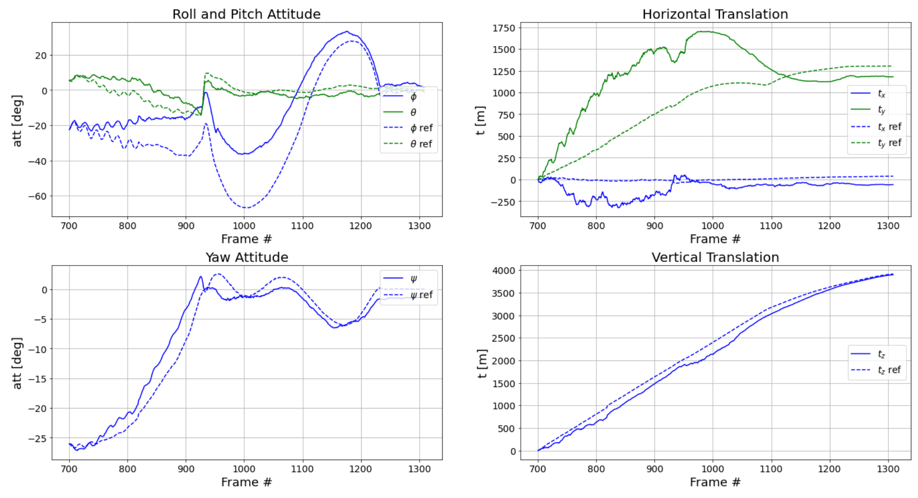
The first frames occur while the heat shield is separating, so it appears in the images. Therefore, we have selected frame 50 to be our reference and estimated homographies with a delta equal to 10 until the end of the sequence. Motion states estimated using SuperPoint are represented in figure 5.17, having a very accurate agreement in attitude and vertical translation, specially in terms of trend following, i.e., neglecting the bias and focusing on the dynamic behavior. Harris Corners is not represented because the results were completely damaged and were considered not significant and SIFT did not detect any point in frame 50 or other frames around, which reinforces our conviction that SuperPoint is a good alternative to classical methods.

Unfortunately, the horizontal translation, which is the most important result for GNC given that the remaining states are being well observed by other sensors aboard the spacecraft, show a poor agreement with the reference from SfM.

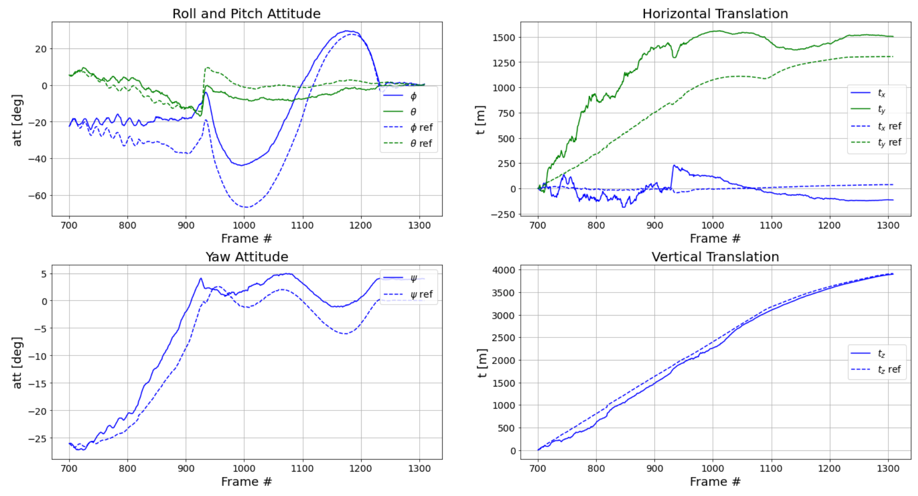
Reference frame: 700, Delta between frames: 1 / 10

From NASA's plan for the EDL mission, we get that TRN starts at about 4 kilometers above the surface. In a mission similar to this one, our algorithms would work at that phase. So, we decided to perform experiments starting at that point. From the position data we use as ground truth, we found that the rover is at an altitude of 4 kilometers at about frame 700. So, our reference now is frame 700 and we evaluate using deltas equal to 1 and 10.

Only SuperPoint and Harris Corners produced valuable results, that are represented in figures 5.18 and 5.19 for deltas equal to 1 and 10, respectively. SIFT clearly failed a lot transitions along the se-



(a) Harris Corners



(b) SuperPoint

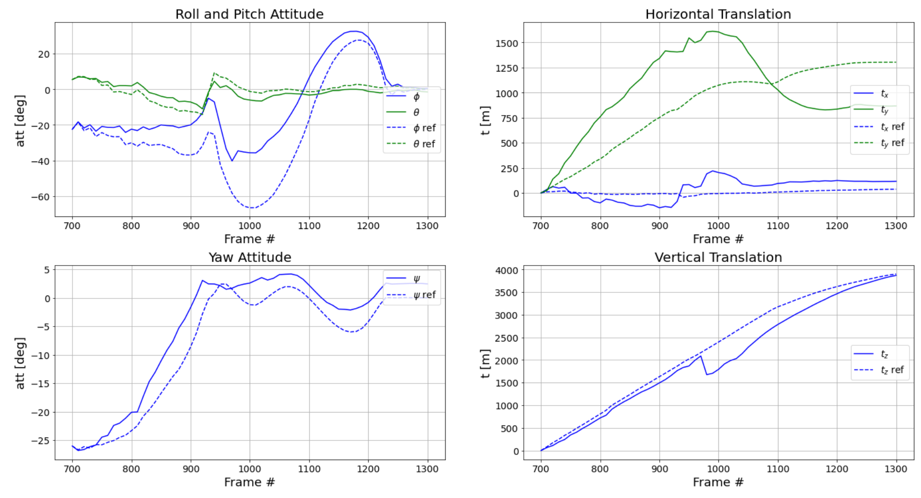
Figure 5.18: Perseverance landing pose recovery, using homographies calculated by image processing techniques, with frame 700 as reference and delta between frames equal to 1

quence, in both conditions, which made results insignificant, that is why we do not present them.

SuperPoint and Harris Corners show similar results, but SuperPoint is getting slightly lower deviations, specially in translations and in the condition of delta equal to 10, as we espected. Again, SuperPoint reveals to be more powerful than conventional detectors for larger transitions as it happened in the other datasets.

We consider these results extremely promising, since there are a lot of uncertainties in the whole process. We do not have the exact navigation states from the mission and the values we use as references are subject to errors. When we choose the key frame, we assume to know the motion states, such as attitude and altitude above ground. We have noticed during our experiments that small changes in that altitude contribute to significant differences, which allows to conclude that these processes are hugely sensitive. Hence, small inaccuracies in altitude can explain the deviations we see, namely in translation.

Another factor that could be causing errors is the assumption of a linear model to the camera, which do not take into account the possible distortions and can influence the results. However, this assumption


(a) Harris Corners

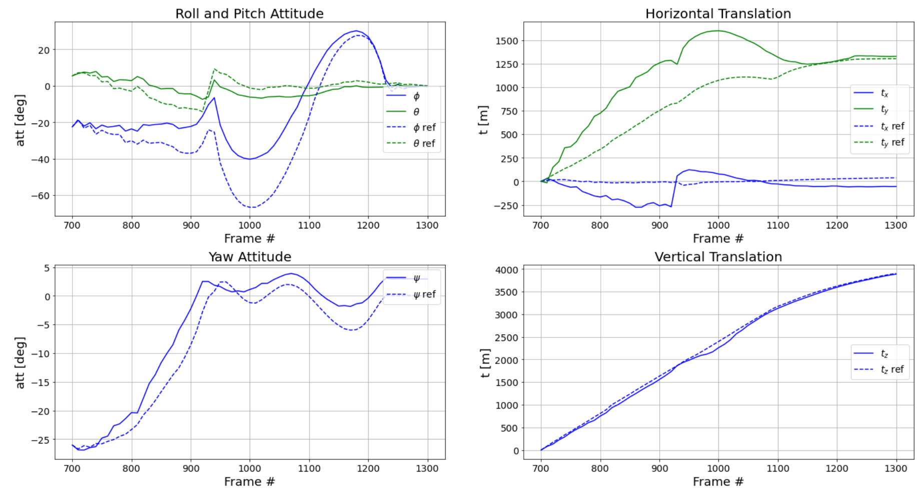

(b) SuperPoint

Figure 5.19: Perseverance landing pose recovery, using homographies calculated by image processing techniques, with frame 700 as reference and delta between frames equal to 10

is enough for our goal, that is to compare detection algorithms. These refinements of the estimations go beyond the purpose of our work.

We did not go further in detail on this dataset because there are a lot of information that can be not 100% tunned, but the outcomes we got are really promising. Despite all uncertainties we consider this evaluation very important due to the fact that this is the most recent and representative dataset for our problem and the satisfactory results points that we are following the right path and this is still an open field for investigation.

5.5 Discussion

The first goal of this extensive experimental campaign was to confirm that the promising results of CNNs on almost every CV related task were also applicable to the motion states estimation task using a vision-based navigation system during spacecraft landing missions, and it was accomplished. Results on every dataset shown that SuperPoint detections could at least perform similar estimations as the ones using classical methods.

Besides that, this investigation have shown that an homography-based vision system could largely benefit from SuperPoint as its detections are stronger and keep on being correctly matched along larger intervals of frames and larger viewpoint and illumination changes, which gives very useful information to the navigation system. According to classical methods, Harris Corners have shown better results under some conditions, e.g., using the Mars landing dataset, while SIFT had better performance for the Moon simulation dataset. Nevertheless, SuperPoint kept being consistent and showing satisfactory results for all the scenarios performed, presenting always the same or higher accuracy estimating motion states, which makes it a more stable and reliable method that gives more confidence when used for real world applications.

Chapter 6

Conclusions

6.1 Achievements

Autonomous navigation is an area of great interest among the scientific community and space industry is not an exception. To accomplish autonomous navigation, the spacecraft needs to know where it is (perception). Specially in space missions, where GPS is not available, cameras proved to be decisive for precision navigation. That made vision-based navigation essential for planetary or lunar landing missions.

Concurrently, Artificial Intelligence is becoming hugely popular and recent advances in hardware capabilities allowed DL models to achieve unimaginable performances in tasks that were done by humans in the past. Computer Vision is one of those fields, since CNNs are replacing conventional hand-engineered methods in almost every task, which induced us to do this investigation on combining these domains.

Therefore, we developed a comparison on classical solutions to the problem of estimating motion states of a spacecraft using camera, during a landing mission, against a solution based on a DL algorithm. To achieve that, an extensive study was performed comprising the topics described on the first chapters. Computer vision concepts such as feature detection, homography estimation, camera pinhole model, methods for the extraction of 3D information from the planar homography, DL and CNNs were studied.

A complete framework to evaluate the performance of any feature detector for the task of estimating position and attitude of an spacecraft along a landing trajectory, using a sequence of images, acquired by a mounted camera, was implemented. Two classical feature detectors (Harris Corners and SIFT) and one using deep learning (SuperPoint) were used to perform evaluations on that framework. Tests were made in three video sequences. The first one from a UAV landing in a Mars representative terrain, from Spin.Works. The second from a simulated landing trajectory on a synthetic lunar surface, also from Spin.Works and, finally, the video sequence published by NASA from Perseverance rover's landing on Mars.

Results have shown that the deep learning detector achieved at least the same navigation performance on all datasets, as the conventional methods, and that it even outperforms them under some conditions. SuperPoint proved to bring enormous advantages in detecting stronger keypoints and descriptors that remain being detected and correctly matched on much more images of the same scene along the video sequence. That is reflected on much better estimates of relative pose between much higher intervals between frames. This quality is of great importance as it can be used as complementary information to the relative pose between successive frames, in order to refine the estimates and reduce cumulative errors that arise from successive small errors from inter-frame predictions. Also, the outcomes show that even using the assumptions of a linear camera model and a planar surface, results are quite satisfactory. Finally, SuperPoint was trained with general images completely outside the domain, but even so it produces better results than hand-engineering methods.

These promising results from SuperPoint suggest that this problem is another task where convolutional neural networks are outperforming conventional computer vision methods and that this field is worth of investigation. Space exploration will be revolutionized in the next years by the power of these AI techniques.

6.2 Future Work

This problem is still an open field for investigation and the next planetary missions are expected to largely rely on the improvements that could be made by the scientific community until then. The first, and probably most obvious, future work is to implement this solution in hardware to evaluate real-time performance and integration into the navigation systems during a simulated planetary landing using UAVs on representative surfaces, similar to the one used on the UAV video sequence from Spin.Works.

More research could be done to integrate geometric constraints and the camera pose estimation pipeline into the trainable network, so that it optimizes the detected keypoints supervised by the ultimate goal of relative pose estimation. Besides that, it would be interesting to investigate the introduction of temporal information into the homographies estimation in order to try to reduce cumulative errors. Also, it would be important to study the introduction of sub-pixel precision into the detector part of the network.

Another important aspect to improve is computational efficiency of the deep learning network. The VGG-style backbone has lot of standard 2D convolutional layers that could be converted to most efficient building blocks as the ones proposed by MobileNets.

Bibliography

[1] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *Procedings of the Alvey Vision Conference 1988*. Alvey Vision Club, 1988. doi: 10.5244/c.2.23. URL `https://doi.org/10.5244%2Fc.2.23`.

[2] J. Shi and Tomasi. Good Features to Track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994. doi: 10.1109/CVPR.1994.323794.

[3] D. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999. doi: 10.1109/ICCV.1999.790410.

[4] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL `https://doi.org/10.1023/B:VISI.0000029664.99615.94`.

[5] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded Up Robust Features. In A. Leonardis and A. Bischof, Horstand Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33833-8.

[6] E. Rosten and T. Drummond. Machine Learning for High-Speed Corner Detection. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33833-8.

[7] E. Rosten, R. Porter, and T. Drummond. Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, 2010. doi: 10.1109/TPAMI.2008.275.

[8] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary Robust Independent Elementary Features. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15561-1.

[9] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, London, 2^{nd} edition, 2020.

[10] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011. doi: 10.1109/ICCV.2011.6126544.

[11] B. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI). volume 81, 04 1981.

[12] R. H. . A. Zisserman. *Multiple View Geometry*. Cambridge University Press, 2^{nd} edition, 2004.

[13] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[14] T. Mitchell. *Machine Learning*. McGraw-Hill, 1^{st} edition, 1997.

[15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[17] A. Newell, K. Yang, and J. Deng. Stacked Hourglass Networks for Human Pose Estimation. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 483–499, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46484-8.

[18] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis. 6-dof object pose from semantic keypoints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2011–2018, 2017. doi: 10.1109/ICRA.2017.7989233.

[19] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. volume 9351, pages 234–241, 10 2015. ISBN 978-3-319-24573-7. doi: 10.1007/978-3-319-24574-4_28.

[20] L. Pasqualetto Cassinis, R. Fonod, E. Gill, I. Ahrns, and J. Fernandez. CNN-based pose estimation system for close-proximity operations around uncooperative spacecraft. 01 2020. doi: 10.2514/6.2020-1457.

[21] K. Yi, E. Trulls, V. Lepetit, and P. Fua. LIFT: Learned Invariant Feature Transform. volume 9910, pages 467–483, 10 2016. ISBN 978-3-319-46465-7. doi: 10.1007/978-3-319-46466-4_28.

[22] D. DeTone, T. Malisiewicz, and A. Rabinovich. SuperPoint: Self-Supervised Interest Point Detection and Description. pages 337–33712, 06 2018. doi: 10.1109/CVPRW.2018.00060.

[23] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. HPatches: A benchmark and evaluation of handcrafted and learned local descriptors. pages 3852–3861, 07 2017. doi: 10.1109/CVPR.2017.410.

[24] Y. Ono, E. Trulls, P. Fua, and K. M. Yi. LF-Net: Learning Local Features from Images. In *NeurIPS*, 2018.

[25] A. Barroso, E. Riba, D. Ponsa, and K. Mikolajczyk. Key.Net: Keypoint Detection by Handcrafted and Learned CNN Filters. pages 5835–5843, 10 2019. doi: 10.1109/ICCV.2019.00593.

[26] M. Dusmanu, I. Rocco, T. Pajdla, M. Pollefeys, J. Sivic, A. Torii, and T. Sattler. D2-Net: A Trainable CNN for Joint Description and Detection of Local Features. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8084–8093, 2019. doi: 10.1109/CVPR.2019.00828.

[27] J. Revaud, P. Weinzaepfel, C. R. de Souza, N. Pion, G. Csurka, Y. Cabon, and M. Humenberger. R2D2: Repeatable and Reliable Detector and Descriptor. *ArXiv*, abs/1906.06195, 2019.

[28] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1.

[29] I. Rocco, R. Arandjelovic, and J. Sivic. Convolutional Neural Network Architecture for Geometric Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1–1, 08 2018. doi: 10.1109/TPAMI.2018.2865351.

[30] W. Shi, J. Caballero, F. Huszár, J. Totz, A. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. 06 2016. doi: 10.1109/CVPR.2016.207.

[31] W. Förstner and E. Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. *Proc. ISPRS intercommission conference on fast processing of photogrammetric data*, 6(2):281–305, 1987.

[32] E. M. . M. Vargas. Deeper understanding of the homography decomposition for vision-based control. *[Research Report] RR-6303, INRIA*, 2007.