
The University Of Sheffield.

# Monocular Sparse Snapshot Localisation And Homing For Small Fixed-Wing UAVs

*Author:*
Elias LATTASH

*1st Supervisor:*
Dr. Charith ABHAYARATNE

*2nd Supervisor:*
Dr. Michael MANGAN

*A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy*

*in the*

Department of Electronic and Electrical Engineering
Faculty of Engineering
The University of Sheffield

8th November 2021

# Declaration of Authorship

I, Elias LATTASH, declare that this thesis titled, "Monocular Sparse Snapshot Localisation And Homing For Small Fixed-Wing UAVs" and the work presented in it are my own. I confirm that:

- The work on this thesis was done wholly while being a PhD student at the University of Sheffield.
- Where I have consulted the published work of others, it is always clearly attributed.
- I have acknowledged all main sources of help.
- I declare that this thesis is entirely my own work.

*"always giving thanks to God the Father for everything"*

EPHESIANS 5:20

# Abstract

This research is an investigation into developing computer vision based mechanisms to address the safety related concerns connected with piloting a small fixed-wing Unmanned Aerial Vehicles (UAVs). Many consumer fixed-wing UAVs do not come equipped with accurate Global Navigation Satelite Systems GNSS technology. Professional drones which do, are prone to jamming, spoofing and other problems, and reliance on the skill of the remote pilot is not failure proof either. Therefore having an automated safety mechanism is of vital importance, especially on fixed-wing UAVs given their high speeds and high flying altitudes. Many works have proposed localisation solutions on rotor-wing UAVs, but few have addressed the problems related to fixed-wing UAVs, or taken advantage of resources uniquely available to them. In this work: 1) An understanding is formed of traditional visual odometry (VO) methods and their effectiveness on fixed wing UAVs. Suitable motion estimation algorithms are proposed, evaluated and their disadvantages concluded. 2) A learning based visual localisation system is introduced using semantic segmentation and particle filtering for the absolute localisation of a fixed-wing UAV. The solution relies on sparse monocular top-down imagery captured by the UAV, and takes advantage of an onboard Google Earth map. The system runs in real-time on a modest CPU and achieves an accuracy that is close to recent state-of-the-art methods, with the added advantage of global localisation capacity. 3) The learning based system is extended to cover the motion estimation part of the system. A deep learning rigid registration CNN is proposed for the registration of segmented images. The proposed localisation system can be used to enable homing on small UAVs enabling beyond visual line of sight operations, it can add an extra layer of redundancy to fixed-wing navigation systems. It can also be used on GNSS denied environments such as other planets.

# Acknowledgements

Thanks be first to God Almighty. I am sincerely grateful to my first supervisor Doctor Charith, without whom I would not have been given the opportunity to do this study. I thank God for his paternal care which he never withheld from me nor from my other group members at times of difficulty. I am also grateful to my second supervisor Doctor Michael for all of his helpful comments especially during the first two years when the research was taking shape. I am grateful to the Engineering and Physical Sciences Research Council for the Doctoral Training Award scholarship they generously offered me. I remain indebted to the charitable hospitality of this country which welcomed and protected me. And finally, I thank God for Father Alam's guidance and prayers throughout this journey.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AT** | Aerial Triangulation |
| **BA** | Bundle Block Adjustment |
| **BoW** | Bag of Words |
| **BVLOS** | Beyond Visual Line Of Sight |
| **CAA** | Civil Aviation Authority |
| **CML** | Concurrent Mapping and Localisation |
| **CNN** | Convolutional Neural Network |
| **DLT** | Direct Linear Transform |
| **DoF** | Degree of Freedom |
| **EKF** | Extended Kalman Filter |
| **FPS** | Frames Per Second |
| **GLONASS** | Global Navigation Satellite System |
| **GNSS** | Global Navigation Satellite System (generic) |
| **GPS** | Global Positioning System |
| **ICP** | Iterative Closest Point |
| **IMU** | Inertial Measurement Unit |
| **KF** | Kalman Filter |
| **LIDAR** | Light Detection And Ranging (as in *RADAR*) |
| **LUT** | Lookup Table |
| **MAE** | Mean Absolute Error |
| **MAP** | Maximum A Posteriori |
| **MLE** | Maximum Likelihood Estimation |
| **MSE** | Mean Square Error |
| **PF** | Particle Filter |
| **PI** | Path Integration |
| **PnP** | Perspective from n Points |
| **PPK** | Post Processing Kinematics |
| **RANSAC** | Random Sample Consensus |
| **RTK** | Real Time Kinematics |
| **SD** | Standard Deviation |
| **SfM** | Structure from Motion |
| **SGT** | Spectral Graph Theory |
| **SLAM** | Simultaneous Localisation And Mapping |
| **SSD** | Sum of Square Differences |
| **SUA** | Small Unmanned Aircraft |
| **SVD** | Singular Value Decomposition |
| **ToF** | Time of Flight |
| **VLOS** | Visual Line Of Sight |
| **VO** | Visual Odometry |

# List of Symbols/Units

| | |
|---|---|
| $^\circ$ | Degrees |
| $kg$ | Kilo Grams |
| $km$ | Kilo Meters |
| $m$ | Meters |
| $W$ | Watts |
| $coords$ | coordinates |
| $d$ | distance (m) |
| $\psi$ | Pitch rotation angle ($^\circ$) |
| $\phi$ | Roll rotation angle ($^\circ$) |
| $\theta$ | Yaw rotation angle ($^\circ$) |

# Notes to Reader

In this thesis, photographs from two main datasets are used in figures throughout the text, which are ISPRS Potsdam [3] and Sensefly Merlischachen [2]. In addition, many terms are used interchangeably to refer to the same object:

- The following terms are used to refer to a monocular RGB image in general, or more specifically a top-down aerial photo, unless otherwise noted: *frame(s), snapshot(s), photo(s), photograph(s)* and *[monocular] image(s)*.
- The terms *Unmanned Aerial Vehicle (UAV)* and *drone*, are used to refer to a small UAV (under 20kg according to the Civil Aviation Authority's classification [8]). It is also referred to as Small Unmanned Aircraft (SUA), which is fixed-wing unless otherwise noted.
- The terms *robot* and *agent* are used in a general sense to refer either to a vehicular or aerial robot, but not necessarily a UAV.

I would like to dedicate this
thesis to my beloved fiancée,
who has the heart of a child,
Yue.

# Chapter 1

# Introduction

The last century has witnessed a soaring usage of remotely piloted Unmanned Aerial Vehicles (UAVs), commonly known as drones. Because they are comparatively much cheaper to operate than manned aircraft, they have entered and revolutionised many fields in which they continue to be increasingly utilised. Some of their applications include: agricultural crops scanning [9], urban planning [10], city mapping (such as Google Maps) and photogrammetry based historical site 3D reconstruction [11], disaster management (wildfires and earthquakes) [12], search and rescue [13], parcel delivery [14, 15, 16], last mile delivery [17] and many other usages in many diverse fields.

The UAV market continues to expand and develop, with some reports [18] predicting it to triple by the end of this decade, with a growth of £7 billions per year. However, a number of challenges accompany the advent of UAVs. As the UAV market continually grows, UAV risks grow with it, and we are increasingly faced by growing concerns to address those risks (safety, security and privacy). One of the notable and most critical challenges we face today is safety, not only the safety of the drone itself but also the safety of its surroundings (any people or property possibly residing under the UAV's navigation space).

The critical issues, challenges and public risks of UAVs are well known, and they have already been identified for example by the Department for Transport in early 2017 [19]. Many goals had been set, such as, (a) maximising security and safety at all times and (b) real-time situational awareness. But they are yet to be fully addressed. This thesis is an attempt to address the safety critical issues of flying a fixed-wing UAV as opposed to a rotor-wing UAV (see Figure 1.1 for a graphical comparison), and to propose appropriate computer vision based solutions. It will focus on small fixed-wing UAVs, which can be also referred to as Small Unmanned Aircraft (SUA), under the weight of 20kg (as according to CAA classification [8]), which fly at a height which allows the surface of the ground to remain clearly seen.

Some professional grade fixed-wing UAVs, for example the ones used for photogrammetric scanning such as drones sold by Sensefly [20] and Wingtra [21], are guided by an advanced satellite direct georeferencing technology which is expensive for the average customer in its current state. These drones use state-of-the-art technologies in waypoint satellite navigation (eg. Real Time Kinematics (RTK) and Post Processing Kinematics (PTK) [22] technologies), and the whole system costs upwards of £17,000. This price tag

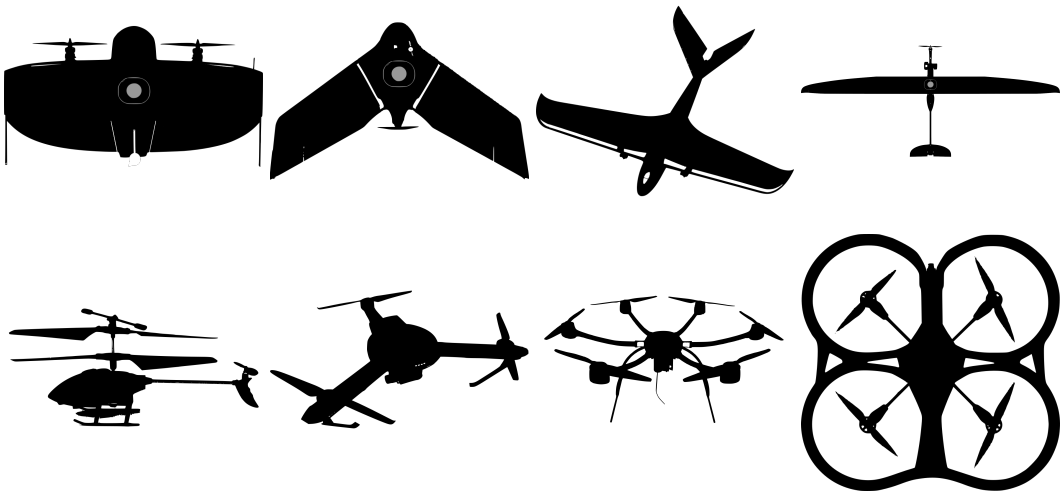

FIGURE 1.1: Top: fixed-wing UAVs. Bottom: rotor-wing UAVs.

is enough to push away customers not using these UAVs for professional purposes into finding cheaper solutions. In contrast, consumer grade low-end fixed-wing UAVs do not rely on battery intensive and high-cost external satellite navigation technologies, and they are commonly sold up to 100 times cheaper. Many of these UAVs are provided with a downward-facing camera, and they rely on being remotely controlled by a pilot for navigation.

Such fixed-wing UAVs are used to navigate much longer distances and at higher speeds as opposed to rotor-wing UAVs (see Table 1.1 for a comparison). It is argued in this thesis that such UAVs pose inherent safety and security risks for many reasons. These reasons are expoounded in the following section.

| | **Fixed-wing** | **Rotor-wing** |
|---|---|---|
| **Speed** | High | Low-Medium |
| **Altitude** | High (minimum of 100m, up to a few hundred meters*) | Low-Medium (normally operated up to a few tens of meters) |
| **Coverage/ Range** | Outdoor/Long (up to tens of kms) | Indoor & Outdoor/Short-medium (up to a few hundred meters) |
| **Agility/ Maneuverability** | Low | High. Has hovering capacity |

TABLE 1.1: Comparison between small fixed-wing and rotor-wing UAVs. * Flying at altitudes higher than 120m requires explicit permission from Civil Aviation Authority [8].

## 1.1 Motivation

Whether the UAV navigation system relied on advanced Global Navigation Satellite System (GNSS) technology, or on the skill of the remote pilot, both methods have many vulnerabilities and there are many factors that could contribute to a navigation failure. Such vulnerabilities include:

1. Human error: Some reports have outlined that humans are not good pilots, reporting high human error percentages in remotely navigating a UAV which range from 20% up to 60% [23]; some papers reporting human error percentages from 70% - 80% [24], these include *skill-based* errors (failure of the pilot to maneuver a certain situation), *perceptual* errors (failure of the pilot to notice an obstacle) and *decision* errors (failure of the pilot to make the right decision in a certain situation).

   Indeed, given the current UK regulations regarding UAVs, and according to the Civil Aviation Authority (CAA), unless permission is guaranteed, a UAV must remain within the Visual Line Of Sight (VLOS) of the pilot [8]. I.e. the UAV must never be flown out of eye sight range. But despite this regulation, it is not enough to guarantee safe flight. Given this reliance on the user and his level of experience, the UAV poses safety concerns.

2. GNSS failures: A GNSS signal transmitter power is equivalent to a weak light bulb, 27W in the case of Global Positioning System (GPS), which exists on a satellite more than $20,000km$ away. This means that the received GNSS signal on the ground is very weak which makes it succeptible to considerable malicious or unintentional interferences which can cause a navigation failure. Such interferences include:

   (a) Jamming: A jammer is a transmitter which transmits a signal that is slightly stronger than GNSS signal, which is enough to deflect it. Since the GNSS signal is already weak, a $0.1W$ transmitter for example can interfere with and block GPS signals within a radius of 10km [25]. Jamming poses considerable risk to UAVs as such jammers are available to purchase on the internet by civilians [26]. Military GPS jamming tests affect GPS signal reception, for example 173 cases were recorded during a 6 months period in 2017 [27]).

   (b) Spoofing: A spoofer on the other hand is a transmitter which replicates the GNSS signal. This causes receivers to ignore the real satellite signal and listen to the spoofer instead, which causes it to have false location and time measurements. In a maritime global journey numerous losses of signal were recorded sometimes lasting for hours [28].

   (c) Situational reliability: There are 24 operational satellites dispersed in space and used by GPS technology. To localise correctly the receiver needs to receive signal from at least 4 satellites. Therefore signal unreliability is introduced near hills, mountains or large

man made structures which cause either signal reflection or obstruction.

(d) Unintentional interferences: Includes natural interferences like ionospheric disruption or unintentional interferences from radio frequency (RF) waves.

3. Software errors: A famous example is the commercial airplane Boeing 737 Max. Boeing introduced a new software meant to run in the background to compensate for a change in engine positioning. Due to complications related to this software, two plane crashes occurred, one in October 2018, and another in March 2019, killing a total of 346 people [29]. The software errors in this particular case can be traced back to bad software engineering as the software relied on a single sensor [30], which highlights the importance of redundancy. Errors can also be traced to human self-seeking decisions [31], for example, compensating safety in the name of minimising production costs, matching delivery schedules and making more profits.

Even if the chances of such failures were rare, their impact remains massive as they have the potential to cause catastrophic damage. They pose a considerable safety concern by placing a significant physical danger on humans residing in UAV's flying space. Direct collisions with humans or man made structures can cause serious harm. These issues make it of vital importance to have a redundant and automated safety mechanism that is independent and does not rely on external aid such as GNSS. Indeed, this need has already been acknowledged by numerous technical reports [32]. This is why it is important to have separate localisation systems that backup each other and keep each other in constant check by providing feedback to each other, especially so on fixed-wing UAVs given their high speeds and high flying altitudes.

This research is proposing the usage of the integrated cheap cameras and onboard processing to aid in navigation and to improve the safety of the UAV. The usage of computer vision can aid in dealing with safety issues that can possibly arise due to human, software or GNSS errors. It is therefore important to assist the pilot using technology, and this is the reason why in this work, solutions are proposed which can be used either to improve UAV safety, or to influence legislation. These proposals will be described in the following section.

## 1.2 Aim and Objectives

Even though a UAV failure might be rare (for example due to GPS failure), or more common such as a human piloting error, but its impact is great, as it poses considerable risks. The aim of this research is to develop computer vision based tools to address the safety related concerns connected with piloting a UAV, and possibly be able to influence legislation. This is done by

using computer vision, utilising an onboard down facing camera and moderate onboard processing, to aid in navigation (see Figure 1.2). The following objectives are proposed to improve UAV safety:

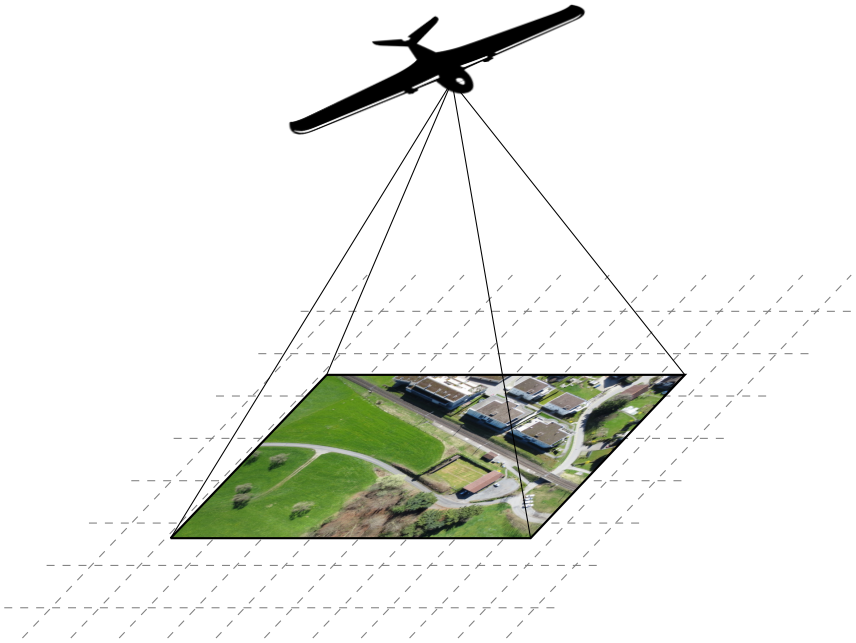

FIGURE 1.2: Fixed-wing UAV taking top-down snapshot.

1. Give UAVs an autonomous *homing* capacity. Homing is a safety measure which gives the fixed-wing UAV the ability to head back to its take off location (home) autonomously and in a straight line in case of an emergency. For example, when the UAV goes out of the sight of the pilot, when the pilot makes a navigation mistake or in case of GNSS failure.

2. Improve the safety of flying a fixed-wing UAV by adding more redundancy. Using advanced GNSS localisation technologies on some UAVs reduces the chances of failure on high-end UAVs. But this alone does not eliminate all possible risks, therefore adding redundancy to any navigation system improves its safety and reliability. Such localisation mechanism is even more valuable for low-end UAVs which do not rely on state-of-the-art GNSS localisation.

3. Contribute towards satisfying the requirements of government safety regulations such as flying beyond visual line of sight (BVLOS) [8]. If it was possible to localise a small UAV, and make it head back home autonomously, it can be shown that a direct visual line of sight is not required for a safe return. Therefore it can allow more autonomy to be safely implemented.

4. The UAV should be able to navigate safely in new environments without being previously trained or adapted to fly over these environments.

## 1.3 Contributions:

The objectives described in Section 1.2 have led to the following thesis *contributions*:

1. The proposal of a deep learning segmentation based UAV localisation system given an aerial map (Chapter 4). The proposed system enables localisation in new cities which the UAV has not seen before. This is done through the segmentation of both the aerial map (e.g. Google Earth) and the captured snapshots using a Convolutional Neural Network. The CNN is not trained on any data derived either from the navigated environment, nor the aerial map. It is shown that using these segmentations it is possible to localise the UAV in an urban environment which enables homing, achieving accuracy comparable to state-of-the-art systems but with the added advantage of global localisation.

2. The proposal of a deep learning based rigid aerial image registration system (Chapter 5). The problem of registering segmented aerial images with large rotation and translation is studied, and an investigation is done into the possibility of transforming the traditional visual odometry system component into a learning based localisation system.

3. Understanding the efficacy and the shortcomings of traditional localisation methods on fixed wing UAVs (Chapter 3), and the proposal of appropriate traditional algorithms for fixed-wing UAV visual odometry (VO).

## 1.4 Thesis structure

The remainder of this thesis is divided into the following Chapters:

***Chapter 2*** contains a comprehensive literature review. The review covers state-of-the-art algorithms in UAV monocular navigation, visual odometry, deep learning techniques such as segmentation and registration, in addition to some inspiring biomimetic navigation techniques.

***Chapter 3*** provides an understanding of the application of traditional motion estimation techniques on fixed-wing UAVs. It proposes suitable visual odometry (VO) localisation techniques, addressing the first objective.

In ***Chapter 4***, deep learning is investigated in order to build a novel segmentation based UAV localisation system given an aerial map. A Convolutional Neural Network (CNN) is proposed which is trained on a separate dataset but can segment any aerial map image. This chapter addressed the second contribution of this thesis.

In ***Chapter 5***, a systematic study of the possibility of extending deep learning to other parts of the UAV localisation system is addressed, particularly to motion estimation. A suitable rigid based deep registration network which aimed to replace traditional image matching algorithms was introduced. This chapter addressed the third contribution.

And finally, ***Chapter 6*** contains a summary of the conclusions of this study. Possible applications are proposed and possible future research which can be made in this field are outlined.

# Chapter 2

# Background

This chapter is going to be a literature review of state-of-the-art methods which are related to Unmanned Aerial Vehicle (UAV) navigation. The review covers numerous fields: A short introduction will be provided first in Section 2.1. Monocular navigation will be explored in Section 2.2, a deep learning review will be provided in Section 2.3, a review of insect navigation methods in Section 2.4, other notable navigation techniques will be covered in Section 2.5 and finally a summary is provided in Section 2.6.

## 2.1 Introduction

In recent years, the surge of demand on unmanned aerial vehicles (UAVs) or drones, in the commercial industry, has been driving more investments in UAVs, and eliciting more UAV-related research in many universities. In 2016 global UAV sales registered $13 billion, and UAV related solutions across industries reached $127 billion [33], with a projected spending of up to $11 billion per year [34]. These numbers prove that UAVs accommodate for a big market that continues to grow.

Drones are being used today for many purposes, such as: long-range photogrammetric environment scanning (for extraction of measurements like distances and volumes) [21], cartography including orthographic environment scanning (ortho-mosaicing), damage estimation after natural disasters like floods, fires and earthquakes, and agricultural crop scanning for yield estimations [20], inspection in difficult to reach areas [35], investigation of contaminated areas [36], delivery and parcel transport [37], surveillance, wildlife surveys, historical site archiving (as a successor for the use of photography for archiving [38]), transportation [39], using swarms as an expression of art [40], exploration, filming and journalism, humanitarian aid, environmental protection and search and rescue emergency services [41], weather forecasting, data collection... etc.

The reason for the big interest in UAVs is their advantages over traditional manned aircrafts (both fixed and rotary wing systems)[42]. Such advantages include:

1. A UAV does not require a pilot to reside in the aircraft, which prompts the ability to fly it in more dangerous areas and take risks that otherwise would be life endangering for pilots, for example flying in toxic environments, bad weather conditions, fires.. etc.

2. A UAV costs cheaper to operate than flying a passenger size aeroplane or a helicopter, with quicker, more flexible and reliable deployment times.

But despite the agility and flexibility of drones, they have numerous limitations in various areas when compared to ground vehicles. These limitations include:

1. Stability: It is difficult for a hovering drone to maintain sufficient stability which is necessary to operate an attached moving device like an arm.

2. Weather conditions: A UAV is more susceptible to be affected by weather conditions especially wind, which can have a destabilising effect on the drone while flying.

3. Navigation complexity: Even though that obstacles laying on the ground might not be of concern to the UAV, but it still has to manage obstacles that might be more complex to navigate through, for example, a gap in a wall, or avoiding a descending column from a ceiling.

4. Recoverability: A slight crash or accident against an object by a UAV is likely to introduce catastrophic damage which renders the drone inoperable unless being salvaged, and it can even be completely irretrievable.

In general, each robotic platform (ground or aerial) has its own limitations which makes each platform able to address a specific problem more efficiently than the rest. This is why each platform has its own applications. An optimum solution to complex problems is very likely to require a combination of both aerial and ground vehicles.

## 2.2 Monocular Navigation Overview

In this section, current monocular navigation techniques will be described: Visual Odometry (VO) in Section 2.2.1, Visual SLAM in Section 2.2.2 and photogrammetry in Section 2.2.3. Photogrammetry means monocular 3D scene reconstruction, and it is also referred to as Structure from Motion (SfM). Finally some insect navigation techniques are discussed in Section 2.4.

It might be important to link the 3 previous terms ahead of delving deeper into them, as VO, Visual SLAM and SfM are related to each other. VO is a special case of SfM; and SLAM is VO with the added ability to recognise previous locations (loop closing).

### 2.2.1 Visual Odometry

The term VO was first coined in [43]. Visual Odometry (VO) is a dead reckoning technique used to estimate the distance travelled by a robotic vehicle/UAV using solely visual input, as described in the two comprehensive

introductions [44, 45]. Dead reckoning is the process of determining current position using a previously memorised locations given speed and direction of movement; subject to drift due to inaccuracies in speed/direction measurements.

VO is like a spider hanging from a thread in one of the corners of a room. The spider has to maintain at least a single thread attached from itself to one of the walls so that it can have at least some control. A single line of thread would allow it to determine its position on one axis. And the more threads it can link itself with its environment (walls around it), the more accurately it can determine its location in the environment. This is called localisation. And in VO, a robot/drone tries to do exactly the same thing, but instead of using cobweb, the links to the environment are established by tracking features in visual input over time. For example, a robot establishes a connection to a corner, by observing this corner over time, in other words, by tracking it across frames using the visual sensor. This is called Visual Odometry.

It is possible to combine the visual data with information coming from other internal sensors such as an inertial measurement unit (IMU) which increases localisation accuracy. Or it could be combined with an external source of information such as a global navigation satellite system like GPS or GLONASS.

VO does not offer the ability to recognise places visited in the past. So we can think that VO is like building a linked list of consecutive visited places, which entries are the main features that are tracked over time. It is possible to plot this list on a map, but the result wont be coherent and the resulting map wont help in navigation due to the lack of loop closure, and due to drift.

#### 2.2.1.1 Global & Relative VO:

VO can be classified into two categories [46]: *Global VO*, i.e. the ability to localise the UAV along the way from take off until reaching the destination. And *relative VO*, which is the ability to localise the UAV in relation to observable objects in the environment, for example to help in landing the UAV. The first is long term, the second is short term.

One of the early works implementing relative VO is the visual odometer system implemented by Amidi [46]. The system estimates a helicopter's location by processing a video input from vertically positioned stereo cameras, and calculating frame to frame difference using sum of squared differences (SSD). This information is combined with data coming from a gyroscope to estimate motion. Amidi's thesis [47] displays good relative (short term) odometry results.

Other navigation methods employ computer vision techniques to do the required navigation. For example in [48] the navigation is done by extracting roads from aerial photographs and matching them to each other to help localise the drone. In [49], [50] global navigation is done using image registration, which is used to match captured frames after extracting edges, against a database of aerial/satellite photographs.

### 2.2.2 Visual Simultaneous Localisation and Mapping (SLAM)

Simultaneous Localisation And Mapping (SLAM), also called Concurrent Mapping and Localisation (CML); is a technique that has received an intensive amount of research over the past two decades. There are many available introductions to SLAM available such as the comprehensive tutorials by Whyte and Bailey (University of Sydney) [51, 52], and reviews [53].

The main difference between Visual SLAM and VO, is loop closure (see Figure 2.2). We can say that SLAM is doing VO with the added ability to recognise places visited in the past, in other words, Visual SLAM is VO with the added capability to close loops or perform loop closure. So any Visual SLAM system can be reduced to a VO system if we take away its loop closure capability [53].

Authors in [53] propose a valid classification describing SLAM as composing two systems: Front-end and Back-end. The Front-end includes localisation (feature detection, feature matching, triangulation) and mapping (including loop closure). The Back-end includes more sophisticated and intensive operations like line-fitting and other optimisation (Newton Gradient Descent) and estimation techniques (MAP, MLE) that operate on data coming from the Front-end.

If VO was about building a sequential list of visited landmarks, then SLAM would be about producing a coherent map from this list that the robot can use to navigate the environment and localise itself within it. Mathematically, it is possible to categorise SLAM into two classes [54]:

1. *Online SLAM*: which is the estimation of the current robot/UAV location depending on current sensor measurements and landmark observations (an example is the EKF-SLAM, described later). The second part is Full SLAM or *Offline SLAM*: which is the estimation of the map and the complete robot path depending on all previous measurements and landmark observations (an example is the Graph based SLAM algorithms Graph-SLAM, which is a linked-nodes representation of robot/UAV poses and observed landmarks).

2. Active-SLAM is a group of SLAM algorithms that consists of 2 stages: first identify possible regions to explore; then explore those areas then decide whether to continue or withdraw.

Visual-SLAM, is a kind of SLAM that is dependent on visual sensors, as opposed to 2D/3D LIDAR SLAM. It is a particular case of Structure from Motion (SfM), (see Table 2.1 for a comparison). The difference between VO and SfM is in some way identical to the difference between online and offline SLAM. VO can be classified into three different categories (see Figure 2.1):

**Localisation:** In Visual-SLAM, the standard procedure is: feature detection, feature matching, then triangulation. These steps provide the ability to determine the location of a robot. LIDAR SLAM follows the same procedure, but it has different sensors: so it starts with capturing lidar frames (point clouds), then point cloud matching. GNSS and IMU data can be combined

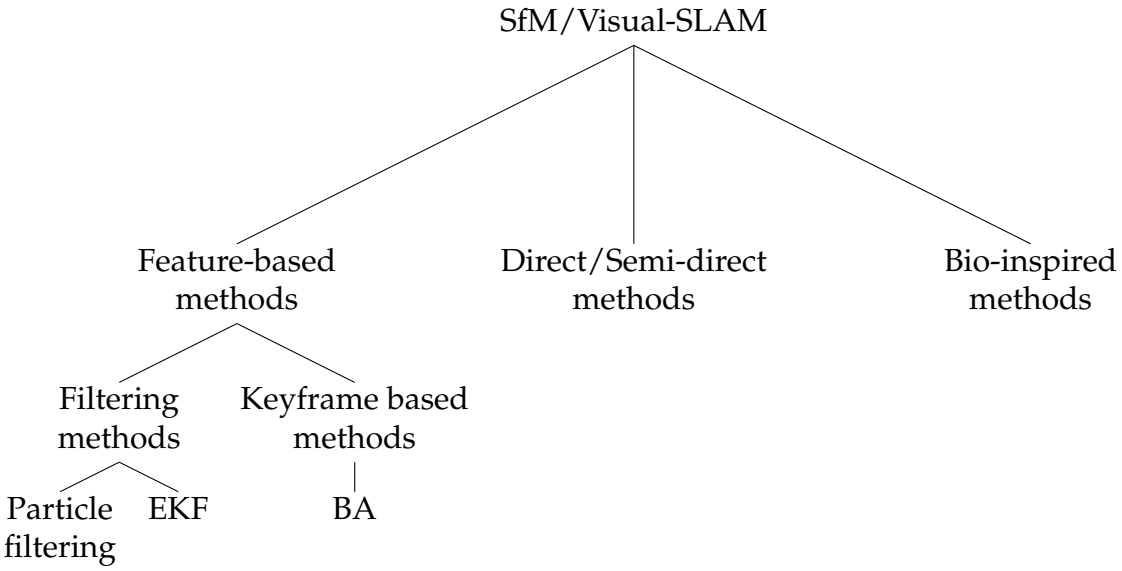


FIGURE 2.1: Classification of Visual-SLAM methods

in either systems for more accuracy. And BA can be used for more accurate results.

**Mapping:** Mapping is the process of memorising observed landmarks with locations relative to each other. It can be classified into four types [53]:

1. Landmark based maps: or topological mapping, models the environment as a collection of landmarks or features, the geometry between them is preserved. It has a sparse representation.

2. Occupancy grid mapping: segments the environment into equal segments (cubes) and assigns an occupancy probability to each segment.

3. Raw dense 3D representations: are raw representations, such as point clouds. Point clouds can be produced using depth cameras (RGBD), 2D & 3D LIDAR scanners [55] and Direct methods (LSD-SLAM TUM university [56]). It is also possible to produce these point clouds using photogrammetry and SfM. Their disadvantage is the high volume of raw data they contain. This kind of representation requires high processing power, but provides high accuracy.

4. High level 3D representations: provide high-level understanding of the environment (shapes & structures). These methods either segment the environment into separate recognisable 3D objects with known geometrical relations; or they use semantic labeling (just like the BoW approach), but on the 3D object level.

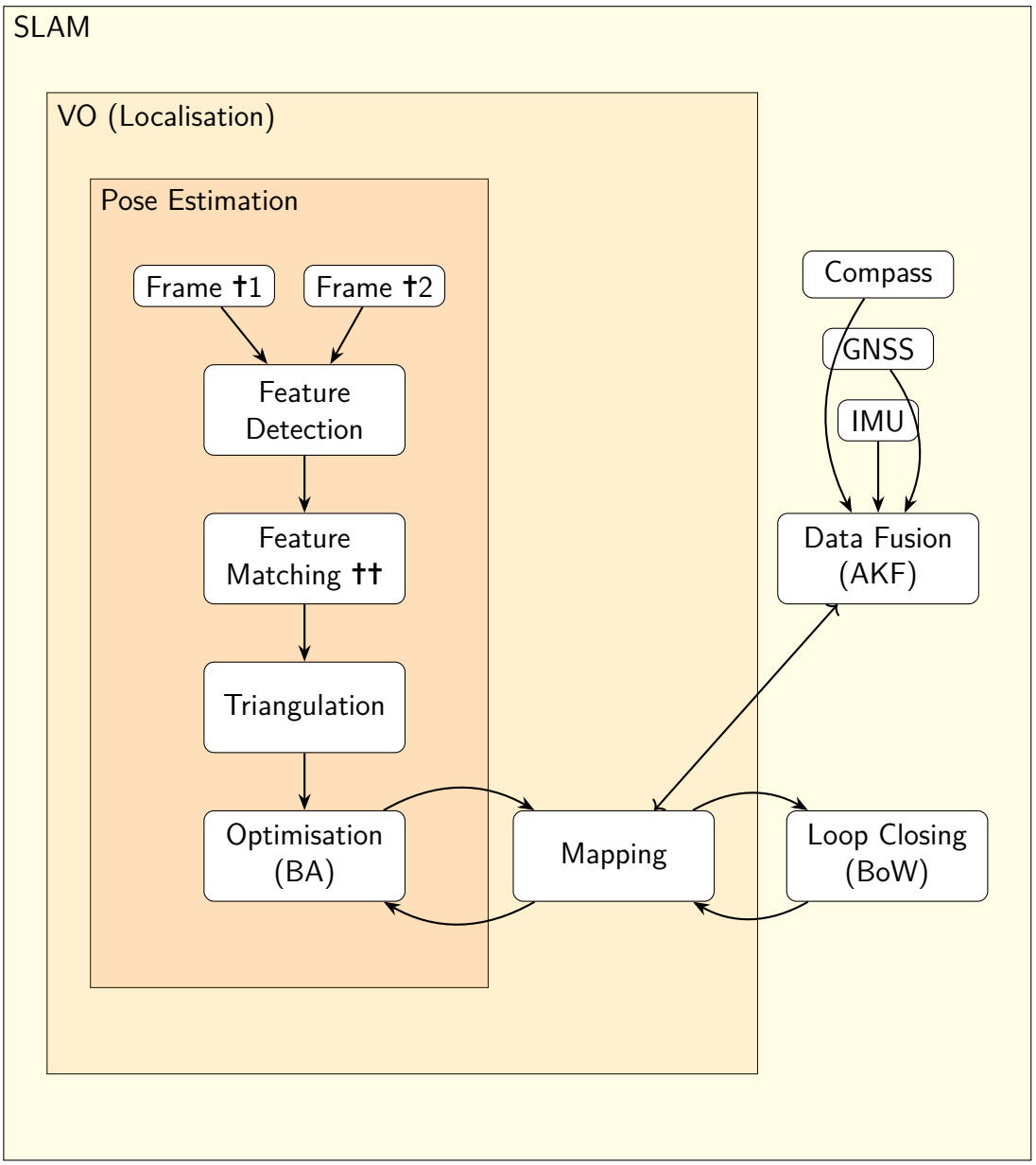


FIGURE 2.2: Dissection of a SLAM system. (†) Frames can be produced using Monocular camera (as in our situation), Binocolour or RGBD cameras, or 2D & 3D LIDAR sensors. (††) RANSAC can be applied to remove outliers after the feature matching step.

**Loop closure:** Loop closure is a method of telling that a certain landmark (or a group of landmarks) has been visited in the past, therefore identifying a place. By visiting this place again, a loop is formed, cancelling all drift resulting from navigating this loop. Without loop closure, SLAM is reduced to mere odometry. This was one of the main reasons that SLAM was developed, when it was realised that the observation of landmarks can help reduce the odometry positioning errors.

This means that the only difference between Visual-SLAM and Visual-Odometry is loop closure (see Figure 2.2). This makes VO concerned with

the accuracy of the local trajectory, whereas in SLAM, it is important to have a consistent global trajectory in a map.

There are different techniques used to recognise visited places, one of them is the Bag of Words (BoW) algorithm [57]. In BoW, each image is tagged by a number of words forming a vector, which are in turn stored in a tree called a vocabulary tree. Every time a new snapshot is captured, the tree is queried for a matching vector. If one is found, a loop is identified. BoW is used in many algorithms such as the monocular ORB-SLAM [58].

#### 2.2.2.1 Feature based methods:

SLAM algorithms that depend on feature detection to estimate the camera motion can be called Feature-Based methods [44]. The workflow of feature based methods is split into two stages: first the detection of features in the images (hence the name), and then trying to estimate the camera perspective and the geometry of the scene. The first stage is done using well known feature detectors such as SIFT or ORB.

**Filtering methods:** There are many filtering algorithms, of those we mention the Extended Kalman Filter (EKF) based, and the Particle Filter (PF). Filtering methods are probabilistic filters [59], and they attempt to estimate the current robot location using a probability distribution by combining input information gathered using robots sensors. For example the robots location can be estimated using either information coming from monocular sensors alone, or after combining them with information coming from another sensor like an IMU or a GNSS signal.

The main difference between the two is the number of Gaussians they use. Both filters use Gaussian distributions to represent the robot state, an EKF can have a single Gaussian, whereas a PF can have as many as the particles it is using.

EKF-SLAM is described in detail in [54]. It uses a Maximum Likelihood Estimator (MLE), which is a statistical method that estimates current (the posterior) state using previous states/observations. MLE is related to MAP.

In one sense, we can say that an EKF is used to combine information coming from different sensors. For example in [60] it is used to fuse camera poses with IMU data to provide more robust camera location estimates. In [61] it was used to determine image scale by fusing visual odometry data with IMU and PID controller data. Early EKF Visual SLAM systems include Davison's work (Imperial College London) [62, 63] (Figure 2.3)

The second kind of filtering is the Particle Filter (PF), which is mostly used with 2D & 3D LIDAR sensors. PFs are Monte Carlo sampling based algorithms. One of the early algorithms to employ it was FAST-SLAM [64, 54], using a filter called Rao-Blackwellized PF. A PF application on UAVs include its usage in [65] to localise emergency sound signals coming from lost victims.

**Smoothing methods:** Also called Keyframe based methods [66]. Keyframe based methods are selective methods, i.e. they do not process all input information, but a selected subset of this information (for example by selecting

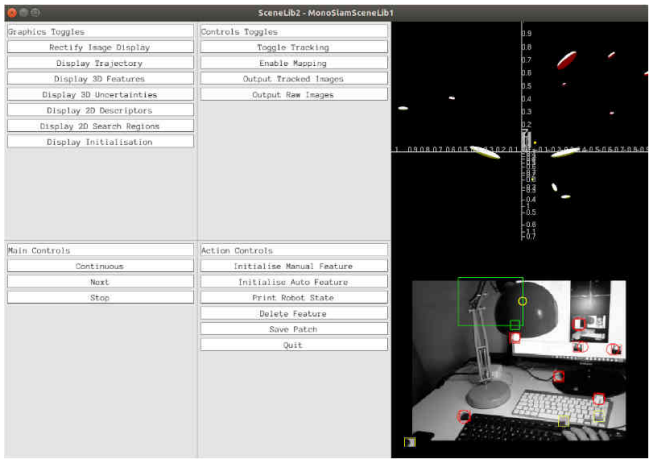


FIGURE 2.3: Mono-SLAM (SceneLib2) running on our machine

one frame every n number of frames). Bundle Adjustment (BA) is an example of an offline smoothing based technique (actually, it is called offline because it requires a minimum recent number of input camera poses, this is why it is also called Windowed Bundle Adjustment). BA is refining the calculated camera perspectives from frames so as to get a consistent 3D camera movement (camera path or trajectory) which would enable us to deduce odometric information.

Since BA is selective, it operates on a fewer number of frames, and this allows it to detect more features in a single frame. This is the main reason why Strasdat & Montiel et al. [66] determined that BA based methods are more accurate and robust than filtering methods. One of the most successful papers in this category is PTAM by Klein and Murray [67], and it is the basis of many other algorithms like [60, 68, 58, 69].

When it comes to navigation, one of the best performing state of the art SLAM algorithms in this category is the recent algorithm by Raul Mur-Artal from Universidad de Zaragoza, ORB-SLAM2 [70] (Figure 2.4), which is based on the earlier ORB-SLAM [58].

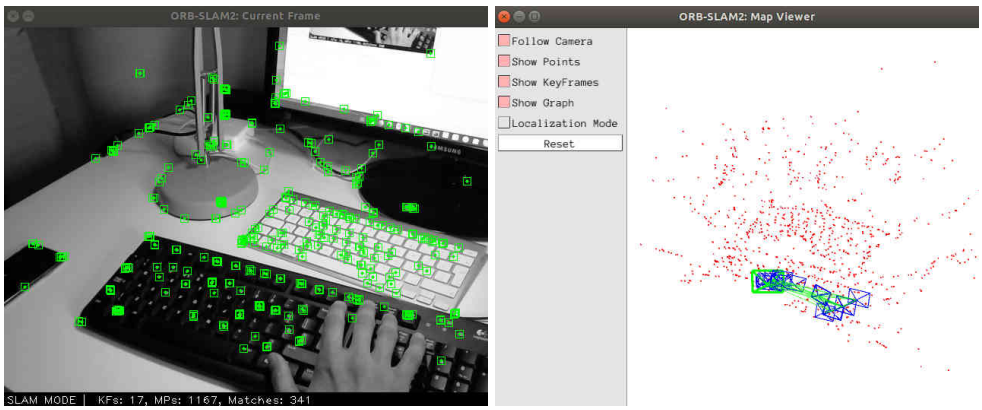


FIGURE 2.4: ORB-SLAM2 running on our ubuntu machine. Left: Current frame with detected features. Right: Mapped features in 3D.

Researchers at the highly regarded Robotics and Perception group at the University of Zurich have extended ORB-SLAM2 to make it much more robust in varying lighting conditions using active exposure control [71].

One of the earliest projects to use monocular navigation is the SFLY project [69, 60, 72] (2009-2011), which was led by Davide Scaramuzza from ETH Zurich. Their algorithm processed monocular images in a GNSS-denied environment; and it was based on the popular algorithm by Klein and Murray [67].

Most recent advances in this category, and one of the best performing is a system that is still being developed and tested by Davide Scaramuzza at ETH Zurich. The SLAM system is dependent on a new type of neuromorphic visual system, called an event camera. The system is developed and sold exclusively by a Swiss company called iniVation. These cameras provide a low response latency, and high dynamic range which allows them to work much better in the dark, but they can capture only motion. Because of the super fast sensitivity of the camera, it allows very robust feature tracking. A SLAM system of this kind has been produced which combines a photographic camera, an event camera and an IMU [73], compared with the traditional SLAM system which only consists of a photographic camera and an IMU. This system and thanks to the event camera, easily beats the ORB-SLAM2 algorithm mentioned above.

#### 2.2.2.2 Direct and semi-direct based methods:

Direct & Semi-Direct Pose Estimation Methods, also called appearance or global based methods [44]. As opposed to Feature-Based methods that depend on feature detection, direct methods estimate the camera perspective directly from differences in pixel contrast values of the image. J. Engels (Technical University Munich) algorithm LSD-SLAM [56] is an example.

The disadvantage of Direct based methods when compared with feature based methods, is that they require higher processing power [44], as feature based methods are well mature and they are optimised for speed as they only use a fraction of the available information in the images. However global based methods perform better in low textured environments [53]. An example is LSD-SLAM developed at Technical University of Munich by Jakob Engel and others [56]. Semi-Direct methods such as SVO [74] is a combination of feature based and global based methods, and it is proved to produce the most efficient performance.

### 2.2.3 Photogrammetry/SfM

Photogrammetry is performing 3D reconstruction and producing a 3D model of a scanned object using a given set of photographs. This operation can be done in real time using a continuous flow of high frame rate images (video), or post capture using multiple photographs of the object taken from different perspectives.

Photogrammetry results in a 3D model of the environment, or a depth map, which can be used for measurement purposes, like distances and volumes. The depth map can be also used to produce an orthographic projection of the environment, an orthomosaic which serves as a map.

A special case of photogrammetry is structure from motion (SfM). Both SLAM and VO, can be considered as a real-time structure from motion technique. SfM is determining the cameras perspective by comparing two overlapping photographs. By determining the corresponding features in each photograph, and by doing triangulation, both the perspective of the camera and the camera trajectory can be calculated, and by doing this, a reconstructed model of the environment can be produced. SfM is concerned with calculating the camera perspective, and VO is concerned with calculating the trajectory (See Table 2.1 for comparison).

#### 2.2.3.1 Real-time reconstruction

This kind of reconstruction is done in real-time using a high frame rate input, and it requires a SLAM system to be running side-by-side. One of the recent and best performing algorithms for real-time reconstruction is the work by Scaramuzza Regularised Monocular Depth Estimation (REMODE) [75]. The same algorithm is used in [76] to determine safe landing locations for UAVs.

#### 2.2.3.2 Offline reconstruction

Performed offline after the photographs have been captured. No SLAM is required, and the minimum amount of overlap between each frame and the next is a minimum of 30%. But the exact location and orientation of each photograph have to be known; hence it requires the use of a GNSS system [22]. The images captured are processed post-flight to produce a 3D scan of the environment, this scan is a point cloud which can be used to obtain numerous measurements like distances and volumes. This is done by performing photogrammetry on the captured images, which are also used to build orthomosaic maps.

This photogrammetry reconstruction stage is highly complex, and it is done using professional software such as Pix4D [77].

| | Process | Goal | Operation | Data input | Optimisation |
|---|---|---|---|---|---|
| **VO** | Same as in Figure 2.2 | 3D camera motion calculation (i.e. path or trajectory) | Online (real-time) | Consecutive/ incremental | Not required |
| **SfM** | Same as VO | 3D reconstruction of scene structure | Offline | Unordered | Required (BA) |

TABLE 2.1: Comparison between VO & SfM

### 2.2.4 State Of The Art

Numerous tutorials [44, 45] offer lengthy and detailed information of how to implement a VO system, however, VO is rarely used alone due to inaccuracies resulting from drift. Perhaps it is only used up to a few hundred meters [78]. This is especially true for safety critical operations such as flying a UAV. For this task, a loop closing scheme is needed, and this is achieved through SLAM [60, 79, 70, 73]. SLAM is used in different kinds of robots and environments today, for example, rotor-wing UAVs which fly in closed or short range environments [60, 73], aerial robots within limited outdoor environments [60, 69, 80, 81] and wheeled robots within cities [82, 83].

## 2.3 Deep Learning

### 2.3.1 Introduction

Deep learning is used to refer to layered machine learning networks which aim to automatically learn a representation of some specific data, avoiding the need for traditional hand engineered machine learning algorithms which are meticulously written to find such patterns of information. Its development was due to the failure of traditional machine learning algorithms to generalise on large tasks. Since their introduction, deep learning networks have vastly surpassed state-of-the-art conventional algorithms in many fields, for example, image classification [84], semantic segmentation [85], speech synthesis [86], speech recognition [87], face recognition [88], object detection [89] and many others.

A deep learning network's goal is to find a specific relationship between input data units, which is done sequentially through multiple layers, each representing the inputs using fewer but more representative features. Each network is composed of an input layer which receives input training data units in batches, one or more hidden layers (hence the name 'deep'), and an output layer which produces the representative information predicted by the network (see Figure 2.5).

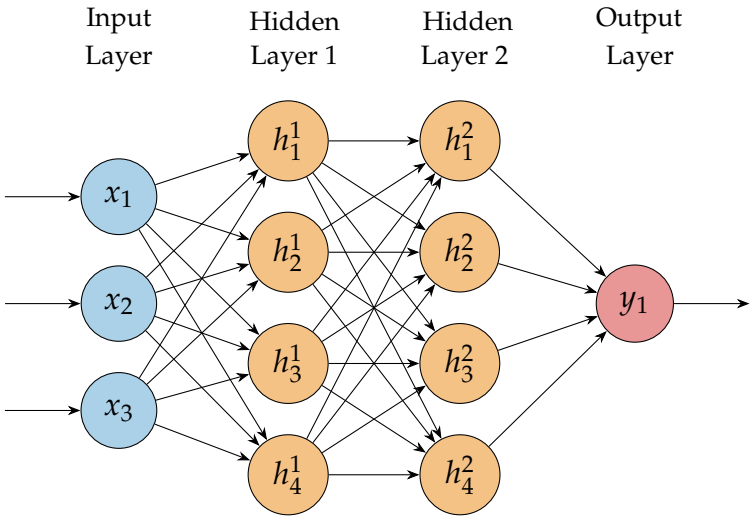


FIGURE 2.5: A single output neural network with 2 hidden layers and 3 inputs. The intermediate transformed information which is the output of each layer is called a *feature map*.

Each layer in a deep learning network is composed of many nodes. Each node resembles a simple non-linear function which maps the inputs to the outputs (called the *activation function*), in addition, each node has a weight and a bias. The numerical value of each node is found by multiplying each one of its input by the weight of this input, and adding the bias value. For example, the value of the node $h_1^2$ in Figure 2.5 can be found like so:

$$h_1^2 = f(w_1^1 h_1^1 + w_2^1 h_2^1 + w_3^1 h_3^1 + w_4^1 h_4^1 + b_1^2), \tag{2.1}$$

expressed in matrix form:

$$h_1^2 = f(\begin{bmatrix} w_1^1 & w_2^1 & w_3^1 & w_4^1 \end{bmatrix} \begin{bmatrix} h_1^1 \\ h_2^1 \\ h_3^1 \\ h_4^1 \end{bmatrix} + b_1^2), \tag{2.2}$$

which can be generalised to resemble an entire layer like so:

$$h^2 = f(W^1 h^1 + b^2). \tag{2.3}$$

Each one of these layers can be represented by the following graph (Figure 2.6). To some degree, the graph resembles a neuron; and its development origins might be traced back to neuroscience through the field of computational neuroscience. This is the reason each deep learning network is also called a *neural network* (NN). The combination of these simple units in layered structures gives the NN its powerful performance, as layers learn increasingly more accurate representative features of the training data.

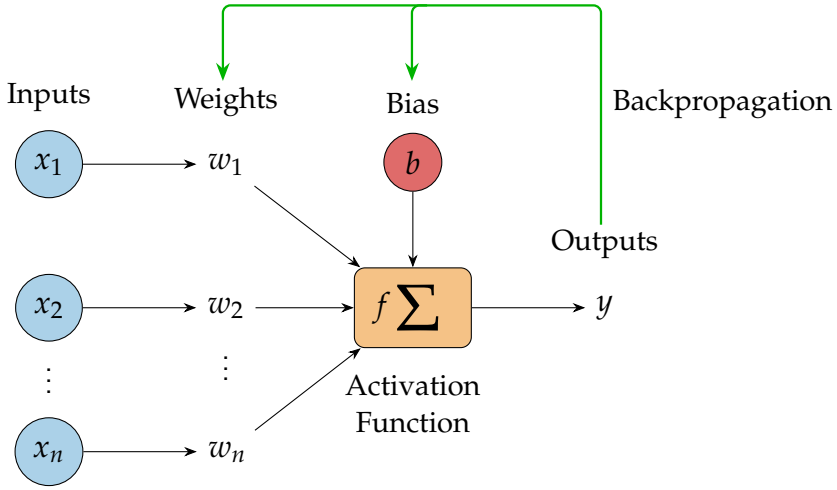


FIGURE 2.6: A representation of the operations done within each node (also applicable to a whole layer). The backward green arrow represents backpropagation.

#### 2.3.1.1 Activation function

There are many activation functions which can be adopted in Figure 2.6, one of them is the sigmoid function which is commonly used in regression problems. The output of the function is a probability distribution score between 0 and 1:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.4}$$

The Rectified Linear Unit (ReLU) is another function which simply suppresses any values smaller than 0. It has a simpler mathematical function therefore it has better performance than the sigmoid function, hence it is used more widely:

$$f(x) = max(0, x). \tag{2.5}$$

A generalisation of the Sigmoid function is the Softmax function, which is used for classification problems to distinguish between different classes:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}, \qquad \text{for } i = 1, \cdots, k. \tag{2.6}$$

Where $x_i$ is the input vector to the function and $k$ is the number of classes. $e^{x_i}$ is the exponential function applied to the input values; and maps negative values to small positive ones. The division term $\sum_{j=1}^{k} e^{x_j}$ ensures that all function outputs are in the range 0 to 1 which also sum to 1.

#### 2.3.1.2 Objective function

Each neural network has an *objective function* (also referred to as cost function, loss function, energy function or error function). In supervised learning this function measures the amount of error between the predicted and the

ground truth values. The aim of the neural network is to optimise this function using a minimisation algorithm, depending on the amount of error between the ground truth and the predicted values. This algorithm is called an *optimiser*, and the optimisation is done by modifying the weight and bias values of the neurons of the network by finding the derivative of the function $f(x)$ such that its value is minimised:

$$f'(x) = \frac{\delta y}{\delta x}. \tag{2.7}$$

Since the derivative is the slope, it provides a guide of how to minimise $f(x)$. This operation is done iteratively and gradually using an algorithm called *gradient descent*. In neural networks, a modified randomised version called Stochastic Gradient Descent (SGD). Unlike gradient descent which estimates the gradient on the whole dataset at once, SGD estimates the gradient on a small subset of samples chosen randomly from the training dataset. This process is repeated iteratively until the whole dataset is covered. SGD remains the most efficient training algorithm for neural networks until this date [90]. SGD is done recursively starting from the output layers and going in reverse layer by layer until the input layer is reached. In a NN which has thousands of parameters, derivation of the objective function can be done using the *chain rule* (i.e. deriving more than a function together) giving rise to a strategy referred to as *backpropagation* [91].

#### 2.3.1.3 Datasets

Dataset size and its quality (the existence of outliers) play a vital role in the development of a NN. Dataset size is indispensable to solve complex problems, the bigger the dataset, the more accuracy that can be achieved [92]. A dataset is splitted into 3 separate sets composed from a single large shuffled dataset. The data units belonging to each set are organised in batches. It is said that an iteration (or an epoch) has passed when all of the batches have passed through the network once:

1. Training set: Contains 70-80% of the overall data. Used to train the network, that is, the weights and biases of the network are updated so as to achieve a high loss function score on this set.

2. Validation set: Contains 15-25% of the overall data. This set is used to estimate the generalisation capacity of the network after each training iteration. Its main advantage is that it is a good indication of overfitting or underfitting (explained in the following sub-section).

3. Testing set: Contains 10-15% of the dataset units. Used to test the final generalisation ability of a NN after it has finished training.

#### 2.3.1.4 Neural Network Types

Neural networks can be classified into different categories based on the type and amount of supervision they receive during training. The datasets

used must have common features so that a relationship between these features is discoverable by the network:

1. Supervised learning: Ground truth labels (solutions) $l$ are available for dataset units $x$, these labels are used to guide the network how to learn relevant representations from $x$, i.e. the goal of the network is to find the probability $p(l|x)$. Used typically to solve problems such as classification, regression, object detection and image segmentation.

2. Unsupervised learning: Dataset units $x$ are unlabelled. The network has to discover a valid relationship between data units and learn on its own a generalisable representation probability $p(x)$. Used typically to solve problems such as clustering, de-noising and visualisation.

3. Semi-supervised learning: The dataset is partially labelled. Both labelled and unlabelled data units are used to find a generalisable representation $p(l|x)$.

4. Reinforcement learning: A more recent kind of learning where training data is generated in real-time. An agent (the NN) learns by interacting with an environment (sensing and acting) and it is rewarded according to some policy (learning strategy). Applied mainly in robotics like self driving cars. A successful and famous example of reinforcement learning is AlphaGo [93] which learned by playing against itself, and won against the world champion at the game of Go.

## 2.3.2 Common NN Concepts

### 2.3.2.1 Overfitting

Overfitting is a main challenge in deep learning. The term overfitting is used to describe the network when its generalisation capacity starts to degrade during training. If the network performance on the validation set stops improving during training, but it continues to improve on the training set, the network is said to be overfitting. At this stage, the network starts to memorise the training set, without being able to find a good representation which is applicable to the validation set. An over-fitted model is a model that performs well on the training set, but fails to generalise its knowledge to the validation or test sets (see Figure 2.7). The opposite of overfitting is underfitting, which is when the network's performance can't improve on the training set.

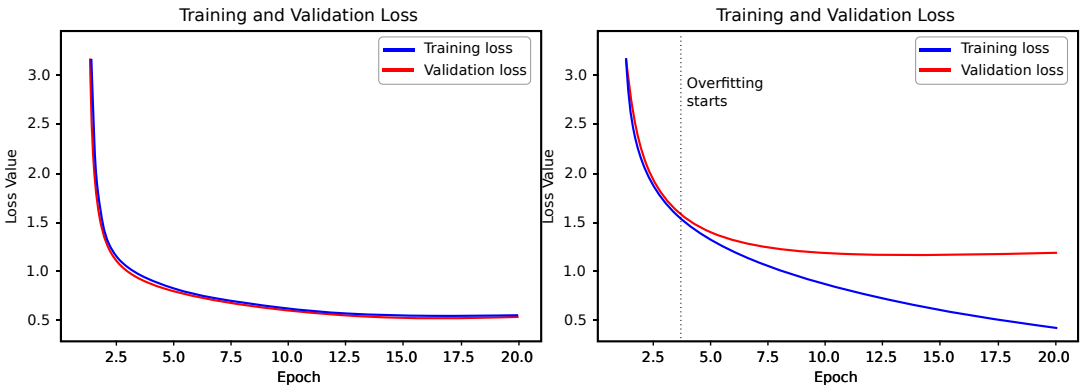


FIGURE 2.7: Left: A typical good learning curve. Right: Validation curve stops improving starting from the 3rd epoch, unlike the training curve. This is a major sign of overfitting.

Overfitting is solved using different techniques. For example, by removing outliers (noise) as much as possible from the dataset. Or by increasing the dataset size, as it is more difficult for a network to memorise a large dataset. This can be done through data augmentation, by translating, rotating, scaling and deforming existing photos from the dataset so as to generate more data. Or by inversely reducing network architecture complexity, as small networks have lower capacity to memorise the training set.

A different group of solutions is called *regularisation*. Popular regularisation techniques include weight regularisation which penalises the loss score when the weight coefficients are large, such as L1 (absolute) and L2 (square) regularisation (also called weight decay). L1 regularisation adds a sum of the absolute values of the weight coefficients to the loss function. It is commonly used when doing feature selection, as small weights get shrinked to zero, which keeps the important features. L2 regularisation adds a sum of the square weight coefficients to the loss, which unlike L1, does not nullify less important features. Another way is applying dropout which is an intermediate function between the layers of a NN, which randomly turns off a percentage of input feature maps [94]. This operation helps in reducing the reliance of neurons on each other.

#### 2.3.2.2 Transfer Learning

It is well known that building and training a neural network from scratch takes a considerable amount of engineering effort and time to design the network. This is not to mention the massive amounts of training data required. This is why transfer learning techniques were developed to adapt an existing and pre-trained neural network to learn a new task using a limited amount of training data (up to hundreds of images in the case of ConvNets).

For example, transfer learning can be done on a pre-trained image classification network trained on ImageNet [95] to classify images of sheep. The classification layers, usually the fully connected layers are replaced with untrained ones. The whole network is frozen except the newly added layers, then it is trained on the sheep dataset.

#### 2.3.2.3 Fine-Tuning

After a network has been trained, the whole network is frozen except for the first or last few layers. The learning rate is set to a much smaller value than before (eg. $10^{-6}$), and training is restarted on the current task at hand, where the same dataset is used. This process makes small adjustments to the weights and biases of the defrozen layers (hence the name fine tuning) so that they are more adapted to extract more relevant and accurate feature representations either on a high level (top defrozen layers), or on a low level (last defrozen layers). The network must be monitored carefully while training because the risks of overfitting when fine tuning are very high.

#### 2.3.2.4 Hyperparameter Optimisation

An automated randomised searching mechanism which allows the network find the optimum parameters which produce the highest training loss accuracy. It iteratively trains the network with randomly selected sets of settings from ranges of values specified by the user, such as dropout percentage, network depth, neuron counts in each layer, learning rate, activation functions.. etc.

### 2.3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), also referred to in literature as ConvNets, are the application of NNs on images. This is done by using a set of filters which are convolved with an input feature map at layer $n$ in a sliding window movement. This operation produces a new feature map at layer $n+1$ for each kernel which contain more accurate feature representations (see Figure 2.8). For example, in the first layer the CNN might learn to detect random edge features. In the following layer it might learn to combine the edges into shapes, and in the final layer it might learn to recognise complete shapes into numbers. Development of NNs in general has been driven by the sizes of the available datasets. This is especially true for CNNs as dataset sizes had been central to the development of better and more advanced CNNs [96].

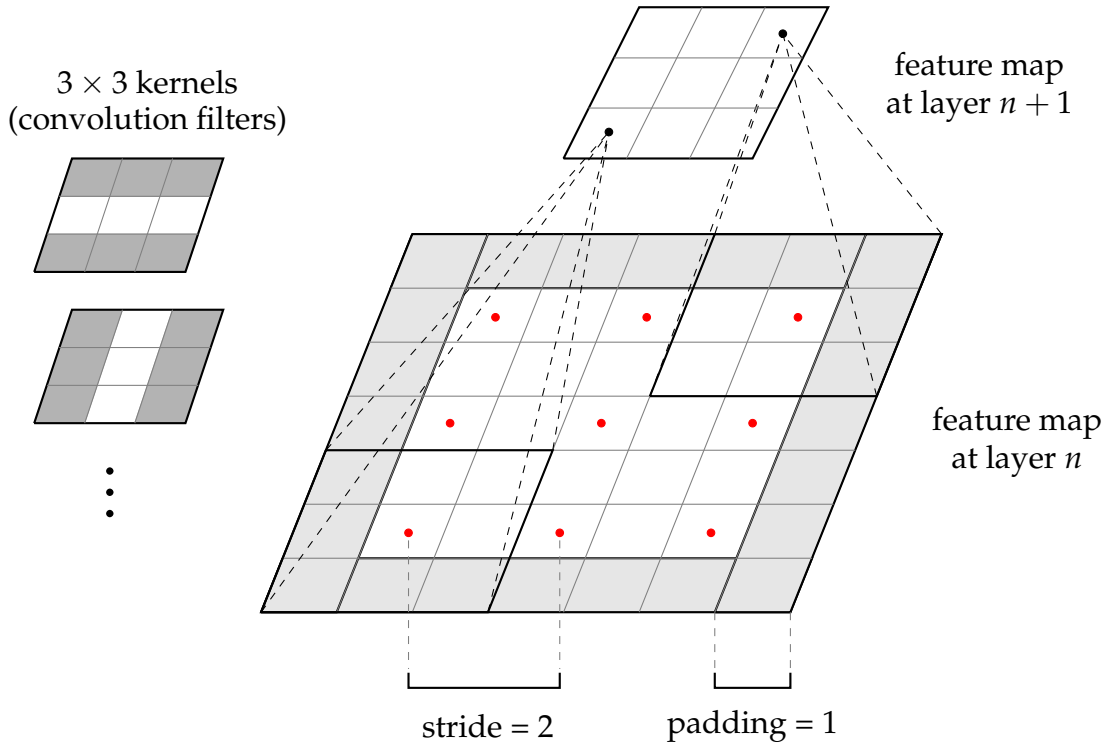


FIGURE 2.8: Convolution operation in a CNN. Each kernel (left side) is convolved with the feature map in layer $n$ to produce a new feature map in layer $n+1$, which contains a denser representation of features in the input feature map. The kernel has a stride of 2 pixels and the input feature map is padded with zeroes.

The first application of CNNs was on the recognition of handwritten numbers first in 1989 [97], and then improved a decade later [98]. The field of CNNs afterwards remained dormant for nearly two decades, mainly due to the lack of processing power [99]. In 2009 a large scale image dataset was released called ImageNet [95], which induced new progress in the field of CNNs. Shortly after the introduction of ImageNet, the first deep CNN for ImageNet classification was introduced, the network was called AlexNet [100]. The breakthrough which AlexNet made was that it proved that adding more layers to the CNN improved the visual classification capacity of the network, and it beat all competing hand engineered feature extraction methods by a big margin.

Many more deeper networks which surpassed AlexNet have been proposed since then, e.g. GoogleNet [101], VGG [102], ResNet [103], MobileNet [104, 105], U-Net [106], DenseNet [107].. etc, especially with the availability of large training datasets such as ImageNet in 2009 [108], CamVid in 2009 [109], KITTI in 2013 [110], ISPRS Potsdam in 2013 [3] and many others.

#### 2.3.3.1 Segmentation CNNs

Image segmentation is one of the many domains where CNNs have been successfully applied. Image segmentation is the classification of each pixel of

a given photo as belonging or not belonging to a certain category. Segmentation specialised CNNs can be classified in 3 categories:

1. Fully Convolutional Networks (FCNs): In 2015 it was demonstrated that replacing fully connected layers of a CNN with convolutional layers can adapt the network to better solve semantic segmentation. This framework was called the Fully Convolutional Network (FCN) [111] and it quickly became state-of-the-art outperforming earlier methods. Later many architectures based on FCNs appeared:

   (a) Encoder-Decoder architectures were first proposed in DeconvNet [112] and later improved in SegNet [113]. As can be seen in Figure 2.9, an encoder-decoder network is composed of two halves: an encoder which downsamples the input image capturing relevant representations, and a decoder which upsamples the result into a full resolution segmented image. The encoder is composed of convolution layers followed by max pooling layers. Max pooling is a downsampling layer identical to a convolution layer with the kernel being a max function which returns the maximum value within a square region. The decoder is composed of upsampling followed by convolution layers. The upsampling layers are guided either by switch variables [112], or by the pooling indices of corresponding layers [113] in the decoder. The decoder ends in a softmax classification layer.

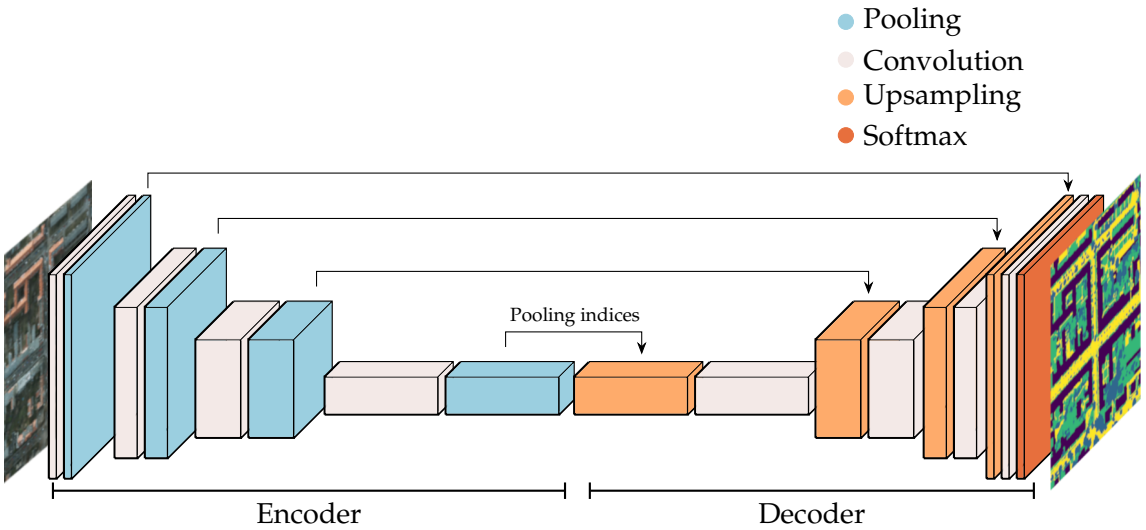


FIGURE 2.9: Encoder-Decoder CNN architecture.

   (b) U-Nets: Were first proposed in [106] and they quickly became state-of-the-art for medical image segmentation. U-Nets are also encoder-decoder networks, but their distinction from other networks is that with each upsampling step, the feature maps from the equivalent downsampling layer is copied and concatenated with the layer being upsampled, this is termed skip connections.

(c) DeepLab techniques: These techniques combine the encoder-decoder model with other algorithms [114, 115]. They combine techniques which improve the segmentation capacity of the network such as:

i. Atrous Convolution: increases the field of view of convolution filters without increasing the number of parameters, therefore reducing computational requirements which leads to efficient dense feature extraction.

ii. Atrous Spatial Pyramid Pooling (ASPP): used to detect objects at multiple scales. This operation is traditionally done introducing rescaled versions of the same image, but it could be computationally expensive. In ASPP, an input feature map is sampled at different rates which are combined before convolution, which simulates multiple field-of-views.

iii. Fully-connected Conditional Random Field (CRF): A fully connected CRF layer is added to the end of the network which improves its capacity to capture fine details.

2. Regions with CNN Features (R-CNNs): Another class of algorithms is R-CNNs which combine CNNs (eg. AlexNet) with region detection methods. The first paper of this kind was introduced in [116] and it was called R-CNN.

   R-CNNs have 4 main stages:

   (a) Generate category-independent regions using region detection algorithms.

   (b) Extract feature vectors using a CNN.

   (c) Classify the feature vectors using an SVM.

   (d) Finally segmentation is done by testing these features against both the foreground and the background.

   In general RCNN methods [116, 117, 118, 119] depend on object detection algorithms, which makes them very effective on photographs which contain objects such as people and vehicles, but they are not very practical for the segmentation of aerial photographs.

#### 2.3.3.2 Regression CNNs

Regression is the problem of predicting a continuous value. CNNs are adapted to solve regression problems by replacing the ending classification layer with a fully connected layer with a linear activation function. This allows the network to map the learned feature representations into continuous numerical values. There are many regression problems which are addressed by CNNs, they include: pose estimation [120], object tracking [96], image registration [121] and others.

It might be important to note that when transfer learning is applied on a CNN with the purpose of solving a regression problem, and when fine-tuned

correctly, the resulting network can achieve similar results to other existing networks which are engineered to solve specific tasks [122].

#### 2.3.3.3 Siamese CNNs

A Siamese neural network is a kind of convolutional neural networks which is composed of two branches [123]. The two branches share the same layers and the same variables (both biases and weights), and they learn to find a relationship between a pair of images. Each branch extracts relevant features and then both of them are connected to a number of fully connected layers, which learn to find the relation between the two input images. Siamese networks have been applied successfully to many problems such as visual odometry [124, 125] and image comparison [123].

### 2.3.4 State Of The Art

Deep learning continues to prove its success in nearly every field, vastly overcoming traditional methods in many disciplines. Segmentation continues to be at the heart of image understanding and recognition. It has been successfully applied in medical applications [126], remote sensing [127], autonomous driving and numerous other fields [128]. Regression networks have likewise been very successful. Today they are used in numerous applications [129] such as face landmark detection [130], pose estimation [131] and image registration [132].

## 2.4 Insect Navigation

In the field of biology, insect navigation strategies have been well studied and researched. Insects apply efficient techniques to navigate, collect food and survive using, comparatively with a mammal's brain, very simple neural structure equivalent to less than 0.01% the amounts of neurons that a human has.

This is a quick study about some of the observable navigation strategies that insects rely upon. One particular interesting strategy is called snapshot navigation, we will be investigating its employability on UAV navigation. It is possible to classify navigation into 3 hierarchical layers [133], ordered by complexity:

1. PI (Path Integration): which is a dead reckoning technique [134]. Insects use it to estimate their location using the past memorised landmark snapshots. PI is performing odometry by estimating the time and distance travelled in a certain direction towards the next destination, and eventually integrate all stored paths between different landmarks into a single vector that leads to destination.

2. Topological navigation, which is the ability to perform PI over many paths and the ability to combine those paths into a unified map. It

was argued in [134] that the navigation behaviour of insects can be explained using vectors and snapshots alone, and there is no need to employ a higher level metric (topological) map.

3. Survey navigation, which is topological navigation with the added ability to infer new paths and take shortcuts.

### 2.4.1 Ant navigation

It is well known that ants can establish and follow pheromone trails as means for navigation, we are not interested in this kind of navigation, however, ants have two other navigation mechanisms [135] which are of interest to us:

1. Path Integration (PI); also called vector navigation [136]: Ants maintain a vector that points home, and they do so by employing multiple techniques like: employing celestial and wind compasses [137], recognising olfactory landmarks (odours) [138], skyline [139] and polarised skylight observation (magnetic compass), and using odometry to estimate travelled distance using visual optic flow (both rotational and translational) and step count memory.

2. Remembering snapshots of landmarks: Generally ants try to match the actual view to the memorised one, this is called retinotopic matching [1] whereby ants match the observed scene to a stored snapshot in their memory.

It has been shown in [140] that ants use path integration to maintain a home vector; so even though that the ant might leave home in a winding path, its return path is always straight. The ant Cataglyphis fortis can perform successful homing after having travelled away from home for hundreds of meters. Its behaviour is very identical to the kind of behaviour which will be described later in Figure 3.1. It is notable to say that an ant does not use memorised places to recall directions, but to recall orientation of movement to be done next [141]

Insects like ants, including bees are sensitive to ultraviolet light. One of the advantage of this sensitivity is that it enhances the skyline view. In [142] a skyline retinotopic matching algorithm was proposed. Notably, the authors suggested that tagging snapshots with derived compass information (from wind, sun or sky polarised pattern) cannot explain all insect navigation abilities.

### 2.4.2 Bee navigation

Bees use the same mechanisms as ants to navigate [136]. Bees use the sun, visual memory (landmark snapshots & panoramic views) and odours.

A bee segments a route travelled [1] using PI (Figure 2.10). Bees also use panoramic views as snapshots or landmarks 2.11: [1] This means that the bees perspective is horisontal whereas ours is vertical (top-down view).

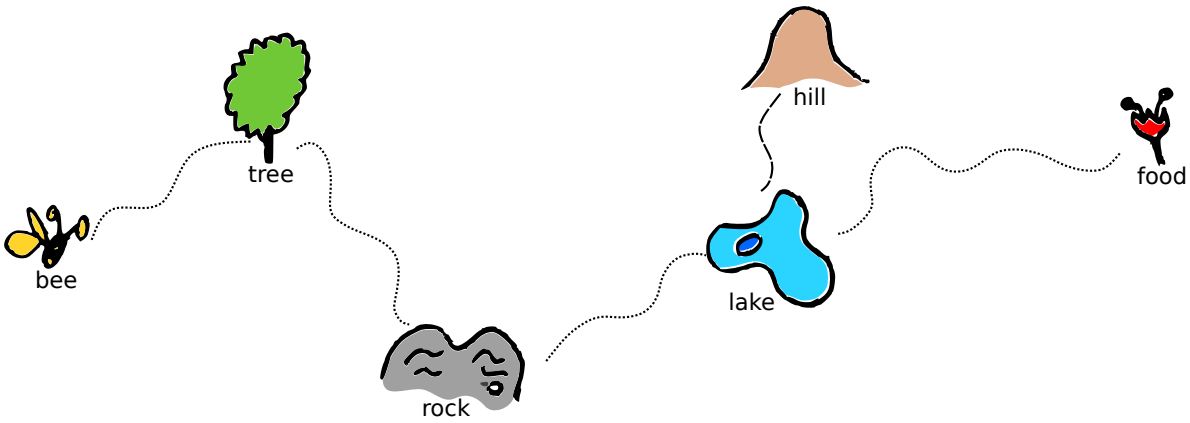


FIGURE 2.10: Path Integration in bees. The bee moves from one familiar location to the other so as to arrive to its destination.

Bees have colour constancy which allows them to recognise colours and see in day and night [143]. They can recognise colour, shape and texture; and they are sensitive to motion (optic flow) which enables them tell the foreground from the background, adjust speed and provide flight stability and control.

But do not depend completely on vision, and when the visual information is not sufficient to localise the goal, they use other cues like celestial cues, magnetic cues, and visual panoramic cues [143]. Similar behaviour has been recorded with ants.

### 2.4.3 Notable behaviour of other insects

It has been shown in [143] that wasps search for home entrance relative to a known landmark. And when this landmark is moved by a human, the wasp searches for nest entrance relative to the new position of the landmark. This highlights the importance of vision for insects. Similar behaviour has been documented with ants and bees. This means that there is a general tendency at insects to use landmarks as a cue that tells in which direction to head next.

Insects have many navigation advantages over the visual and cognitive systems commonly used by mammal; as they have sensitivity to different kinds of cues that help them navigate such as:

1. Near panoramic 360 degrees view, as the compound eyes they have provide them with larger field of view.
2. Sensitivity to ultraviolet light. [142]
3. Sensitivity to magnetism.

The question whether or not insects use the same kind of cognitive map that is used in mammals, is still contested/contented by some researchers until today [144, 145]. The use of a cognitive map requires more mental capacity than using a simple odometry model. And the opinion that bees do not use

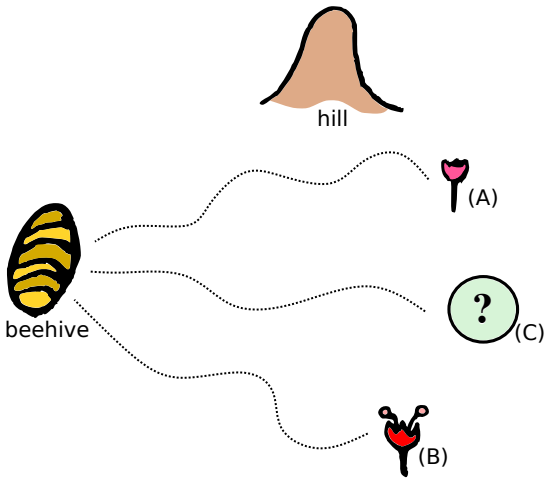


FIGURE 2.11: Bees homing behaviour. This experiment performed in [1] is evidence that bees use panoramic vision to aid in navigation. For example in this particular study, the bee knew how to navigate from both **A** and **B**, to and from the beehive. When the bee was captured and released in a point **C** between **A** & **B**, the bee navigated home directly, which is evidence that the bee must be using the visible hill to localise itself.

cognitive maps is supported by powerful logical argument: bees behaviours should be interpreted and explained by the simplest available solution. Authors in [145] contend that as long as bees behaviour can be explained by the simplest available models, we cannot assume a more complex model, which is the cognitive map.

### 2.4.4 Biomimetic navigation

Biomimetic navigation techniques are navigation algorithms inspired by insect navigation techniques. Most available visual biomimetic homing algorithms use panoramic photography which imitates insects' large field of view [146]. Some projects [147] develop customised sensors which are dedicated to the replication of insect vision, in the hope of replicating insects' navigation capacities.

Some navigation methods are based on reactive navigation using optical flow sensors [148]. This mechanism is similar to how bees and insects navigate [149]. The University of Sheffield has done some research in this direction. The Green Brain project [150] which ended in 2016 aimed to simulate the bee brain that is responsible for visual navigation, i.e. to do visual odometry. The simulated brain produced odometry measurements using optical flow for a drone flying down a binary striped/chequered corridor in a straight line, which mimics the behaviour of bees who fly in straight lines.

Brains on Board project, started in 2017 is a continuation of the Green Brain project. What is significant about both projects is that rather than using deep learning to produce a simulation model, it uses a pure mathematical

model, a reverse engineering of the central complex of the bee [151], to simulate the visual capacity of the bee, using optic flow [152].

Generally, existing biomimetic navigation algorithms can be classified into two kinds [153]:

1. Continuous metric navigation: estimates of the locations of landmarks as well as the robot's location are continually estimated and their relative metric locations mapped. An example of this kind of navigation includes all kinds of SLAM algorithms for example [54]. We have discussed this kind of navigation in Section 2.2.2.

2. Homing based navigation: this model of navigation is based on the snapshot model observed in ants and bees. Whereby current location is estimated based on a similarity measure between current view and memorised snapshot, and mapping is done using a graph model composed of edges and nodes, the nodes being the memorised snapshots, and edges representing the required transitions required to move from one snapshot to the other (translations and rotations). Implementations include [133] (using a panoramic stereo camera). This kind of navigation can be implemented using two methods [143]:

   (a) Feature detection and matching across snapshots. An example implementation is the Average Landmark Vector (ALV) [154] where each landmark is associated with a compass direction (vector), and homing is initiated by averaging multiple vectors associated with relative landmarks that lead to home. This mimics the PI navigation behaviour.

   (b) Image difference minimisation using intensity comparisons, this can be done for example using Sum of Squared Distances (SSD). This mimics the retinotopic snapshot matching in ants.

Some biologically-inspired techniques fall under the SLAM category because they perform loop closure:

1. One of the most successful methods in this category is M. Milfords RatSLAM, which is inspired by rodents nervous system. Open-RatSLAM [155] is a neural simulation (now open sourced) of a model that represents the rodents hippocampus, which is capable of navigating through and mapping an environment as large as a whole suburb [156].

2. Another algorithm is Sequence SLAM (Seq-SLAM) [157], also developed by M. Milford, and later Sequence Matching Across Route Traversals (SMART) [158] which are developed for wheeled road vehicles. The main principle of these systems is image matching, they store a map of the environment, the map being a sequence of images taken from a front-facing camera on a vehicle. The images are converted into low-resolution contrast patterns. What is notable here is that input live stream of images is compared to the stored dataset and vehicle location

is determined. This comparison isnt done on the level of a single photograph, but a sequence of photographs, which increases localisation accuracy, as each sequence becomes like a unique face feature.

**Snapshot navigation:** Given that insects use panoramic vision for navigation, particularly to do visual homing, we think that it is possible to produce the same behaviour using top down snapshots. Indeed, it has already been proved in [159] that it was possible to perform navigation by matching top-down images. Authors simulated insect navigation by matching satellite images. Their algorithm is based on insect retinal matching, as top down satellite images are pixelated and compared against a map (also satellite image). This kind of navigation will be investigated and explored more detail in Chapter 4, where a localisation system based on this model will be introduced.

## 2.5 Notable Navigation Techniques

Some of the older navigation techniques depended on the use of artificial beacons deployed in the environment, also called ground control points (GCPs), this is done to accurately perform triangulation so as to localise the UAV during flight. Newer methods use direct georeferencing [22] which combines inertial data with data coming from a satellite navigation system (GPS) to localise the drone.

Other UAV navigation methods depend on radio transmitters or cameras for localisation. Such systems can perform localisation very accurately. One system [160] shows an impressive performance between two drones juggling a ball to each other. This system uses 8 cameras to help detect a marker on the drone, which helps in accurately localising the drone.

Correct image registration is a fundamental part of any image navigation technique. As links between photographs have to be calculated accurately, and this is done by matching each pair of photographs with each other. Image registration is a well researched topic that has already achieved a high level of accuracy, for example in its use in image stitching [161]. The use of image matching in navigation is dependent on correct feature detection and matching.

An effective approach in feature matching is to consider the distribution of features and their relations to each other (like distances and angles). This technique is called Spectral Graph Matching [162]. And it is being applied to match 2D point clouds [163]. The importance of this kind of matching is that it is scale and affine invariant [164]. Tuples (triplets) of features can be formed, and matching procedure is done by comparing the angles formed by each tuple.

## 2.6 Summary

This chapter was a review about state-of-the-art visual navigation algorithms, as well as some related biomimetic navigation algorithms. In general

many biomimetic navigation algorithms were developed to perform panoramic navigation, this is because a panoramic view contains sufficient amount of information that is sufficient for many insect and animal navigation [165]. In the next chapter, we are going to describe a snapshot navigation technique and prove that it is sufficient to navigate using top-down images.

# Chapter 3

# Snapshot-based Localisation and Homing

The previous chapter (Chapter 2) was a discussion of state-of-the-art algorithms in visual odometry (VO), Simultaneous Localisation And Mapping (SLAM), deep learning and some related insect and biomimetic navigation techniques. In this chapter, localisation and homing using solely the snapshots taken from the UAV is studied, with a focus on the application of VO on fixed-wing UAVs. The aim is to provide an understanding of VO algorithms for homing and to understand the efficacy of their application specifically on fixed-wing UAVs.

## 3.1 Introduction

Visual Odometry has been applied successfully on aerial robots and ground vehicles in numerous environment settings. For example, indoor robot localisation [166], outdoor vehicular localisation [167, 43], UAV localisation [168], etc. Homing, defined as the capacity to navigate back home (i.e. to the navigation origin), is generally done by tracking the stored environment features which had been collected along the way, until the initial location is reached. And generally this is how existing navigation systems do to head back home.

However there is an area which remains unexplored and uninvestigated. Today it is well known [140] from research in insect navigation that some insects despite navigating for long distances, head back home in a straight line.

Most of the localisation research is done in environments that are normally full of obstacles, for example, localising a handheld camera in closed indoor environments [166, 67, 70], aerial robots within relatively limited outdoor environments [60, 69, 80, 81] and forests [169], wheeled robots in natural environments [43, 167] and within cities [58, 83]. Because of this, the application of straight line homing remains unexplored. As it is not possible for a car within a city, to head back home in a straight line because of buildings. Nor it is possible for an indoor robot to navigate from room to room in a straight line because of walls and other obstacles.

In a normal SLAM robotic system, VO is done all the way from the beginning until the end of the navigated path. Loops are closed as part of the

SLAM system. When homing is traditionally done, the robot has to return back home by following the same way it has navigated (see Figure 3.1).

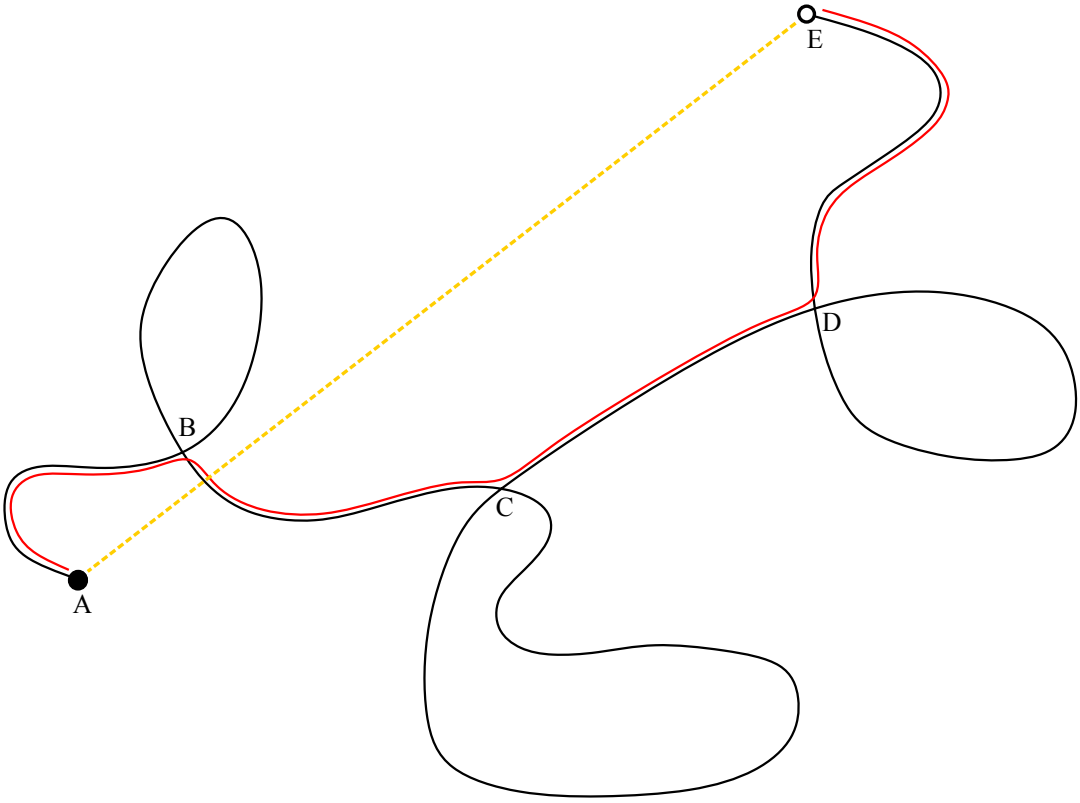


FIGURE 3.1: Planned return (homing) path, a hypothetical scenario. The **Black** path is a possible UAV travel path. Points *B*, *C* & *D* are intersections at which a loop closure occurs. In a SLAM system after loops are closed, the UAV returns home by following the **Red** path, which consists of stored feature points of the environment along the way. The **Orange** dashed line from *E* to *A* is the optimum homing vector.

But the case is different for fixed-wing UAVs. It is only possible to take a straight homing path in an open environment, therefore such navigation technique is applicable specifically on fixed-wing UAVs. Because they fly in an open air environment that is free from obstacles, it would be theoretically possible to solve the problem of homing in a straight line. The possibility of applying such homing methods on fixed-wing UAVs remains underexplored. Would it be possible to achieve such a goal using a top-down sensor alone? This inspiration comes from the way ants navigate and do homing [140, 154].

The dataset which will be used is an aerial dataset of top-down snapshots taken in an urban city using a fixed-wing UAV (Figure 3.2). Each snapshot has an overlap with the next as little as 49% and up to 74%. Traditionally, a video is used to do visual odometry where most frames are dropped and enough are retained to do the motion estimation. This kind of video for fixed-wing UAVs is not available (refer to the reasoning introduced in

Chapter 3.4.1). However the existing snapshots help in testing the minimal amount of overlap which is required for successful localisation.

FIGURE 3.2: Sample snapshots from Sensefly Merlischachen dataset [2].

The aim of this chapter is to investigate and propose suitable visual odometry methods to track the UAV. These methods rely on motion estimation between each two consecutive frames. The essential problem is to estimate the motion of the UAV across frames, therefore the aim is to propose the appropriate solution which can make this possible (see Figure 3.3).

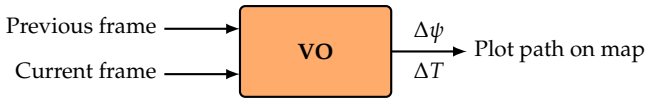


FIGURE 3.3: The motion estimation system (VO) which is proposed in this chapter. Its aim is to find the rotation ($\Delta\psi$) and translation ($\Delta T$) between two consecutive frames.

And since the camera of the UAV is perpendicular to the ground, estimating the location of the snapshot centre will be equivalent to the location of the UAV. Doing this repetitively across snapshots would draw a UAV path, which makes it possible to head back home in a straight line. This kind of homing would help reduce the UAV navigation time, therefore reducing energy consumption and safety risks of flying a UAV, and improving Visual Line Of Sight (VLOS) regulations.

The remainder of this chapter will be organised as follows: In Section 3.2 a literature review of related work is done. In Section 3.3, the problem is formulated and the planned solution is described, the initial results are outlined in Section 3.4 and finally a short conclusion in Section 3.5.

## 3.2 Related Work

In the previous chapter, visual odometry and SLAM methods were reviewed. The problem of visual odometry is solved by estimating the motion between each two consecutive frames. Therefore in this section, related motion estimation algorithms are reviewed.

The general workflow that applies to most algorithms is described in (Figure 3.4). After frames are captured, feature detection and matching are done between the current frame at time $t$ and previous frame at time $t-1$. Outliers are removed, the motion is then estimated using a Perspective-n-Points (PnP) triangulation algorithm. In the case of SLAM, loops are detected and smoothing is applied using either windowed or global Bundle Adjustment if necessary. This whole process is iterated in a loop. And the location of the camera, therefore the robot, is estimated.

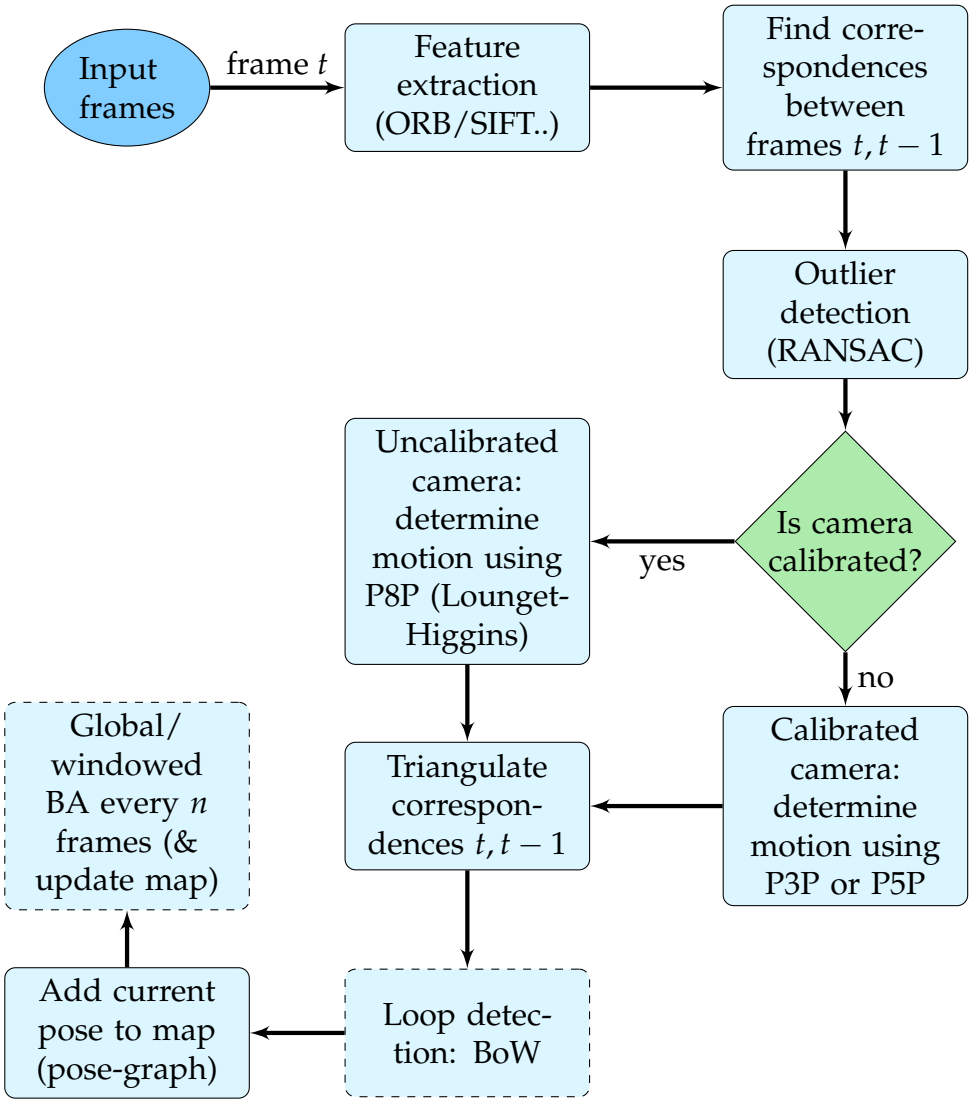


FIGURE 3.4: The general motion estimation workflow for motion estimation algorithms. To do VO, this workflow is repeated in a loop. Dashed boxes can be done without, as they represent the difference between SLAM (with) and VO (without).

The majority of VO algorithms have little variation from this general model. It is important to note here that these systems rely on consecutive snapshots with scale being estimated up to scale. This means that these systems are more susceptible to failure due to momentary occlusions. Enough overlap is required across 3 consecutive frames to allow for successful feature detection and matching.

To estimate the motion of the camera across two frames, different mutual points (correspondences) are needed to be detected and matched. The points are required to constrain the degrees-of-freedom (DoF) of the camera. This is a well studied problem called Perspective-from-n-Points (PnP), and it will be explained in detail in Section 3.3.

Naturally, a free moving rigid 3D object has an unconstrained motion described by 6 DoF (degrees-of-freedom). As more point on this object are pinpointed, more of its degrees of freedom will become constrained, as according to the following table (Table 3.1):

| **PnP** | **DoF** |
|---|---|
| P0P | 6 DoF |
| P1P | 4 DoF |
| P2P | 2 DoF |
| P3P | 0 DoF |

TABLE 3.1: Different number of correspondences are required to determine the camera pose across frames. To constrain a freely moving object with 6 Degrees-of-Freedom (DoF), 3 points are required using the Perspective-n-Points (P3P) algorithm.

This means that the minimum correspondences required to constrain a free moving object with 6 DoF is 3 points. In general, the more correspondences are found across frames, the more accuracy will be achieved for motion estimation. There exists other solutions that solve the PnP problem using more than 3 points, they begin with the P3P algorithm (using 3 points), others include, P4P, P5P, P6P, P7P and P8P.

It might be helpful to distinguish between motion/pose estimation and triangulation; as the first estimates camera location, and the second estimates points' locations in the real world.

In the cases when the camera does not have free 3D motion, it makes the motion estimation problem simpler to solve. Supposing that a vehicle with a mounted forward-facing camera moves a planar movement, it becomes possible to constrain the motion of the vehicle using a single point correspondence [170]. The non-holonomic constraint helps in removing one degree of freedom from the vehicle (as the rotation of the camera becomes tied to the rotation of the car).

What is needed to estimate the camera motion has now been described. And since motion estimation is at the heart of the problem of localisation, Table 3.2 provides a comparison between different localisation solutions. They are categorised into two classes: problems where it is possible to constrain

the motion, and those where the motion cannot be constrained, this is when the camera is assumed to have 6DoF.

| | **Models with constrainable motion** | **Models with unconstrained motion** |
|---|---|---|
| **Degrees Of Freedom** | · 2DoF (planar movement 2DoF, 1DoF of camera is tied with car depending on nonholonomic constraint). [170]<br><br>· 3DoF (2DoF planar movement, 1DoF camera) [166] | 6DoF (unconstrained camera moving in 3D space) [58, 62, 67] |
| **Bundle Adjustment (BA)** | None. | Windowed BA, Global BA. |
| **Motion estimation** | Essential matrix estimation from motion translation & rotation parameters [166, 170]. | Fundamental matrix estimation from correspondences, solved using SVD: P3P, P8P. |
| **Outlier removal** | · 1pt RANSAC [170] (few iterations).<br><br>· RANSAC. | RANSAC |
| **Scale** | Constant (up to a scale) | Observe a known size object or use extra sensor. |
| **Feature extraction** | Depending on algorithm (SIFT, Harris or KLT). | Depending on algorithm (SIFT, ORB..) |

TABLE 3.2: Comparison of the main categories of motion estimation algorithms with ours.

## 3.3 Proposed Method

As it was shown in the previous chapter, Section 2.4.4, most available biomimetic navigation techniques use panoramic snapshots to do homing, and they do so by minimising the difference between the view being seen, and the stored snapshot in memory. This means that from two views, the insect can deduce a direction of movement. This is identical to having a compass, hence it is called a visual compass, or snapshot navigation. The aim of this research is to simulate the same behaviour but on top-down snapshots. It has already been proven in [159] that top-down snapshots can indeed be used for navigation, and the goal here is to advance this study.

Another notable behaviour that we will be depending on is Path Integration (PI). PI is the process that ants use to average out all perceived movements (orientations and translations i.e. step counts) to form a heading direction that leads to home. PI is equivalent to dead-reckoning or visual-odometry (Section 2.2.1), and it is well known and well studied for insects like bees and ants [136, 140, 145]. Insects use this kind of navigation to return to their nests in a straight line, for example desert ant can return home in a straight line after having navigated up to 600m in a long winding path (see Figure 3.1). Considering that VO is the aggregation of a consecutive number of vector headings/directions, the final aim would be to form a total vector, the *homing vector*, which points directly to home (i.e. the navigation starting point). And what enables this kind of homing is the fact that fixed-wing UAVs fly in open space with no obstacles.

From here, our starting point is going to be photogrammetry. Photogrammetry literally means 'measuring graphically using light' and it has been around for more than a century. Today with the advent of UAV, photogrammetry scanning using drones is growing. Sensefly [20] and Wingtra [21] for example, are among many other companies that are helping grow this field today.

These UAVs depend completely on GPS for navigation, and they follow GPS paths that are pre-determined (waypoint navigation) [146] (see Figure 3.15). The snapshots they capture are taken solely for performing photogrammetry which is performed post-flight. This procedure is done on powerful computers due to its complexity and processing power requirements.

The goal of this chapter is to employ these snapshots for navigation. The snapshots have a minimum overlap with each other of 49% up to 74% (numerous datasets are available online [2, 171]), and this limitation needs to be accounted for. The aim is to give the UAV the capacity to do two things:

1. Deduce a compass direction from each consecutive pair of snapshots (i.e. form a vector).

2. Average out all of those vectors to form a global vector, the *homing* vector.

A fundamental problem that needs to be solved is the estimation of scale. Traditionally in VO algorithms, correspondences across 3 frames are found and used to estimate scale. But a small amount of overlap between the snapshots means that it would impossible to find correspondences across 3 frames, such as the datasets used in this thesis. Therefore traditional methods would fail. The reason is that it is possible to position objects in ways that the locations of the projections of some chosen points would not change in the image (see Figure 3.5). This phenomenon is related to pinhole camera model (central projection) which is explained later. To calculate scale, one of the following conditions must be satisfied [44]:

1. An extra sensor is needed to determine scale (IMU, GNSS..).

2. It can be calculated from observing an object in the scene with a known size.

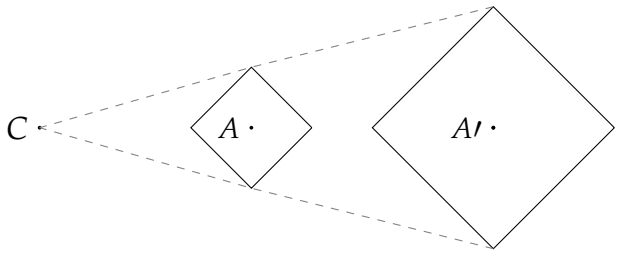


FIGURE 3.5: Impossibility of determining scale from two snapshots or image projections. The two differently sized squares (centres $A, A\prime$) are placed at different distances to the camera, such that they generate exactly the same projection image through the projection centre $C$.

There is little need to account for scale in the case of a fixed-wing UAV, such as the datasets used here; as the UAV flight height is near constant and the UAV maintains a fixed altitude. The recorded GPS coordinates from the snapshots for the Merlischachen dataset [2] ranges from 606-616m above sea level. If the varying altitudes of different parts of the village above sea level are taken into account, it would result in a UAV altitude of 162m from the ground. Therefore it is safe to assume that the UAV maintains a constant height, and scale is not needed to be estimated.

The final aim of this chapter is to do visual odometry on the UAV, eventually giving the UAV the capacity to return back to its take-off location automatically in emergency situations. The first stage is to extract a direction vector from each consecutive couple of snapshots. Then the averaging of all these vectors should result in an averaged vector that points home (Figure 3.6). In insects this is called Path Integration, which is equivalent to VO.

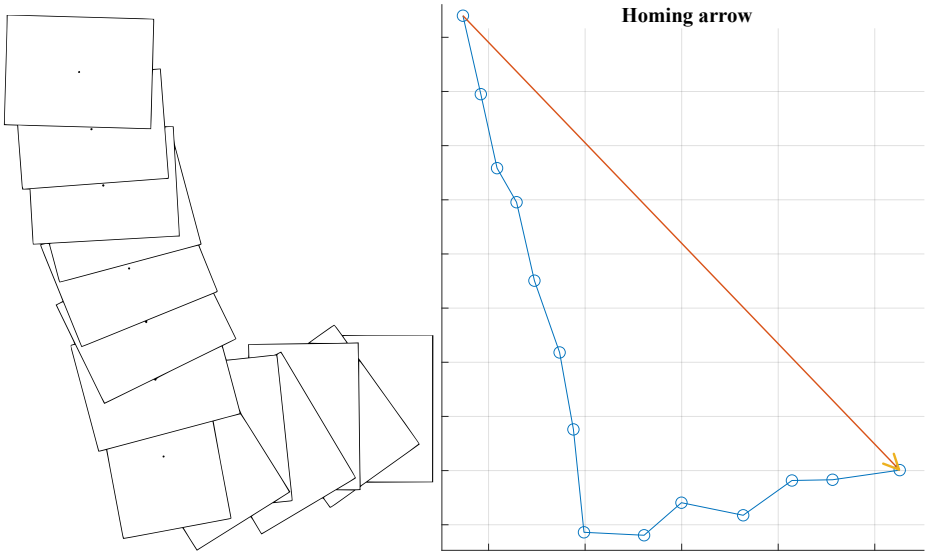


FIGURE 3.6: Left: Registered input images. Right: Transitions from one snapshot to the next; the visual compass/PI (considering relative distance & orientation). Scale can be ignored if the UAV is flying in a horizontal plane. A home vector (orange) is maintained constantly along the way.

There are two different ways to solve the motion estimation: Either using epipolar correspondences, P5P (since it is the most accurate), or P2P (minimal case solution). Postulate planar scene. BA not necessary. Constant scale (see Figure 3.5). SURF feature extraction. Or: using homographies. Both of these methods will be expounded in the coming sections.

### 3.3.1 Problem Formulation

For the case of this research, two assumptions are made: the UAV moves in a 2D plane (2 DoF), which means that a constant height throughout the flight is assumed. Indeed, this condition is satisfied in the datasets. Negligible variation in height is recorded by the GPS coordinates. Another assumption is also made, it is assumed that the camera tilts only on one axis (Yaw), therefore adding another DoF. In reality, as it will be discovered later, due to weather conditions the tilting assumption is not always preserved, as the wind causes disturbances to the flight of the UAV. These interruptions mean that the snapshot will not contain pure yaw rotation at the moment of exposure. However, it is assumed that the other angles (pitch and roll) are negligible. This provides simpler problem formulation at the cost of accuracy. In total, the UAV has 3 DoF, which means that a minimum of two correspondences are needed between each two consecutive frames to constrain the movement of the camera (therefore localise the UAV). One point to constrain the 2D movement of the UAV, and another point to constrain the yaw rotation (see Table 3.1).

Having 3 DoF and a fixed-wing UAV is a special case among existing solutions. For example, solutions existing for a free moving hand held camera in 3D space (6 DoF) [58, 67], a self driving car in a city (2 DoF) [83], a free-moving indoor robot (3 DoF) [166], or a rotor-wing UAV (6 DoF) [60].

The camera is solidly fixed to the centre of the UAV, this means that the orientation and location of the camera, are equivalent to the UAV position and orientation. The coordinate system which will be used to describe the position and orientation of the UAV is the $NED$ (North East Down) model [172]. It is centered on the UAV, and it is composed of 3 perpendicular axes: North ($N$), East ($E$), and Down ($D$) axes. The $N$ axis stretches from the centre of the UAV to its head. The $E$ axis stretches from the UAV centre to the eastern wing, and the $D$ axis stretches from the centre of the UAV towards the centre of the Earth. In the next section and given this model, Euler rotations will be used to describe the 3 different possible rotations using this coordinate system (see Figure 3.7).

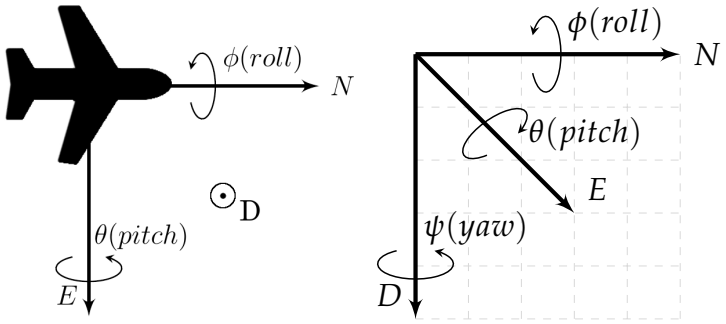


FIGURE 3.7: Camera coordinate system (in addition to the UAV 2D planar movement). Left: a top-down view. The Down vector is perpendicular on the page and is heading to the centre of the Earth. Right: NED north-east-down model & possible UAV rotations in mid-air.

There are two basic assumptions which will be made about the UAV: That it makes a planar flight (flies in a 2D plane parallel to the ground), and that it has a constant velocity with no acceleration. Since the UAV flies at an altitude higher than 100m, it is assumed that the ground is planar or near-planar for urban areas. The solution to this navigation problem requires a familiarity across a number of fields, including: coordinate system transformation (Section 3.3.2), computer vision (Section 3.3.3) and state estimation (Sections 3.3.4 and 3.3.5).

### 3.3.2 Euler Angles

In mid air, the camera can assume different orientations (pitch, yaw and roll). It is possible to describe these orientations using Euler angles. As can be seen in (Figure 3.7), roll is a rotation about the North axis (tail to head), pitch is a rotation about the East axis (wing to wing), and yaw is a rotation about the Down axis.

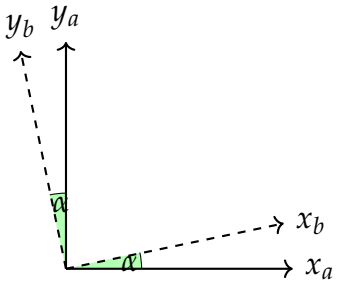


FIGURE 3.8: Rotation about center.

Let us consider the two perpendicular axes $x_a, y_a$ in a 2D plane (Figure 3.8). The two axes are rotated by angle $\alpha$. The rotation from $x_a$ to $x_b$ can be described using the following equation:

$$x_a = R_a^b x_b, \tag{3.1}$$

where $R_a^b$ is the rotation matrix that transforms $x_b$ into $x_a$. There are 3 possible situations:

1. First, assuming that this rotation is about the $N$ axis, then $\alpha$ will become equivalent to $\phi$ which is a roll rotation according to Figure 3.7. And $R_a^b = R_D^E$. It is then possible to estimate $\phi$ using the rotation matrix $R_D^E$ by substituting in equation 3.1:

$$D = R_D^E(\phi)E, \tag{3.2}$$

$$R_D^E(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix}. \tag{3.3}$$

2. Second, assuming that this rotation is about the $E$ axis, then $\alpha$ will become equivalent to $\theta$ which is a pitch rotation (Figure 3.7). And $R_a^b = R_N^D$. Now it becomes possible to estimate $\theta$ by substituting $R_N^D$ into equation 3.1:

$$N = R_N^D(\theta)D, \tag{3.4}$$

$$R_N^D(\theta) = \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix}. \tag{3.5}$$

3. Third, assuming that this rotation is about the $D$ axis, then $\alpha$ will become equivalent to $\psi$ which is a yaw rotation (Figure 3.7). And $R_a^b = R_E^N$. Yaw angle defines the heading of the UAV. Now it is possible to calculate $\psi$ by substituting $R_E^N$ into equation 3.1:

$$E = R_E^N(\psi)N, \tag{3.6}$$

$$R_E^N(\psi) = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

The three kinds of rotation matrices described above are special cases of orthonormal (orthogonal) rotation matrices. These matrices have the following special property:

$$R_a^b R_b^c = R_a^c, \tag{3.8}$$

this property allows the combination of multiple consecutive rotation matrices around different axes (such as the 3 types mentioned above, equations

3.3, 3.5 & 3.7) into a single matrix. This property will be useful for the calculation of the homing vector at a later stage.

$$R(\psi,\theta,\phi) = \underbrace{R_E^N(\psi)}_{\text{Yaw}}\underbrace{R_N^D(\theta)}_{\text{Pitch}}\underbrace{R_D^E(\phi)}_{\text{Roll}} = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix}. \tag{3.9}$$

When using Euler angles, and when the pitch angle is $\pm 90$; the yaw angle is undefined. This perfectly vertical position is rare in a fixed-wing UAV, as the UAV is not acrobatic. And this it is assumed that the UAV will never be in that position. This singularity is termed *gimbal lock*, and normally quaternions are used to solve it instead of Euler angles [172].

### 3.3.3 Pinhole Camera Geometry

Camera calibration is the estimation of intrinsic and extrinsic camera settings. The intrinsic settings are internal camera parameters such as focal length, sensor shape and size, and optical centre (principal point). The extrinsic parameters are the 3D coordinates of the physical camera location in the real world. Both instrinsic and extrinsic settings are required to be known for accurate camera motion estimation.

The purpose of camera calibration matrices is to project the points in the real world, into the image plane. Traditional cameras that are normally used on UAV use the pinhole projection model, and it will be described next.

#### 3.3.3.1 Pinhole Projection

The pinhole camera projection system is a system that projects points in the real world, into a planar projection plane (or image plane), through a single point (called a pinhole). As can be seen in Figure 3.9, $P$ is the world point with coordinates $(X, Y, Z)$, and it is being projected through the camera focal centre into the projection plane resulting in point $p(x, y)$. With $c$ being the optical centre (the centre of the image plane).

Note that the projection plane in reality lies behind the pinhole $O$, but for ease of description, it is regularly drawn inverted, i.e. it lies between the focal centre and the world, at a distance $f$ from $O$. This model is called *central projection*, and it is equivalent to *pinhole projection*.

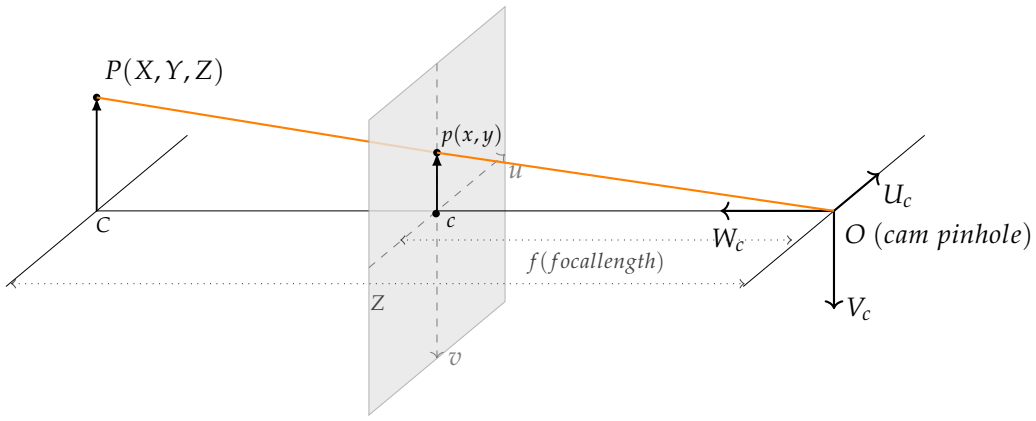


FIGURE 3.9: Pinhole camera projection model. A point $P$ in the real world is projected onto a projection plane (the grey rectangle), through a pinhole $O$. The distance from the projection plane to the pinhole $cO$ is the focal length $f$.

From Figure 3.9; by looking at the two similar triangles $Ocp$ and $OCP$, it is possible to derive the two equations:

$$\frac{x}{f} = \frac{X}{Z} \quad \Longrightarrow \quad x = f\frac{X}{Z}, \tag{3.10}$$

$$\frac{y}{f} = \frac{Y}{Z} \quad \Longrightarrow \quad y = f\frac{Y}{Z}. \tag{3.11}$$

Which means that given a 3D world point, it is possible to calculate its projection point given the 3D world point's coordinates. As can be noticed, any point on the line $pP$ has the same projection point $p$. To resolve this ambiguity, homogenous coordinates are used. They are discussed next.

#### 3.3.3.2 Homogenous coordinates

The homogenous coordinates of a point $p$ are any multiple of this point. For example the point $p(x, y)$ has the equivalent homogenous points $p_a(ax, ay, a)$ where $a \in R, a \neq 0$. The extra homogenous coordinate is added so that each pixel becomes a vector, this is relevant because in a 2D image, the exact depth of a pixel is unknown (as the depth values are dropped in a planar projection). To retrieve the original coordinates, both $ax, ay$ coordinates are divided by $a$.

To determine the depth value, a second image projection of the same scene from a different orientation is required. This will be discussed in detail later in Section 3.3.4, as well as the significance of using homogenous coordinates.

To project a 3D real world point $P(X, Y, Z)$ into a 2D image $p(x, y)$ (world coordinates to pixel coordinates), 3 stages of different transformations are required:

1. **World coordinates to camera coordinates:**

By looking at Figure 3.9, to calculate the accurate projection location, it is important that both $O$ and $C$ be aligned, where $O$ is the centre of the camera, and $C$ is the centre of the world. This alignment is done by using two kinds of transformation matrices: a rotation matrix $R$, and a translation matrix $T$. This operation can be described in matrix form as follows:

$$\begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{3.12}$$

which is equivalent to:

$$C = RT\,X. \tag{3.13}$$

Where $C$ is a point in camera coordinates, $R$ is a rotation matrix, $T$ is a translation matrix and $X$ is a real world point.

Both $R$ and $T$ can be combined into a single matrix called the *extrinsics matrix*, its purpose is to convert the world perspective to camera perspective:

$$M_{extrinsic} = RT = \begin{bmatrix} r_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}, \tag{3.14}$$

where $r_{3\times3}$ is a rotation that is composed of one or more Euler angles described earlier in section 3.3.2. And $t_{3\times1}$ is a translation vector.

2. **Camera coordinates to film coordinates:**

   From Figure 3.9, it is now needed to project all light rays passing through $O$ onto the grey projection plane. For example the ray extending from $P$ to $p$, or from $C$ to $c$. This is done using the two equations (3.10, 3.11). The two equations can be rewritten in matrix form, but to do so both $p$ and $P$ need to be described in homogenous forms:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}, \tag{3.15}$$

   which takes the form:

$$x = PX. \tag{3.16}$$

   Where $x$ is a point on the image plane, $X$ is a point in the real world, and the matrix $P$ is called the *perspective projection matrix*, and its purpose is to project points from 3D to 2D.

3. **Film coordinates to pixel coordinates:**

   Observing Figure 3.9, after $P$ is projected onto the grey projection plane resulting in $p$, and for an accurate projection, there is a need to align $p(x, y)$ which exists on the projection plane, with the pixel coordinates

of the resulting image $(u, v)$. Due to the high sensor density of the projection plane (accuracy in nanometres), perfect physical alignment of the camera pinhole $C$ with the centre of the projection plane $c$ is impossible to achieve. This is why the projected image needs to be aligned and scaled in software. This is done by multiplying film coordinates (the projected image) by the following affine matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.17}$$

This matrix can account for different factors like scale, principal point centre and skew. If it is multiplied by the film coordinates, it will result in pixel coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & c_x \\ 0 & s_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{3.18}$$

which is equivalent to:

$$u = Ax, \tag{3.19}$$

where $u$ is the equivalent pixel coordinates, $A$ is an affine matrix, and $x$ is a point in film/image plane coordinates. Further: $s_x, s_y$ are scale parameters, and $c_x, c_y$ is the principal centre shift amount. The skew parameters are set to zero and skew amount is ignored in the case above.

It is notable here to define the *intrinsics matrix*, also called the camera calibration matrix. Its function is to convert from the camera coordinates, to pixel coordinates. And it is produced by multiplying the affine matrix $A$ (equation 3.17), by the perspective projection matrix $P$ (equation 3.16):

$$I = A_{3\times3}P_{3\times4} = \begin{bmatrix} f_x/s_x & 0 & c_x & 0 \\ 0 & f_y/s_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}_{3\times4}, \tag{3.20}$$

where $A_{3\times3}$ is an affine matrix, and $P_{3\times4}$ is the perspective projection matrix. The intrinsics and extrinsics camera matrices combined are referred to as the *camera matrix (M)*.

Finally, the full pinhole camera projection model can be described and the three transformations above can be combined using the following equation (according to Figure 3.9):

$$\underbrace{\begin{bmatrix} u_{\text{img}} \\ v_{\text{img}} \\ 1 \end{bmatrix}}_{\text{Pixel coords (u)}} = \underbrace{\begin{bmatrix} f_x/s_x & 0 & c_x & 0 \\ 0 & f_y/s_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Intrinsics matrix (I)}} \underbrace{\left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{array}\right]}_{\text{Extrinsics matrix (E)}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\text{3D World coords (X)}}, \tag{3.21}$$

equivalent to:

$$u = I_{3\times4}E_{4\times4}\ X, \tag{3.22}$$

where $u$ is an image point in pixel coordinates (homogenous form), $X$ is a real world point in world coordinates (also homogenous form), $I$ is the intrinsics matrix, and $E$ is the extrinsics matrix.

#### 3.3.3.3 Camera Calibration

Camera calibration is the process of estimating the intrinsic and extrinsic matrices of a camera. By looking at equation 3.22, it becomes possible to construct the intrinsics matrix given default camera specification values. The focal length, the focal centre and the scale are usually provided with most pinhole cameras sold today.

Given this information, it is possible to estimate the intrinsics matrix. But this matrix will never be accurate enough for accurate projection required for pose estimation, as during the manufacturing process of the camera, it is not possible to align the sensor chip on the camera board accurately enough to align the aperture centre with the focal centre of the chip. This is why manual calibration needs to be done.

Possession of the camera is required for manual calibration, which is done by taking multiple photographs of a chequerboard pattern and finding the *vanishing points* which can be used to estimate both intrinsics and extrinsics matrices.

This process gives an estimation of the *camera matrix* $M$, and it helps in minimising reprojection errors as much as possible; which leads to greater accuracies when the camera is used for pose estimation.

#### 3.3.3.4 RANSAC

RANSAC (Random Sample Consensus) [173] is a random sampling algorithm which iteratively collects random samples from a given set, with the purpose of discerning outliers. It can be described by the equation:

$$N = \frac{\log(1-p)}{\log(1-(1-e)^s)}, \tag{3.23}$$

where $N$ is number of samples, $e$ is the probability that a point is an outlier, $s$ is number of points in a sample and $p$ is the desired probability that a good sample is acquired. This equation can be derived as such: The complement probability of $e$ is the probability of an inlier $(1-e)$. $(1-e)^s$ is the joint probability that $s$ sample points are all inliers. The complement probability $1-(1-e)^s$ is the probability of choosing $s$ points which are not all inliers. The term $(1-(1-e)^s)^N$ is its joint probability across $N$ samples. The complement probability of this term is the probability of choosing N samples and not all of them are outliers $1-(1-(1-e)^s)^N = p$. This leads to: $(1-(1-e)^s)^N = 1-p$, after calculating the logarithms we get the equation above.

RANSAC is intended to be used to find the best fit of a mathematical model (for example a regression) to a given data. This is done by taking a random number of samples from the data, and testing them against the present mathematical model. The samples that satisfy the model would be the inliers. It is used widely in the field of motion estimation to refine inlier (true) correspondences across images and discard outliers.

### 3.3.4 Epipolar Geometry

As it was mentioned earlier, a second pinhole camera projection from a different angle is needed to estimate depth. The so called *epipolar constraint* is the only available geometric constraint which can be used for estimating motion between two different snapshots of the same scene [174]. This motion can be estimated up to scale, which means that scale cannot be determined from 2 images alone, this is was explained in Figure 3.5.

Even though that taking a single 2D projection of a 3D scene drops the depth values, but there is a possibility of estimating the depth values using two snapshots of the same scene from 2 different angles. By triangulating a point as seen from the two different camera perspectives, with the first camera location, both depth of the point and the second camera's pose can be estimated. This is called Epipolar geometry, and it is described in Figure 3.10.

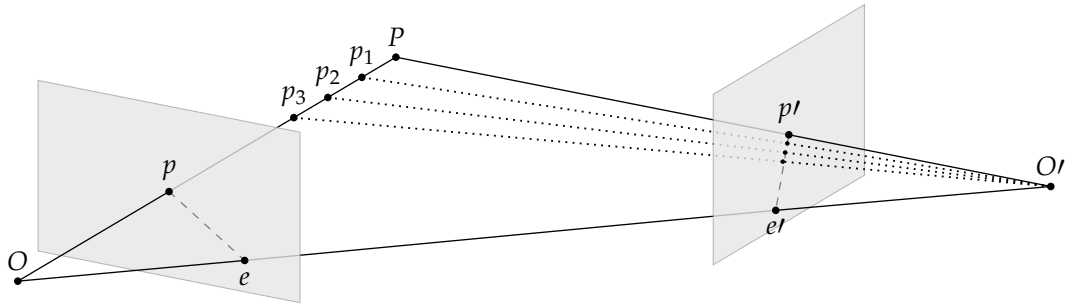


FIGURE 3.10: An epipolar geometry setup. $OO\prime$ is called the baseline. $e, e\prime$ are the epipoles, which are the intersection of the baseline with the image planes. $pe$ and $p\prime e\prime$ are epipolar lines (an epipolar line is the set of all points $\{p\prime, p_1\prime, p_2\prime, p_3\prime, .., e\prime\}$ that form a line in one projection plane, and which correspond to a single point in the other projection plane). Epipolar lines lie on the epipolar plane $OPO\prime$.

The setup of cameras described in Figure 3.10, is a setup of two pinhole cameras $O$, $O\prime$ taking a projection of the same point $P$ from two different points of view. $P$ has two projections $p$ and $p\prime$ in the two grey image planes belonging to the two pinhole cameras. The depth of the real world point $P$ cannot be determined from a single projection $p$, as $P$ can be considered lying anywhere on the segment $pP$, for example $p_1, p_2, p_3$... To determine the depth of $P$, another pinhole projection of the point $P$ is needed from a different point of view $O\prime$. This is when the location of $P$ can be estimated using the projection $p\prime$.

As can be noticed that each projection point in one image plane $p$ corresponds to a line in the other image plane $p\prime e\prime$; this line is called the epipolar line. This is the purpose of the other camera $O\prime$, as using the projection $p\prime$ the location of the point $P$ can be triangulated and its depth estimated.

#### 3.3.4.1 Essential Matrix

The essential matrix is a matrix that relates two camera projection planes to each other (for example, the two grey planes in Figure 3.10). It can be defined and derived as follows: Starting with an equation that describes the relation (rotation $R$ & translation $T$) between one image plane (snapshot) $I_R$ to another $I_L$:

$$I_L = RI_R + T, \tag{3.24}$$

cross product by $T$:

$$T \times I_L = T \times RI_R + \underbrace{T \times T}_{0}, \tag{3.25}$$

the cross product between a vector and itself is zero, therefore the second term of the right hand side is vanished. Now do a dot product by $I_L$:

$$\underbrace{I_L . (T \times I_L)}_{0} = I_L . (T \times RI_R), \tag{3.26}$$

the left hand term totals to zero, because the result of the cross product is a vector which is perpendicular to both $T$ and $I_L$. A dot product with $I_L$ which is perpendicular to it, gives 0. See Appendix A for more details about vector properties. The dot product on the right hand term can be expanded as a matrix multiplication by substituting $I_L$ with its transpose $I_L^T$; and the cross product can be written in matrix form:

$$0 = I_L^T \lfloor T_X \rfloor RI_R, \tag{3.27}$$

which can be summarised as:

$$0 = I_L^T EI_R, \tag{3.28}$$

where $I_L$ is the left image projection plane (snapshot), $E$ is the essential matrix and $I_R$ is the right image projection plane.

#### 3.3.4.2 Fundamental Matrix

Its concept was first introduced by Longuet-Higgins [175], which is a matrix of rank 2 that relates a single point from one camera to the epipolar line in the other camera. It is derived as follows. Start by substituting equation 3.13 ($P_{cam} = RTX$) into equation 3.22:

$$P_{img} = KP_{cam}, \tag{3.29}$$

where $P_{img}$ is a point coordinates in image plane (in pixels), $P_{cam}$ is the point in camera coordinates, and $K$ is the intrinsics matrix. This implies:

$$P_{cam} = K^{-1} P_{img}. \tag{3.30}$$

Now if equation 3.30 is plugged into 3.28:

$$\left(K_l^{-1} P_l\right)^T E \left(K_r^{-1} P_r\right) = 0, \tag{3.31}$$

where: $K_l, K_r$ are intrinsic matrices of left and right frames, $P_l, P_r$ are points' coordinates in image plane for left and right frames respectively, and $E$ is the essential matrix.

Now decompose the transpose of the multiplied matrices (refer to Appendix A for matrix properties):

$$P_l^T \underbrace{\left(K_l^{-1}\right)^T E K_r^{-1}}_{\mathrm{F}} P_r = 0, \tag{3.32}$$

$$P_l^T F P_r = 0, \tag{3.33}$$

where $P_l$ is a point in the left image, $P_r$ is a point in the right image, and $F$ is the fundamental matrix. It is equivalent to:

$$\begin{bmatrix} a & b & 1 \end{bmatrix}_{1\times 3} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}_{3\times 3} \begin{bmatrix} a\prime \\ b\prime \\ 1 \end{bmatrix}_{3\times 1} = 0, \tag{3.34}$$

where $F$ is a matrix with 7 degrees-of-freedom (DoF), 3 for position, 3 for orientation and 1 for scale. If the matrices are multiplied, they result in an equation with 8 unknowns. To solve this equation, Richard Hartley proposed the 8-point algorithm [176] where 7 other similar equations procured from 7 other correspondences (matched points) are needed. The 8 equations produce a linear system and it is finally written in matrix form and the matrix is solved using Singular Value Decomposition (SVD). This solution is called the Longuet-Higgins eight point algorithm [175]. It is solved using Direct Linear Transform (DLT) [176], and its derivation is explained later at the homographies discussion in Section 3.3.5, as a homography is solved using DLT.

An important thing to note here is that the intrinsics matrix is required to calculate the fundamental matrix. But it is not required to find the essential matrix.

#### 3.3.4.3 Motion Estimation

Also known as extrinsic calibration, which is the problem of estimation the location and orientation of the camera in the real world given some known coordinates of a scene [177]. There are 3 kinds of motion/pose estimation described in detail in the excellent visual odometry paper by Scaramuzza [44]. They will briefly described here:

1. 2D-to-2D: this is the case of estimating the motion from projecting a planar scene (2D), into a projection plane of a single camera (2D). It is possible to use this case if the ground is considered as planar.

2. 3D-to-3D: this case is used for stereo cameras. Since stereo cameras can produce a depth map of the scene using triangulation (structure from motion SfM), the goal is to estimate motion given a 3D scene, and to project it into a 3D depth map.

3. 3D-to-2D: the case of projecting a 3D scene into a 2D image plane. This the situation of this research, and the goal is to generally minimise the reprojection error.

#### 3.3.4.4 Reprojection error minimisation

Reprojection error is the process of finding the distance between points in the image plane, and their reprojected triangulations. I.e. after points are triangulated, they are reprojected into the image plane (using the fundamental matrix). The distance between each reprojected point and the original point is calculated. Reprojection error minimisation is the process of minimising this value, and it is expressed by the equation:

$$T_{k,k-1} = \arg\min_{T} \sum_{i} \|u_i - \pi(p_i)\|^2, \tag{3.35}$$

where $T_{k,k-1}$ is the transformation between the two frames $k$ and $k-1$, $u_i$ is the $i$th image point, and $\pi(p_i)$ is the reprojection of the triangulated point $p_i$.

Pose/motion estimation is the problem of estimating the extrinsic calibration matrix, using point correspondences for an object viewed from two perspectives. For the 3D-to-2D problem mentioned above, the extrinsic matrix is estimated by minimising the reprojection error, which is known as the Perspective-from-n-Points (PnP) problem mentioned earlier. The iterative process of minimising equation 3.35 non-linearly is called Bundle Adjustment (BA).

### 3.3.5 Homographies

Also called projective or perspective transformation. A homography is the most general form of a geometric transformation, which may take different kinds:

Transformations are either 2D to 2D, or 3D to 3D; and they include: translation, Euclidean (translation + rotation), similarity (translation, rotation & scaling), affine (translation, rotation, scaling and skewing) and projective transformation (which only preserves straight lines). Each transformation of these includes the previous one, this means that the projective transformation is the most general kind of transformations, and it is called a *homography* (see Figure 3.11).

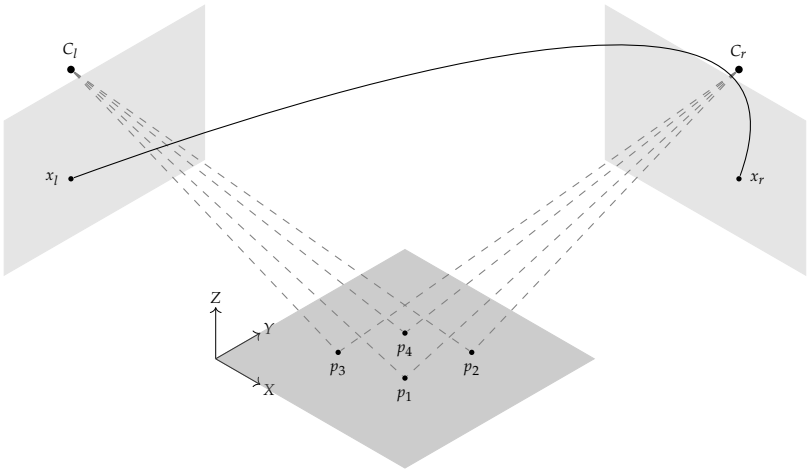


FIGURE 3.11: A homography.

The homography matrix is a special case of the Essential matrix, and because it does not consider the geometry of the scene, its use is limited to 2 kinds of mappings:

1. 2D-to-2D mapping (planar scene): The homography matrix maps two snapshots of the same scene to each other (or the scene to its projection image), given that the projected scene in the two snapshots is planar. Such applications include: image stitching (mosaics) and texture rendering.

$$x_l = Hx_r, \tag{3.36}$$

   where $x_{l[3\times1]}, x_{r[3\times1]}$ are the homogenous forms of a point in the left and right projection planes respectively and $H_{[3\times3]}$ is the homography matrix. And just like the fundamental matrix, it can be solved using SVD. Its derivation and solution are provided below.

2. 3D-to-2D mapping: An orthographic (perpendicular) projection that drops the depth of a 3D object into a projection plane to acquire its 2D image. It is suitable for objects whose depth is within a small range relative to the distance from the camera [177]. This property makes them suitable for the application on fixed-wing UAVs with top-down cameras (which is the case in this thesis).

   Note that for planar surfaces, 3D to 2D mapping reduces to 2D to 2D; and that homography reduces to a similarity transformation when the camera is facing the planar scene (when the Down axis is perpendicular to the ground). Its most prominent application is VR (virtual reality).

The deriviation and solution of a homography are as follows. Starting from equation 3.22:

$$x = I_{3\times4}E_{4\times4}\ \ X, \tag{3.37}$$

where $x$ is pixel coordinates, and $X$ is world coordinates. $I$ and $E$ are the intrinsics and extrinsics matrices respectively. Assuming the planarity of the

scene, it becomes possible to set the $Z$ coordinate of the real world point $X$ to 0. This is the main reason why homography is a special case of the essential matrix. The equation takes the form:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x/s_x & 0 & c_x & 0 \\ 0 & f_y/s_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}. \tag{3.38}$$

This results in cancelling the third column of the matrix $E$. Also, the fourth column of $I$ cancels the fourth line of $E$. This results in:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x/s_x & 0 & c_x \\ 0 & f_y/s_y & c_y \\ 0 & 0 & 1 \end{bmatrix}_{3\times3} \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}_{3\times3} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}, \tag{3.39}$$

the matrices $I$ and $E$ are multiplied and combined into one matrix to produce the homography matrix $h$:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}. \tag{3.40}$$

By taking the projection equations of a world point into each frame:

$$\begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} = H_1 \begin{bmatrix} X \\ Y \\ W \end{bmatrix}; \qquad \begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = H_2 \begin{bmatrix} X \\ Y \\ W \end{bmatrix}, \tag{3.41}$$

by substituting one of the equations above into the other:

$$\begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = \underbrace{H_2 H_1^{-1}}_{\mathrm{H}} \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix}, \tag{3.42}$$

the homography $H$ is a $3 \times 3$ matrix that relates points in one camera frame to points in the other camera frame:

$$\begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix}. \tag{3.43}$$

Considering that the two points are in homogenous coordinates:

$$x\prime_2 = \frac{x_2}{w_2}; \qquad y\prime_2 = \frac{y_2}{w_2}, \tag{3.44}$$

$$x\prime_2 = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}w_1}{H_{31}x_1 + H_{32}y_1 + H_{33}w_1}; \quad y\prime_2 = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}w_1}{H_{31}x_1 + H_{32}y_1 + H_{33}w_1}, \tag{3.45}$$

this is equivalent to:

$$x\prime_2 H_{31}x_1 + x\prime_2 H_{32}y_1 + x\prime_2 H_{33}w_1 - H_{11}x_1 - H_{12}y_1 - H_{13}w_1 = 0, \quad (3.46)$$

$$y\prime_2 H_{31}x_1 + y\prime_2 H_{32}y_1 + y\prime_2 H_{33}w_1 - H_{21}x_1 - H_{22}y_1 - H_{23}w_1 = 0. \quad (3.47)$$

Now write this equation in matrix form after setting $w_1 = 1$ (homogenous coordinates):

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x\prime_2 & x\prime_2y_1 & x\prime_2 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & y\prime_2x_1 & y\prime_2y_1 & y\prime_2 \end{bmatrix}_{2\times 9} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ \vdots \\ H_{33} \end{bmatrix}_{9\times 1} = 0, \quad (3.48)$$

which is an equation with 9 unknowns (the matrix $H$). To solve this series, 6 more equations are required. It is possible to acquire those equations from 3 more correspondences:

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x\prime_2 & x\prime_2y_1 & x\prime_2 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & y\prime_2x_1 & y\prime_2y_1 & y\prime_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{8\times 9} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ \vdots \\ H_{33} \end{bmatrix}_{9\times 1} = 0, \quad (3.49)$$

which takes the form:

$$XH = 0, \quad (3.50)$$

where $X$ is an $8 \times 9$ matrix. It is possible to reduce the unknowns to 8 by assuming that $H_{33} = 1$. Now it becomes possible to solve 8 equations with 8 unknowns, and this is done using SVD.

A homography cannot be used to calculate the camera pose/orientation in all cases, i.e. it is not accurate enough to be used for motion estimation in all possible cases. If homographies are used for image stitching, to be effective for estimating camera motion accurately, either one of 2 requirements are necessary (see Figure 3.12):

1. When the camera rotates purely about a single axis (regardless of scene planarity) without changing its location. I.e. all snapshots have to be taken from exactly the same camera location [161]. This would make the homographies relating the snapshots to each other pure *Euclidean* transformations.

2. When the scene is planar. Here there are two cases:

(a) If the camera moves horizontally parallel to the scene in 2D with its vertical axis perpendicular to the scene. This would make the homographies consist of pure *Euclidean* transformations.

(b) Or if the camera moves in a free 6DoF motion, making the homography transformations *affine*.

In optimal conditions, the first condition of the second requirement applies to the case of fixed-wing UAV with top-down camera. Therefore a solution using image stitching based on this assumption will be described in the following Section (Section 3.3.6), and later tested and evaluated in Section (Section 3.4).

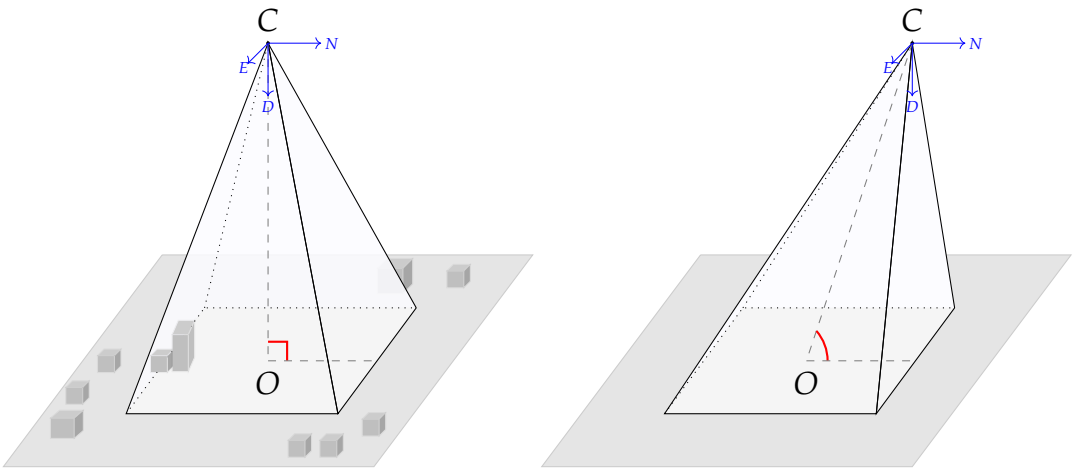


FIGURE 3.12: It is possible to use homographies to estimate UAV motion in two cases: either when the camera axis $D$ is perpendicular to the ground despite scene planarity (left image), or when the scene is planar regardless of camera orientation (right image). In the case on the left, a similarity matrix can be used to estimate UAV motion, as the location of the UAV would be estimated exactly above the image center.

#### 3.3.5.1 Singular Value Decomposition

Every real matrix can be factorised into singular values and singular vectors. It can be decomposed into a product of three matrices, as according to the following equation:

$$A_{[m\times n]} = U_{[m\times m]} D_{[m\times n]} V^T_{[n\times n]}, \tag{3.51}$$

where $U$ and $V$ are orthonormal (orthogonal) matrices whose columns are called left-singular and right-singular vectors respectively. $D$ is a diagonal matrix whose diagonal elements are called the singular values. The SVD factorisation is used to decompose the essential matrix into constituent transformations. For example, the rotation can be acquired from the $U$ and $V$ components.

### 3.3.6 Proposed Solutions

Two algorithms are proposed for predicting the location of a fixed-wing UAV, which is done estimating the motion between each two consecutive top-down aerial snapshots:

1. Using image stitching: As it was shown in Section 3.3.5, when the depth of the world objects is negligible compared to the distance to the camera, planar homographies can be used. Also, since the world objects is constantly in front of the camera, it can be assumed that the world is planar. This condition cannot be assumed in the case of a free moving camera in a 3D world, for example, a camera mounted on a car. As the objects constantly move from the front to the back of the camera. This means that it is possible to stitch snapshots with each other, with the location of the UAV falling above the centre of each snapshot. Therefore stitching the snapshots with each other would be equivalent to estimating the 2D location of the UAV. An algorithm for this is devised as follows (Algorithm 1).

**Algorithm 1** Image stitching algorithm

1: Get first frame $k$
2: Detect SURF features
3: Get new frame $k+1$
4: Detect SURF features
5: Find correspondences between $k, k+1$
6: Refine inliers using RANSAC
7: At least 4 evenly distributed correspondences are needed between the two frames.
8: Estimate planar homography according to equation 3.49.
9: Extract rotation & translation matrices using SVD
10: Update pose-graph.
11: Repeat from (3).

2. Using Perspective-from-n-Points (PnP): As it was explained in Section 3.3.1, it is possible to assume that the UAV has 3 DoF in total. This means that 2 points are required to constrain all of them. Therefore theoretically, a P2P algorithm would be sufficient. However, it was proved [178] that the P5P algorithm outperforms all others. Therefore it will be used here.

**Algorithm 2** Motion estimation algorithm

1: Get first frame $k$
2: Detect SURF features
3: Get new frame $k+1$
4: Detect SURF features
5: Find correspondences between $k, k+1$
6: Refine inliers using RANSAC
7: Collect 5 evenly distributed correspondences between the two frames
8: Estimate essential matrix $E$ using P5P [178]
9: Decompose $E$ into rotation & translation matrices
10: If $k \mod n == 0$: use Windowed-BA for motion estimation refinement every $n$ frames. This step is optional.
11: Update pose-graph.
12: Repeat from (3).

Both of these algorithms rely on feature detection and matching. For the purpose of this research, Scale-Invariant Feature Transform (SIFT) feature detection algorithm is being used. Numerous other algorithms could be used as well, they include detection algorithms, such as, Harris [179], Speeded-Up Robust Features (SURF) [180], Oriented FAST and Rotated BRIEF (ORB) [181], Binary Robust Invariant Scalable Keypoints(BRISK) [182], Features from Accelerated Segment Test (FAST) [183] and many others. And correspondence matching such as Ultra-Robust Feature Correspondence Via Unilateral Grid-Based Clustering (UGC) [184], Grid-Based Motion Statistics (GMS) [185] and others.

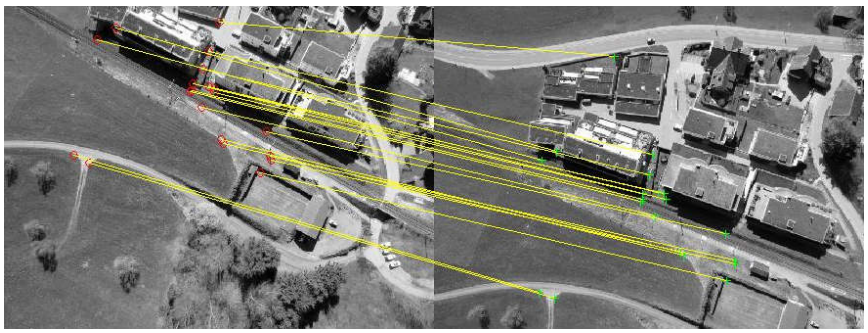

FIGURE 3.13: Feature detection and matching using SIFT.

## 3.4 Performance Evaluation

In this section, the experimental setup of the UAV and the datasets used will be detailed in Section 3.4.2. Then the results from both image stitching and PnP algorithms will be discussed and evaluated in Section 3.4.3.

### 3.4.1 Datasets

The datasets which were chosen as the object of this research contain top-down aerial photos (snapshots) captured by a ground facing camera mounted on a fixed-wing UAV (see Figure 1.2). The consecutive snapshots are limited by the small overlap amount and the big exposure time between them. The *reasoning* why these snapshots are chosen to be used rather than video is twofold:

1. Even though that there exists numerous well performing algorithms that can do localisation using videos (these algorithms are reviewed in Chapter 2, such as, [58, 56]), but technically, they are dependent on correspondences across 3 frames, a condition which is satisfied in most available datasets (e.g. KITTI [110], ISPRS [3]..). But this requirement makes these systems susceptible to failure as identified in [53]. We aim to show that correspondences across 2 frames are enough to do successful localisation and enable homing, which means that having a continuous stream of visual information with high overlap is not a necessary requirement for this task. This is discussed in more detail throughout the thesis.

2. Proper evaluations and testing require having access to the ground truth of the data that is used. A good ground truth of the path taken by a fixed-wing UAV requires having access to accurate GPS technology. This technology is only available in the higher end lines of UAVs which are expensive. And given that there are considerable licensing, costs and skills required to acquire such dataset and to fly a fixed-wing UAV over long distances in the UK, it was decided to use existing datasets for the purpose of this research. These datasets are numerous and they are available online from UAV manufacturers which are used for aerial mapping (photogrammetry), such as, [20, 21]. Such datasets consist of consecutive snapshots which are overlapped with each other up to a certain percentage. Despite the fact that video datasets are numerous for indoor or vehicular navigation, they are rarely publicly available for long-range top-down camera UAVs.

### 3.4.2 Experimental Setup

The datasets which are used in this thesis are acquired from a light weight fixed-wing UAV, with a top-down camera. The dataset which will be used in this chapter is acquired in April 2013, it is composed of overlapping top-down photograph of a village in Switzerland called Merlischachen [2]. The UAV flight altitude is 162m. The UAV flies at a constant speed, and it captures a snapshot every 4 seconds as a minimum time.

Two different paths are going to be experimented on. The first one is a straight path (see left of Figure 3.14). It is composed of 21 consecutive snapshots, the total navigated distance is 1080m. The second path is a longer winding zig-zag path. It is composed of 100 snapshots, and the total navigated distance is 5300m (see right of Figure 3.14).

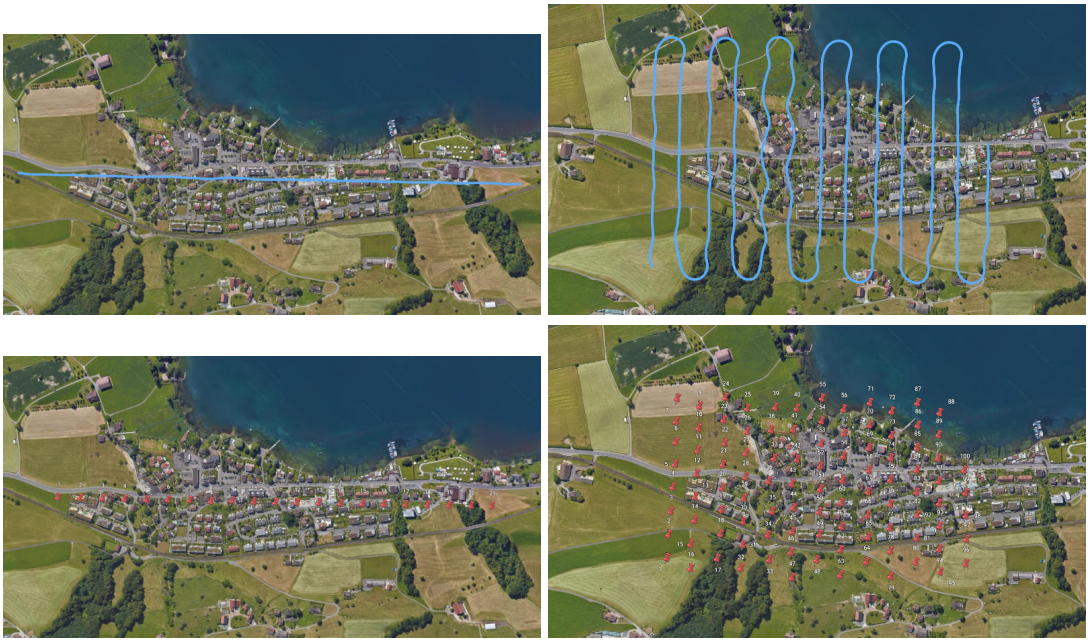

FIGURE 3.14: Top row: The UAV flight paths used from Sensefly Merlischachen village dataset [2]. Bottom row: The down projected locations of the recorded GPS UAV location at the moment of taking the snapshot (numbered). The left column is the first datset which is a straight path composed of 21 snapshots. The right column is composed of 100 snapshots. GPS coordinates are plotted on Google Maps.

The 2D ground-truth locations of the UAV are extracted from the recorded GPS coordinates. They GPS coordinates are recorded at the instant of capturing a snapshot and they are plotted in Figure 3.15. These coordinates represent the paths in Figure 3.14, and they will be used to evaluate the motion estimation accuracy.

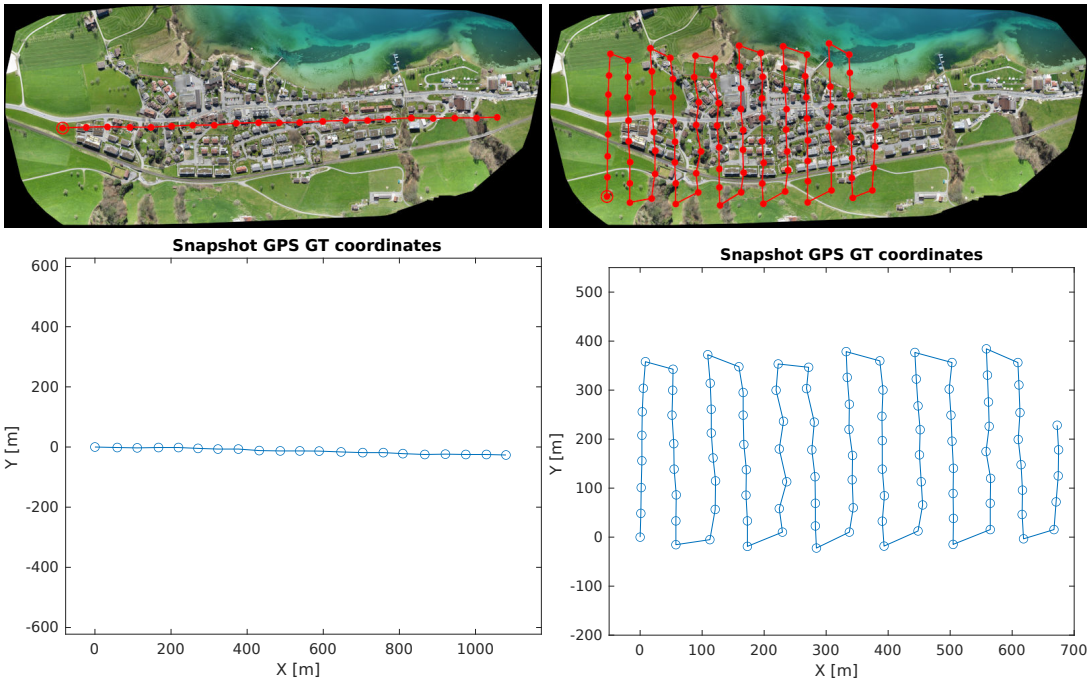


FIGURE 3.15: Top line: The 2D ground truth of the UAV locations at the moment of exposure, extracted from the GPS geotags of Sensefly Merlischachen dataset [2]. The blue line is the path of the UAV, and the small blue circles are the recorded UAV locations. Path lengths are 1080m (left) and 5300m (right).

More details about the captured snapshots can be found in Table 3.3. They include the overlap amount, the time difference, and the navigated distance between each two consecutive snapshots. The average time between snapshots is 5.4 seconds, the average distance is 54.1 meters and the average overlap is 64%. The minimum overlap is 48.8% and the maximum is 75.1%.

| ID | Snapshot name | Time (s) | Nav. Dist. (m) | Overlap (%) | ID | Snapshot name | Time (s) | Nav. Dist. (m) | Overlap (%) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0928 | - | - | - | 26 | 0953 | 6 | 69.4 | 63.1 |
| 2 | 0929 | 5 | 53.8 | 48.8 | 27 | 0954 | 5 | 57.0 | 61.7 |
| 3 | 0930 | 4 | 53.6 | 62.1 | 28 | 0955 | 5 | 54.7 | 64.9 |
| 4 | 0931 | 4 | 49.4 | 66.9 | 29 | 0956 | 4 | 53.1 | 67.7 |
| 5 | 0932 | 5 | 53.3 | 68.5 | 30 | 0957 | 4 | 45.9 | 71.0 |
| 6 | 0933 | 4 | 48.8 | 68.8 | 31 | 0958 | 5 | 53.9 | 53.5 |
| 7 | 0934 | 5 | 53.6 | 73.7 | 32 | 0959 | 10 | 66.0 | 56.8 |
| 8 | 0935 | 5 | 51.9 | 64.3 | 33 | 0960 | 5 | 56.0 | 50.5 |
| 9 | 0936 | 10 | 55.7 | 59.2 | 34 | 0961 | 4 | 51.0 | 75.1 |
| 10 | 0937 | 5 | 54.0 | 60.1 | 35 | 0962 | 5 | 51.8 | 68.5 |
| 11 | 0938 | 4 | 49.6 | 67.3 | 36 | 0963 | 4 | 51.8 | 69.2 |
| 12 | 0939 | 5 | 54.1 | 68.8 | 37 | 0964 | 5 | 52.2 | 71.3 |
| 13 | 0940 | 4 | 48.6 | 66.7 | 38 | 0965 | 4 | 52.1 | 69.9 |
| 14 | 0941 | 4 | 51.9 | 68.3 | 39 | 0966 | 5 | 52.5 | 72.6 |
| 15 | 0942 | 5 | 51.8 | 60.3 | 40 | 0967 | 10 | 55.7 | 65.5 |
| 16 | 0943 | 4 | 51.1 | 67.8 | 41 | 0968 | 5 | 55.5 | 55.2 |
| 17 | 0944 | 12 | 65.0 | 64.6 | 42 | 0969 | 4 | 52.3 | 69.3 |
| 18 | 0945 | 5 | 55.0 | 55.7 | 43 | 0970 | 5 | 55.7 | 64.3 |
| 19 | 0946 | 5 | 53.5 | 71.3 | 44 | 0971 | 4 | 51.1 | 71.4 |
| 20 | 0947 | 6 | 69.0 | 59.4 | 45 | 0972 | 5 | 54.3 | 63.4 |
| 21 | 0948 | 4 | 53.3 | 59.2 | 46 | 0973 | 4 | 50.8 | 64.0 |
| 22 | 0949 | 6 | 65.9 | 64.9 | 47 | 0974 | 5 | 53.6 | 56.4 |
| 23 | 0950 | 5 | 53.1 | 59.9 | 48 | 0975 | 11 | 64.7 | 58.5 |
| 24 | 0951 | 13 | 49.8 | 66.5 | 49 | 0976 | 5 | 54.2 | 56.0 |
| 25 | 0952 | 4 | 48.8 | 64.2 | 50 | 0977 | 4 | 49.1 | 67.4 |

TABLE 3.3: Sample snapshots from Sensefly Merlischachen dataset [2] with corresponding time, distance and overlap amounts. Time is difference between curren frame and previous frame measured in seconds. Distance is calculated using the actual 2D UAV position (using the GPS coordinates recorded by the UAV), and not the distance between the snapshot centres. Overlap is calculated between current and previous snapshot.

### 3.4.3 Results and Discussion

Results from the two proposed algorithms will be discussed and evaluated in this section. First results from image stitching are described in Section 3.4.3.1, then motion estimation using image geometry from PnP algorithm is described in Section 3.4.3.2.

#### 3.4.3.1 Homography Based Image Stitching

Hypothesising the ground as being flat, would enable the usage of homographies to align the snapshots to each other. Therefore it would enable using image stitching to estimate the motion of the UAV, in short this would allow localising the UAV.

As it was shown in Section 3.3.5, for image stitching to work, either one of two assumptions have to be true. For the case of this research, the planarity of the scene is assumed. This would mean that relationship between each snapshot and the next overlapping snapshot, is a similarity transform. A similarity transform is a combination of rotation, translation and scale, as described in Section 3.3.5, and it is represented by the following matrix:

$$A = \begin{bmatrix} s_x * cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & s_y * cos(\psi) & 0 \\ t_x & t_y & 1 \end{bmatrix}, \tag{3.52}$$

where $\psi$ is the yaw rotation angle, $s_x, s_y$ are scale values on both $x$ and $y$ axes; and $t_x, t_y$ are the translation values. Since the UAV altitude is constant, it would mean that the scale does not change. Therefore it is safe to assume that $s_x = s_y = 1$. This would render the transformation into a Euclidean transformation:

$$A = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ t_x & t_y & 1 \end{bmatrix}. \tag{3.53}$$

The matrix $A$ is a Euclidean transformation matrix which maps the previous snapshot to the current snapshot. It assumes that the Down axis is perpendicular to the ground (as in Figure 3.12 (left)), and it is estimated using a RANSAC approach [186]. The location of the UAV is determined by the centre of the snapshot which would be at a constant height exactly above the centre. Therefore by matching two snapshots to each other, it becomes possible to know the path that the UAV took.

After running this algorithm on 2 different datasets, the results in Figure 3.16 were acquired.

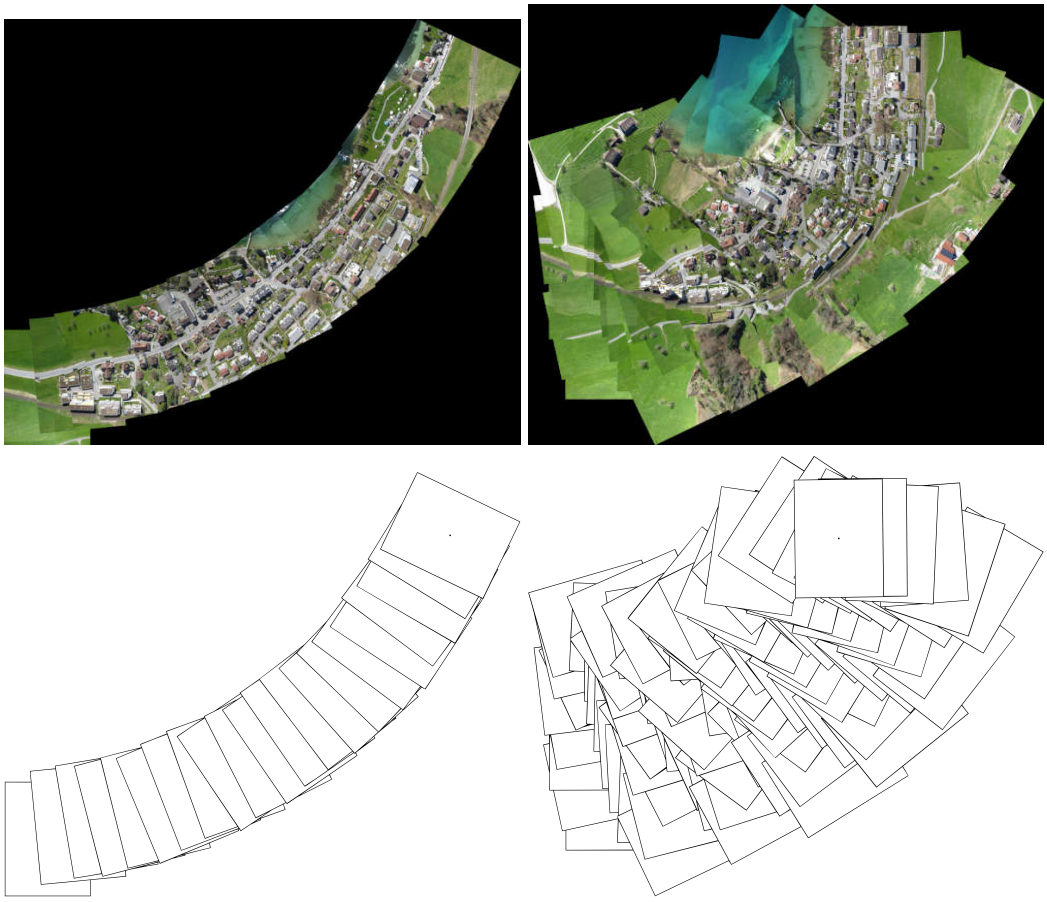

FIGURE 3.16: Graphical depiction of applying homography stitching to the 2 datasets (first dataset on the left column, second one on the right).

The snapshots centres (resembling resulting UAV path) are plotted against the ground truth in Figure 3.17:

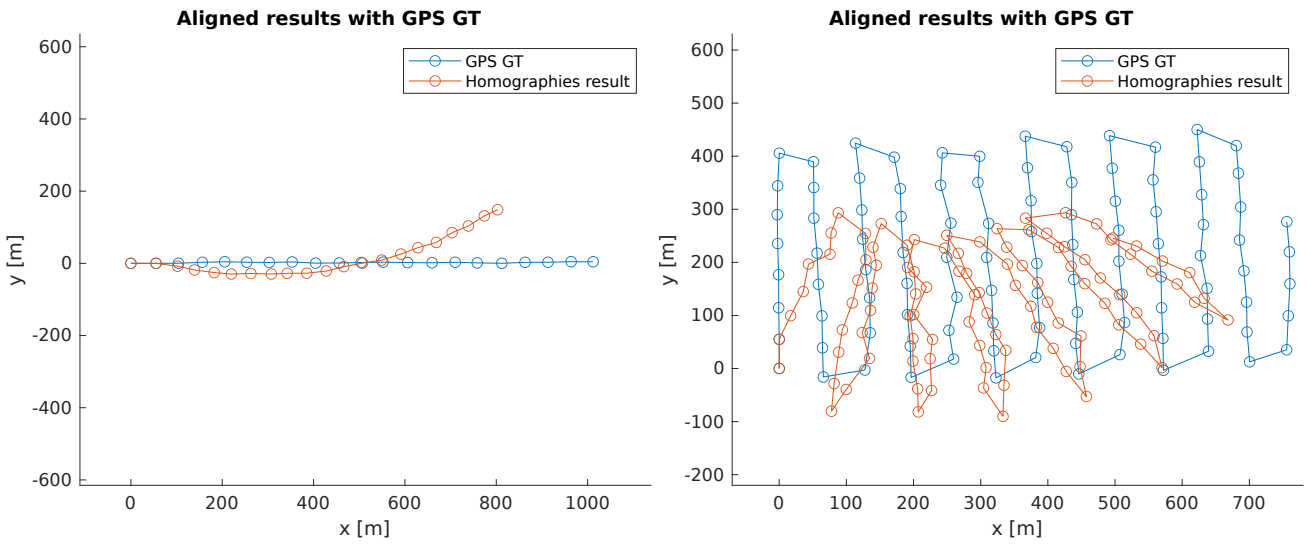


FIGURE 3.17: Comparison the results of homography image stitching to the GPS ground truth.

The resulting path plotted in Figure 3.16 does not accurately follow the actual path of the UAV. This can be seen when compared to the ground-truth in Figure 3.15. In the case of the straight path dataset, the calculated UAV path drifts by an average of 96 meters (8.9%), and a final drift value of 254 meters (23.5%). The second longer zig-zag dataset recorded an average drift of 127 meters (2.4%), and a final drift of 262 meters (4.9%). Despite the fact that the second dataset is considerably longer than the first one (5300m versus 1080m), but its final drift values are less than the other dataset. This is because even though the second path is longer, but the UAV goes back and forth, reaching a point which is located at a Euclidean distance of 750m away from the initial take-off location. The drift values are plotted as follows in Figure 3.18:

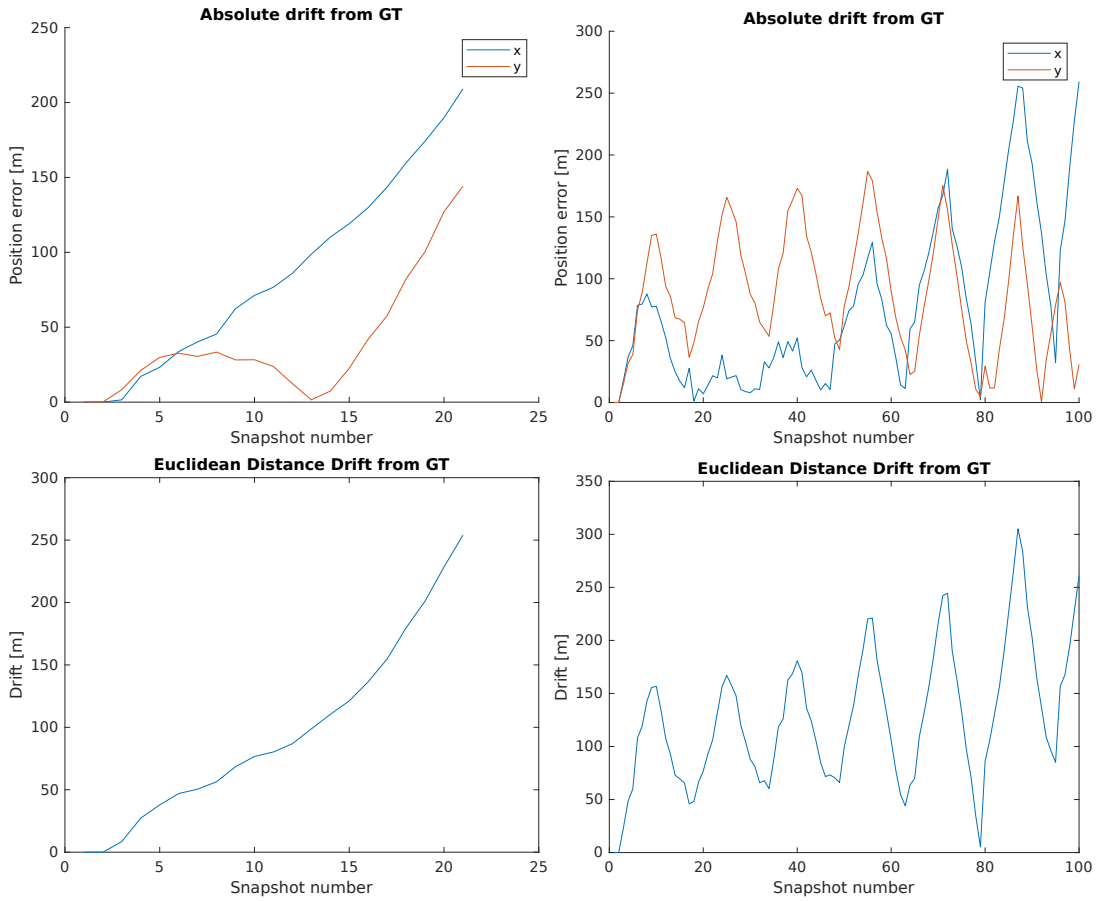


FIGURE 3.18: Drift values resulting from using the image stitching algorithm.

After further investigation and deeper inspection of the snapshots, it is noticed that they are not perfectly aligned with the recorded GPS UAV locations. I.e. the centre of each snapshot, does not fall straight below the tagged GPS location. I.e. the camera does not point perfectly downwards. This is explains the amounts of drift which is seen, especially the zig-zag motion between snapshots (right of Figure 3.17). Therefore, the cause of the drift can be concluded as follows: The camera perpendicularity hypothesis which was made earlier must be invalid for the following reasons:

During flight, and at the instant of taking a snapshot, the orientation of the camera, particularly the Down axis, is not perfectly perpendicular to the ground. This is caused in part due to perturbations exacted on the body of the UAV due to irregular air flow. This causes the Down axis to shift off from the perpendicular position, even though that the camera is facing the ground (see right Figure 3.12).

This means the the orientation is no longer a pure yaw rotation, and more rotations are introduced (pitch and roll) into the orientation equation. And since a yaw rotation cannot accommodate the other rotations, inaccuracy is introduced, therefore the drift. The conclusion of this is that the orientation is more complex and it cannot be captured by a single yaw rotation. Therefore a different route is needed to be taken to address this problem. This is when the epipolar geometry had to be investigated.

#### 3.4.3.2 Epipolar Geometry Results and Discussion

As mentioned earlier in Section 3.3.4, the only available geometrical relation between two overlapping images which can be used to estimate motion is the epipolar constraint [174]. The proposed solution can be represented by the following workflow in Figure 3.19.

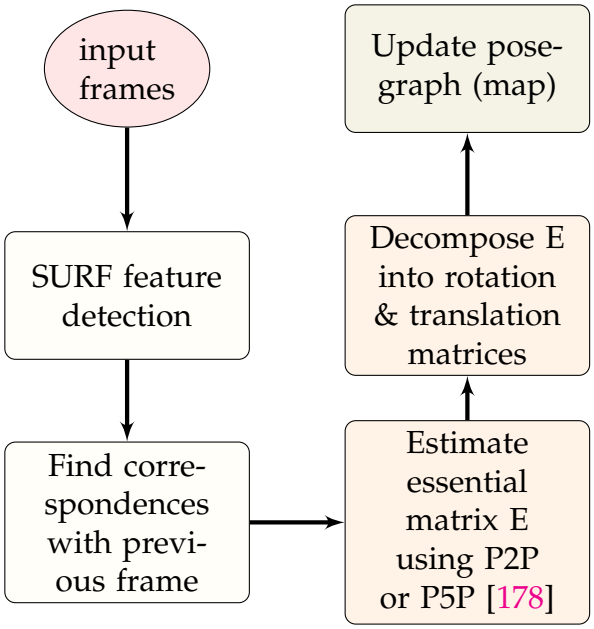


FIGURE 3.19: Motion estimation workflow (following Algorithm 2).

The proposed algorithm starts by feature detection across two frames, detected features are matched to find the correspondences. After this a number of the correspondences are used to estimate the essential matrix.

The minimal number of correspondences which can be used is 2 (as argued earlier). But it was shown in the previous section that the rotation of the UAV camera is not purely composed of yaw rotation, therefore 2 correspondences will not be enough. It was shown [178] that the P5P algorithm

is more efficient than other algorithms which require more correspondences such as P6P-P8P. Therefore five of the correspondences are used to estimate the essential matrix using P5P. This can be done using SVD as described in section 3.3.5.1. Since P5P algorithm estimates 6DoF movement, the depth values as well as the roll and pitch rotations are intentionally dropped.

Note that it is not possible for us to use P3P because it requires correspondences across 3 consecutive frames as it matches 2D features from the 3rd frame, to the 3D triangulated features from the first two frames. And since 3 points are enough to constrain the motion of a 3D object, P3P is usually adopted when correspondences across 3 frames are available.

Given the essential matrix, it becomes possible to decompose it into corresponding rotation and translation matrices as described in [178]. This is also done using SVD. When the decomposition is done the locations are plotted on a 2D map (pose graph). Since the times between each two snapshots is between 4-5 seconds, the algorithm execution is real-time. Noting that the most time consuming step in the proposed algorithm (as well as most VO algorithms) is the feature detection and matching stage.

The algorithm was applied on both datasets, once using Bundle Adjustment (BA) globally (on all frames), and another without using it. The results are plotted along with the ground truth and they can be seen in Figure 3.20.

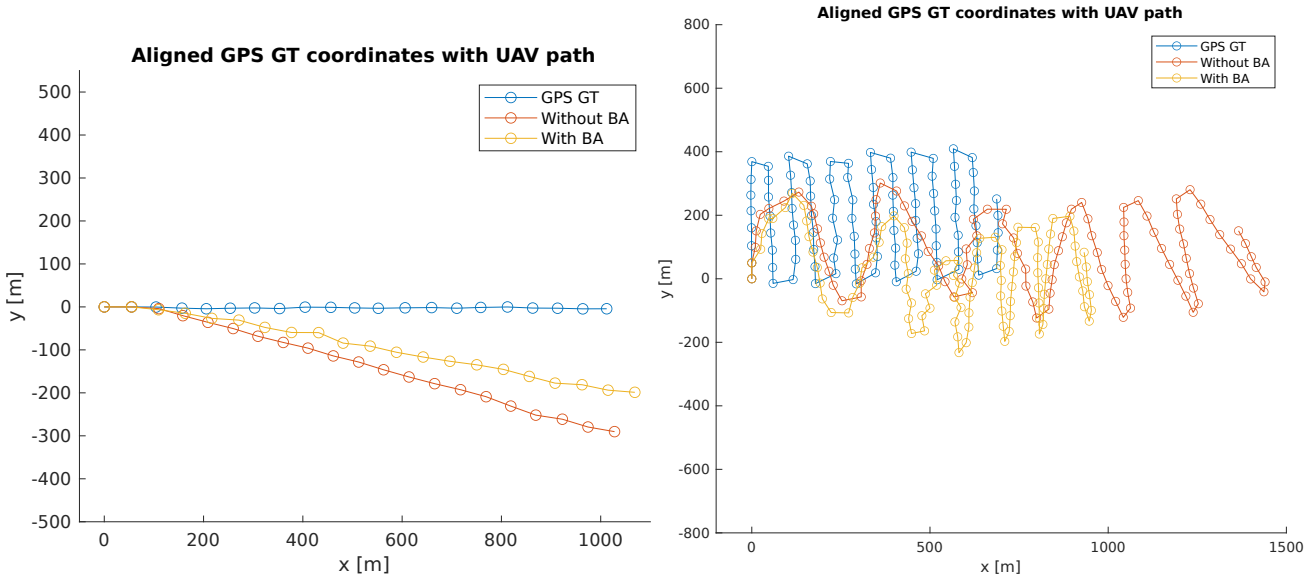


FIGURE 3.20: Mapping the results of Algorithm 2, with global Bundle Adjustment (BA) and without it. The red and yellow paths represent the estimated path taken by the UAV, plotted along the ground truth (blue). The algorithm is tested on two datasets: the straight path (left) and the zig-zag path (right).

For the first straight path dataset, an average drift of 132m (12.2%) on Non-BA P5P, and 96m (8.8%) on BA P5P were recorded. The final drift values were 286m (26.5%) and 203m (18.8%) respectively. On the second longer dataset, the average drift was 466m (8.8%) on Non-BA P5P, and 303m (5.7%)

on BA P5P. The final drift values were 686m (12.9%) and 297m (5.6%) respectively. The resulting drift is plotted in Figure 3.21.

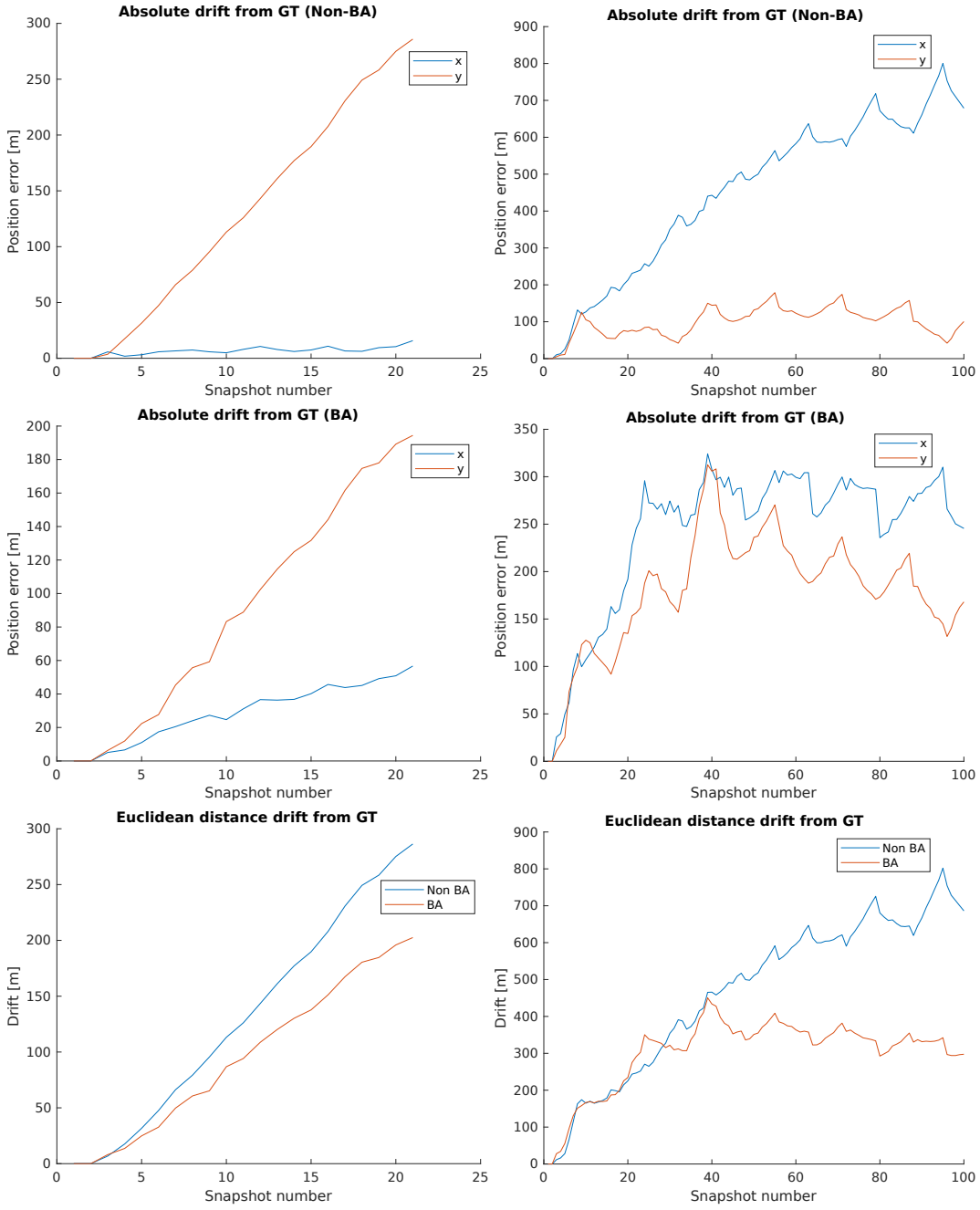


FIGURE 3.21: The drift values resulting from the estimated UAV paths. Left column: straight path dataset. Right column: zig-zag path dataset. First line: resulting drift in both axes without using BA. Second line: with BA. Third line: combined Euclidean drift from both BA and non-BA.

The proposed solutions are compared with each other in Figure 3.22. As can be seen, on the straight path dataset, the most accurate solution is the P5P with BA, due to the employment of global error minimisation. On the

zig-zag dataset the image stitching solution (homographies) performs better on average than P5P with BA, and scores a slightly better final drift result.

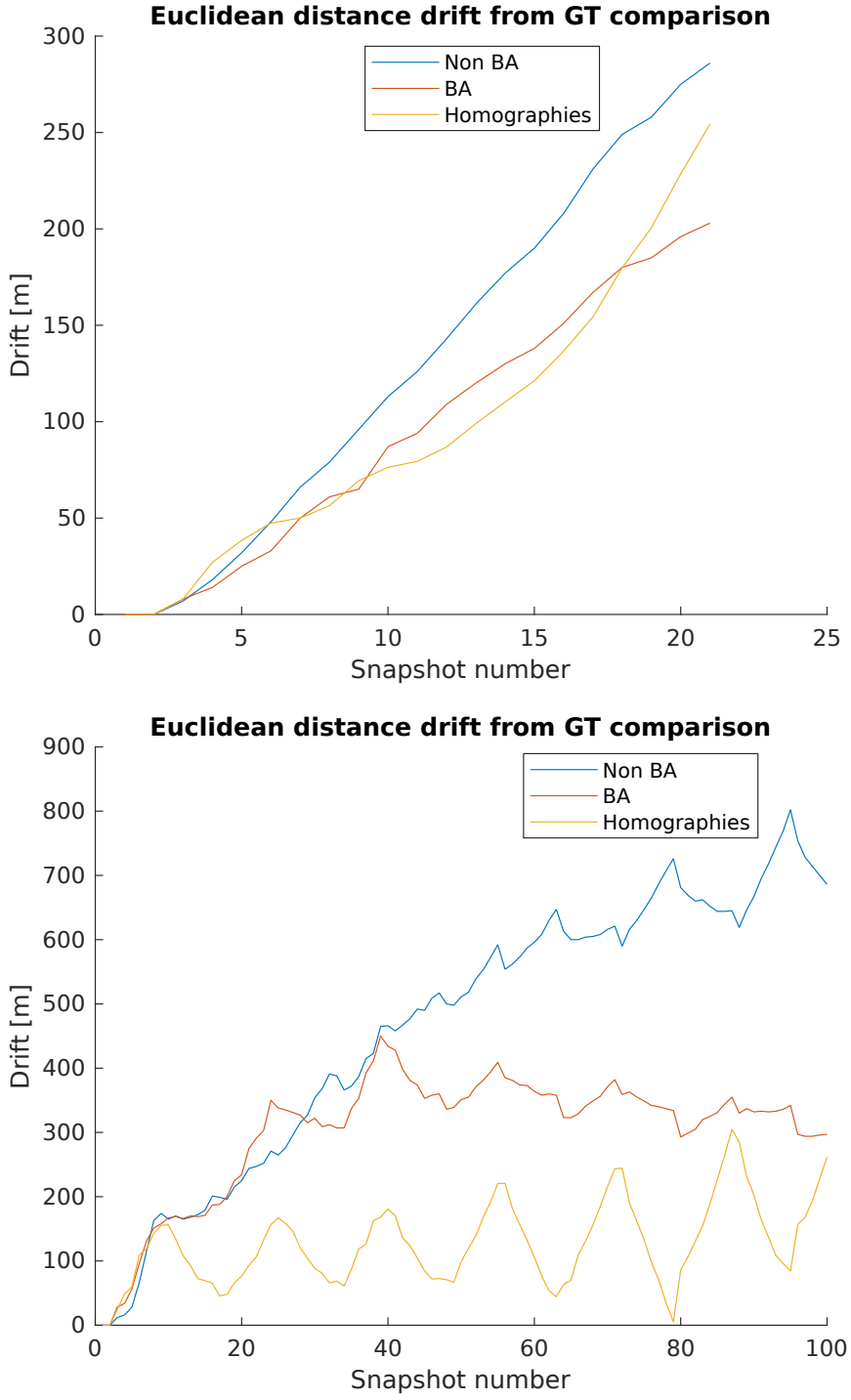


FIGURE 3.22: Drift comparison between using image stitching algorithm (homographies) and Perspective-from-n-Points (epipolar geometry) algorithm. Top: straight path dataset. Bottom: zig-zag path dataset.

To make sense of these results they must be compared to other works.

However, a direct comparison is not possible because the majority of other algorithms rely on correspondences between 3 frames for performing triangulation and scale estimation. In addition, the amount of frames used by other works is considerably larger. However, an extrapolation of results is possible as is done in Table 3.4.

For example, using a video camera mounted on a vehicle, Nister [43] used a combination of P5P for motion estimation, P3P for triangulation and Bundle Adjustment for optimisation. On a dataset of 2944 frames over a 365m long path, the final drift error was 1.63%. In another wheeled vehicle [83] which used P1P without BA, 100m drift was reported across a 3km path, which is equivalent to 3.33%. When VO was applied on UAVs, it achieved less accurate results than vehicular robots. For example, one research [168] used two datasets: the length of the first path was 140m, and the second path was 110m long. The UAV had 6DoF and the navigation system used a Kalman Filter to fuse visual and inertial sensor data. On the first flight it resulted in a 2D drift up to 30m. On the second flight it scored a drift up to 7m. The amount of drift was equivalent to 21.4% on the first dataset, and the second was equivalent to 6.3%.

| Algorithm | Type (Aerial/ Wheeled) | Navigated Distance | FPS | Drift |
|---|---|---|---|---|
| Ours (1) | Aerial | 1080m | 0.2 | 18.8% |
| Ours (2) | Aerial | 5300m | 0.2 | 4.9% |
| Nister [43] | Wheeled | 365m | 5-10 | 1.63% |
| Scaramuzza [83] | Wheeled | 3000m | 2.5-15 | 3.33% |
| Andert [168] | Aerial | 110m | 10-30 | 6.3% |

TABLE 3.4: Comparison between the results of the proposed algorithms and other VO methods (using the best results and the final drift values).

It is difficult to compare the amounts of drift achieved in this thesis to ones in other works. This qualitative difference in performance between aerial and vehicle VO systems can be attributed to:

1. Wheeled vehicles have more stability than aerial vehicles.
2. A higher frame rate means more frames being available for processing, which allows motion estimation and BA algorithms to have more data for processing, which means more accuracy.
3. Drift errors accumulate variably with time.

#### 3.4.3.3 Homing Vector

It is now possible to estimate the homing vector which is calculated by multiplying pose matrices by each other. This results in a total pose matrix. Each pose matrix is composed of rotation $R_{3\times3}$ and translation $T_{3\times1}$, and it has the form:

$$Pose\ state = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is equivalent to summing up all of the translation vectors (according to the orthogonal matrix property described by Equation 3.8). An example of an estimated homing vector can be seen in Figure 3.23, applied on the P5P with BA results. The target was missed by an angle of 10.3° which resulted in a Euclidean drift distance of 203m on the first straight path datset. On the second dataset, the angular drift was 10.2° resulting in a Euclidean drift distance of 298m.

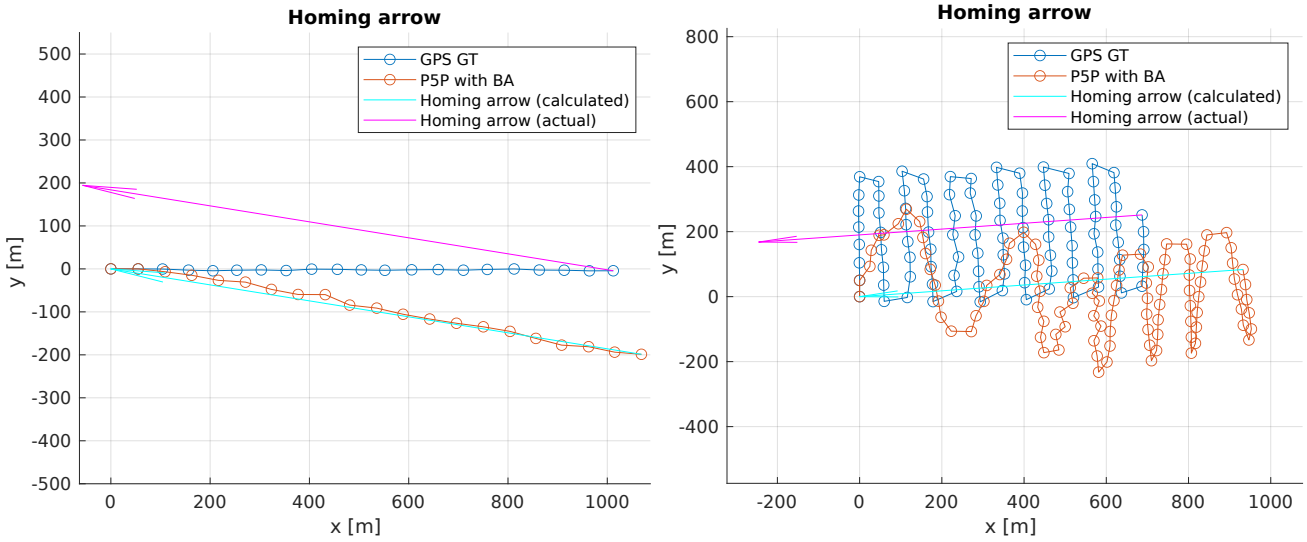


FIGURE 3.23: The estimated homing vectors calculated from the estimated UAV path using P5P with BA. The light blue path resembles the estimated homing orientation and length which should be taken. The purple path is the actual direction which the UAV will take. Due to drift, the homing is not very accurate.

## 3.5 Conclusion

In this chapter, two different visual localisation algorithms for homing were proposed. The algorithms are inspired by insects [136, 140]. The first one relies on image matching employing homographies, and the second one relies on epipolar geometry, estimating the camera pose using Perspective-From-n-Points algorithm. But because of resulting drift, these algorithms cannot be used to plan the homing path for fixed-wing UAV on long journeys as shown in the previous section. Therefore new solutions need to be investigated. The next chapter is going to investigate the possibility of using existing external cues which could aid in the task of homing, and localise the UAV more accurately, such as Google Earth.

# Chapter 4

# Global Localisation Using Sparse Image Deep Segmentation

It was shown in the previous chapter how conventional VO methods cause a serious amount of drift. This drift accumulates to the point that after a few kilometers, it becomes impossible to find the absolute location of the UAV. This makes the task of UAV homing or taking any shortcuts back home out of reach. Therefore, a natural question would arise at this point: Is it possible to use any existing cues that would help in the localisation of the UAV?

The current chapter answers this question by suggesting to use an existing aerial map for the absolute localisation of a UAV. Today, there are many public photographic datasets of the surface of the earth which can be invested in solving this problem. One such resource of information is Google Earth.

## 4.1 Introduction

Many risks are associated with relying on Global Navigation Satellite Systems (GNSS) for localisation which is traditionally used to localise fixed-wing unmanned aerial vehicles (UAVs). For example, as it was argued in Chapter 1.1, GNSS signal is succeptible to considerable malicious jamming and spoofing risks, other than well known signal unreliability when flying close to hills or large man made structures. Even if the chances of such failures were rare, their impact would remain massive. These issues put more value on having a redundant navigation mechanism that is independent and does not rely on external aid such as GNSS.

Visual Odometry (VO) [44, 45] is traditionally used to track the path of a robot such as a UAV. But the resulting location estimate of VO inevitably drifts due to inaccuracies and noise in the measurements. This was shown in Chapter 3. Drift can be corrected by the detection of loops, the so-called loop-closure [52, 57]. Combining VO and loop closure is called Simultaneous Localisation And Mapping (SLAM) [51]. But sometimes loop closures are not always possible, for example when the robot, especially a fixed-wing UAV, takes a long path and semi-circular turns. In this case, the localisation problem becomes more feasible to solve when some information is known about the environment, such as, a ground-truth map [187], or artificial landmarks [188, 189].

In certain applications when flying a fixed-wing UAV at high altitudes, a raw map of the environment is already available but it is not useful without further processing, for example, due to differences in the states of the environment. Google Earth is a rich source of this information, which can be invested in the localisation of fixed-wing UAVs given that they fly at high altitudes, much higher than rotor-wing UAVs. This is what makes an aerial map especially accepssible to fixed-wing UAVs. Being able to localise the UAV on a map, solves the problem of *homing*.

Many available visual UAV solutions take advantage of Google Earth for GNSS denied localisation. These solutions range from relying on object segmentation [190, 191], edge matching [50], feature matching [192, 193, 194], deep optic flow [7] and dense registration [195, 196]. Some methods use a precalculated lookup table (LUT) which resembles a map as in [197, 198]. Despite their success, challenges remain unaddressed in many of these systems, such as, slow performance [7], relying too much on the similarity between stored map and captured UAV snapshots [50], requiring a ground truth map [198] or lack of objective results [192, 196].

The following scenario is considered: a consumer grade fixed-wing UAV with onboard processing and a top-down RGB camera, which has gone beyond the visual line of sight but has access to a stored satellite map and a series of top-down RGB images captured in real time. For this a semantic segmentation based image-to-map matching localisation algorithm is designed. The map is segmented using a dedicated end-to-end Convolutional Neural Network (CNN). The UAV snapshots are segmented in real time using the *same* CNN, and the histogram of the current snapshot is then localised on the map. The proposed system architecture is depicted in Figure 4.1. See Fig. 4.2 for an outline of the particle filter localisation procedure, the whole algorithm is explained in detail in section 4.3.

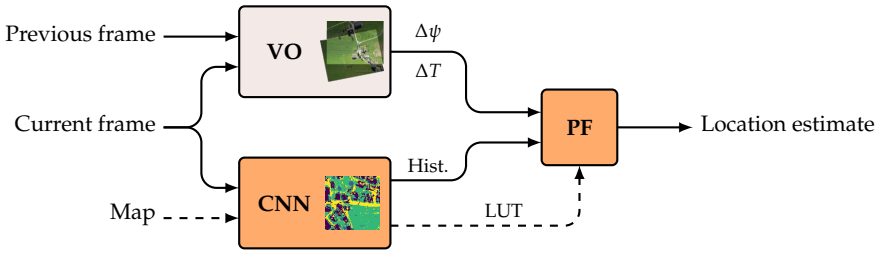


FIGURE 4.1: Proposed system architecture. The motion estimation (VO) component is used from the previous chapter. Two new components are added: The CNN, and the particle filter (PF). The CNN segments the map only once (dashed line) which is converted to a Look Up Table (LUT). It also segments each snapshot which is converted into a histogram of 4 values. Both the LUT and the histogram are used by the PF for localisation.

Semantic segmentation is necessary for understanding the composition and contents of the photo. The main segmentation challenge is the lack of training data belonging to the navigated region. Therefore the proposed

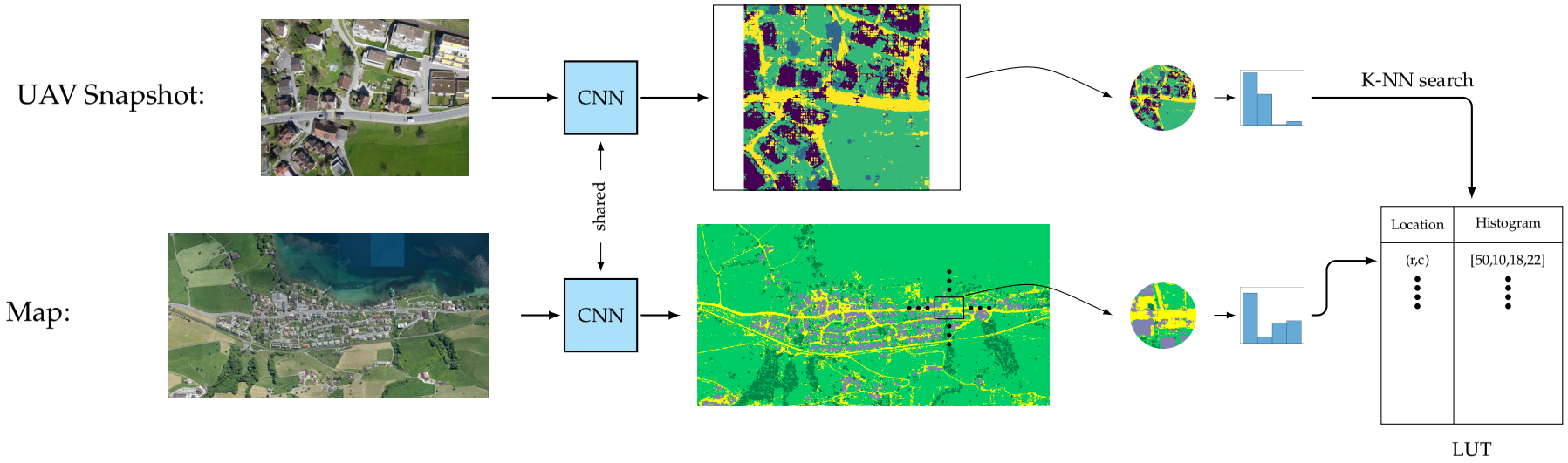


FIGURE 4.2: UAV localisation procedure. The UAV snapshots are segmented in real time but the map is segmented before flight. Segmentation is done using the same CNN. The map lookup table (LUT) is then calculated. A particle filter queries the LUT using K-Nearest Neighbours, preserving only the particles that match the current UAV snapshot histogram. Visual odometry is then done to initiate the next filtering stage (see section 4.3 for details). Histograms are saved in the LUT as a list of 4 percentages that sum to 100 (grass, houses, trees, roads).

CNN relies on a recent progress in aerial segmentation. It takes advantage of mutual features [199] between urban satellite maps and top-down aerial photos, given that they, A) share the same basic categories (roads, buildings, trees and grass), and B) the relationships between these components are similar (for example, roads are narrow, long and connected with each other). This network is trained on the well known ISPRS Potsdam dataset [3] (see Fig. 4.3), and it is not trained on any data derived from the UAV flight testing environment.

FIGURE 4.3: Top row: Sample snapshots captured by the UAV [2] used for localisation. Bottom row: Sample photo from the CNN training dataset (ISPRS Potsdam [3]). The ground truth for both is composed of four categories: houses, roads, trees and grass

The main contributions of this chapter are declared as follows:

1. The proposal of a localisation system which can localise a UAV either independently or act as a backup redundant system to aid GNSS.

2. It is demonstrated how real-time UAV localisation is possible given a CNN with a modest segmentation capacity. Despite the importance of having accurate segmentation, the focus of the contribution is *not* on accuracy.

3. The CNN is trained on a single aerial dataset, but it has the capacity to segment *any* aerial urban map, up to an accuracy percentage which is enough to enable localisation. This enables the UAV to localise itself in any urban setting.

The proposed deep learning segmentation based localisation system achieves localisation performance close to recent state-of-the-art methods [7] while

having the additional advantage of global localisation and real time performance. It is demonstrated that given an aerial map, it would be possible for a UAV to localise itself on a map in real-time by categorising different components of a top-down aerial image, e.g., houses, trees, grass and roads. This makes it possible to plan a return path to its origin. It also enables safe landing, and has the potential to amend UK government fixed-wing UAV safety regulations [8], by enabling safe flight beyond Visual Line Of Sight (VLOS).

The rest of the chapter is organised as follows: In section 4.2 related work is discussed, in section 4.3 the proposed algorithm is described, in section 4.4 the performance of the proposed algorithm is evaluated, and finally conclusions are outlined in section 4.5.

## 4.2 State Of The Art

In this section GNSS denied UAV localisation methods are reviewed in the first half (Section 4.2.1), and since the proposed algorithm is based on image segmentation, latest aerial image segmentation networks are reviewed later in (Section 4.2.2).

### 4.2.1 UAV Localisation

Existing solutions relying solely on VO [44, 45] are enough to cover short distances of a few hundred meters [78], but not longer ones. Even with Visual SLAM solutions [60, 79, 70, 73], they are mostly used in closed or short range environments. No SLAM applications have been found on fixed-wing UAVs, perhaps due to their high flying speeds which make loop closures [57] less likely. However, many works have been proposed to compensate for this drift by calculating the absolute location by making use of an existing map, for example Google Earth maps. Such solutions are reviewed in following paragraphs.

A demonstration of the capacity of using top-down photos for navigation using satellite imagery comes from [159] where the authors propose a basic pixel-by-pixel comparison navigation algorithm which is inspired by insect navigation.

We begin with solutions that use a lookup table. In [198] a geo-referencing absolute localisation system is presented, that uses a hand-labelled Google Earth map composed of 3 labels: grass, asphalt and house. The UAV snapshots are cropped circularly which makes them rotation invariant for matching against the map, and the histogram for each segmented snapshot is calculated in real time. This histogram resembles the ratios of the 3 different categories in the image, and it is matched to a look-up table produced from the map using a particle filter. This solution relies on a given ground truth segmented map, which we consider a shortcoming that we aim to address. Other solutions that use a lookup table include [197], but no convincing results are provided.

Some other localisation systems use edge matching, such as, a sobel edge detector [200, 50]. A Multi Layer Perceptron was used instead as an edge detector and an automatic thresholding algorithm was adopted as an improvement in the latter. The main disadvantage of edge matching algorithms is variance to orientation and scale, both of which require additional sensors to estimate. In addition, illumination changes in the environment can severely affect the thresholding algorithm required for edge detection.

Dense registration localisation methods that depend on pixel intensity values such as Mutual Information (MI) [195, 196], and optic flow [7]. We believe this is the category of solutions that achieves the best state-of-the-art accuracy. MI is used in [195, 196] to estimate the similarity between the captured UAV snapshot and the map. In [195] the accuracy is improved by using a stereo camera to register photos to a 3D map, the lack of availability of which we consider a disadvantage. In other systems MI is shown to produce very little improvement over other much simpler methods such as Sum of Square Differences [196], perhaps because the dataset used is unchallenging. A deep optic flow system is employed in [7] where a Convolutional Neural Network (CNN) is used with a dedicated Lukas Kanade optic flow layer for image registration, followed by state optimisation using photometric Bundle Adjustment. This work is chosen as the best performing state-of-the-art solution against which the algorithm proposed in this chapter is compared.

Other localisation methods rely on feature matching using traditional corner detectors, such as, SIFT, SURF and BRIEF, extracted from the RGB photos. But due to temporal changes in the state of the environment (whether seasonal or man-made) between the stored map and the current environment state, such methods perform poorly, and current available solutions provide unconvincing results [192, 193, 194], one shortage is the lack of comparison with competing algorithms. Other methods focus on object segmentation such as houses [190] and roads [191, 201]; and invest them for localisation. Such methods assume the existence of roads and houses in each captured UAV snapshot, a condition that is not always guaranteed. And since matching relies on certain segments, useful information is possibly ignored.

### 4.2.2 Aerial Image Segmentation

Many deep learning solutions today have become state-of-the-art solutions in a wide range of fields [99]. Within the field of semantic segmentation, deep learning solutions are based on Convolutional Neural Networks (CNN). A sub-set of CNNs is called Fully Convolutional Networks (FCN) [111], which are at the heart of solving the problem of semantic segmentation.

One category of a segmentation network is the Encoder-decoder architecture. It is composed of a CNN stack that downsamples the input image into denser feature maps, followed by an upsampling stack that upsamples the feature maps into a segmented image with the same size as the input. An example of this kind of network is SegNet [202, 113], in which pool indices from the encoder are transferred to the decoder, and the decoder is followed by a

final softmax classification layer. Another category is U-Net [106], in which each step of the decoder is concatenated with feature maps taken from corresponding parallel locations on the encoder.

Many deep learning solutions have been developed particularly for the segmentation of aerial/satellite photos. Works in this field include road extraction [203], building detection [204, 205] and landcover classification (semantic labelling) [204, 206, 207, 208, 209, 210, 211, 212, 213]. The first work that dealt with the lack of training data derived from the navigated region is [214]. The training data used was completely different than the testing data (both belonging to different cities), and it confirmed the generalisation capacity of CNNs on aerial photos.

Another notable work is relation networks (RN). Original RN research aimed at learning relations between different objects using their textual description [215]. It was later expanded to cover convolutional networks [216]. Recently, this research was applied on aerial photos, where progress was done on a relation CNN [199] which can improve the capacity of an FCN to learn how to associate similar distant regions of an aerial photo. Two different modules are used, one that makes use of the contextual information; and another that takes advantage of colour. The modules accumulate information and automatically learn the relationships in between. The modules are designed as two extensions that can be added to any network, and both are made use of at the bottom of the CNN.

Other notable research on image segmentation modular networks is the use of Spatial Pyramids which started with [217]. In the beginning it was used for object detection and image classification by calculating feature description histograms of different image regions. It was then improved in Spatial Pyramid Pooling (SPP-net) [218] which became a module that can be added to the end of a CNN before a classification layer. Its function was to pool information on several global scales, which produced different descriptors that can be eventually classified. Spatial Pyramids were finally applied on convolutional networks to do semantic segmentation in Pyramid Scene Parsing Network (PSPNet) [219], where it was used as a modular layer that can be integrated into an FCN.

## 4.3 Proposed Method

In the first part of this section, the CNN implementation which is used for segmentation is described (Section 4.3.1). In the second part the localisation algorithm is outlined, which is the particle filtering algorithm used to localise the segmented snapshots on the segmented map (Section 4.3.2).

### 4.3.1 Segmentation Convolutional Neural Network Architecture

A lightweight CNN for segmenting both the UAV snapshots and the map is designed (refer to Fig. 4.4 for the CNN structure). The network is a fully

convolutional network [111] which has a U-Net [106] architecture, its encoder being a mobileNetV2 network [105] which is a lightweight network optimised for mobile devices. The encoder network is pretrained on ImageNet dataset [108], which is frozen for transfer learning. Each of the modules described below are added incrementally until the required segmentation accuracy threshold is matched (this test is described in section 4.4.3.1).

The encoder's final output is first passed through 3 bottleneck residual blocks, which are the basic building blocks of ResNet [103]. Each block has double the number of output feature maps of the previous block. This configuration was fine-tuned by testing randomly such that the overall network achieved the best segmentation results.

The output is then passed through two modules taken from state-of-the-art segmentation networks. The first module is a Pyramid Pooling Module, which is adapted from [219]. In this module 3 different pyramid scales are used to process the input and form 4 feature maps. The first feature map is the result of 1x1 global average pooling, the second and the third feature maps are 2x2 average pooling; and the last is 8x8. Each feature map is convoluted using a 1x1 convolution filter, and then upsampled. Eventually all 4 feature maps are concatenated with the original input, and passed to the following module.

The last module is the Relations Module [199]. The relevance of this module (adapted from [199]) is that it was designed for aerial photograph segmentation by focusing on relating distant image segments with each other. Relations Module is composed of two smaller modules: Spatial Relation Module (see Fig. 4.6), and Channel Relation Module (see Fig. 4.7). Both of these modules are connected in parallel, they receive the same input and the results are concatenated.

The output is finally taken through the decoder, where it is upsampled 4 times using transpose convolutions. After each upsampling stage, the output is concatenated with the encoder's *expand ReLU* layer outputs which are taken from blocks 1, 3, 6, 13 of MobileNetV2. A sample segmentation output of the network can be seen in Fig. 4.5. For more sample results see (Fig.4.16 and Fig.4.17).

#### 4.3.1.1 Significance of each Module:

Each module which was added to the network has its own significance and special properties for which it was chosen. These properties are described as such:

1. The residual blocks (Module 1 in Fig. 4.4): Taken from Res-Net [103] which is composed of a large number of consecutive layers of these blocks, each of which is designed to work well in deep layers. The significance of these blocks is that they contain a skip connection whichs enables achieving more depth than the deepest earlier networks such as VGG [102].

2. Pyramids (Module 2 in Fig. 4.4): Shown to be useful in connecting distant regions of the image to each other.

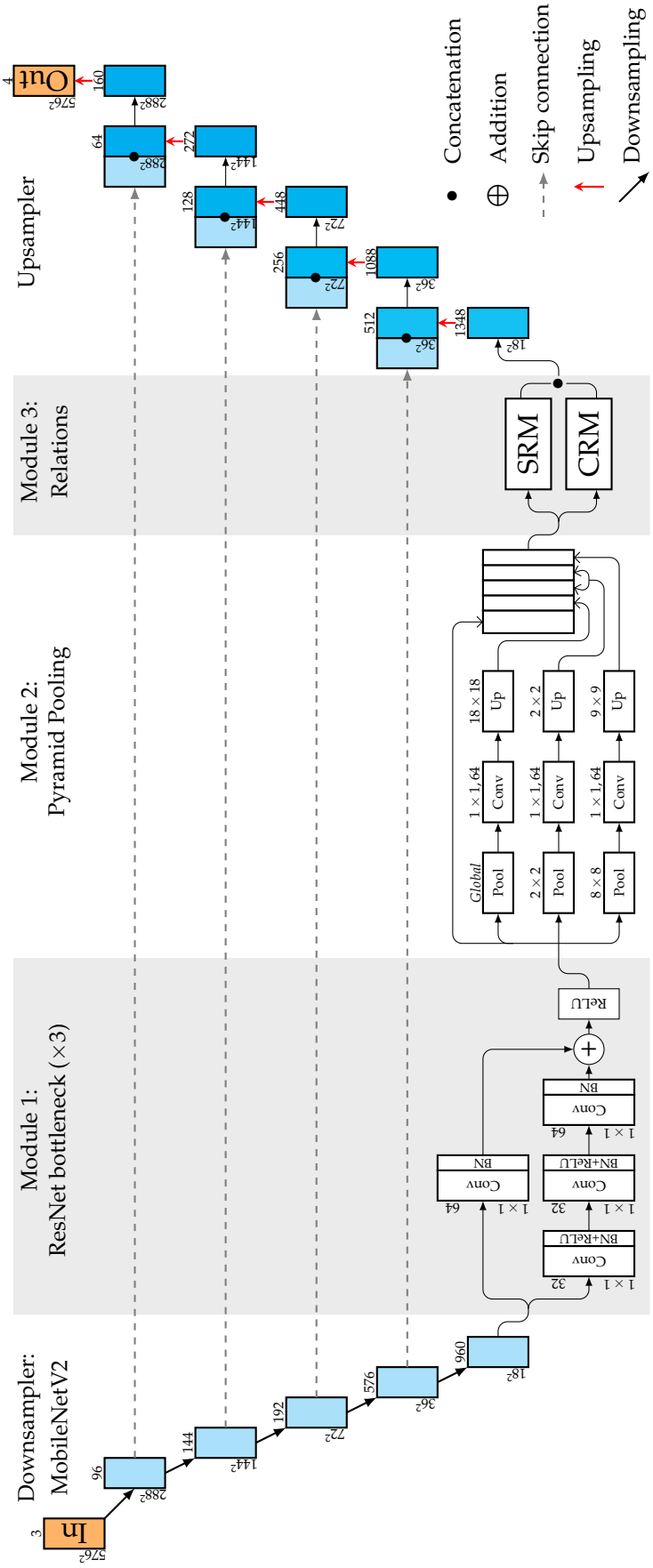


FIGURE 4.4: The proposed custom U-NET, with the 3 modular extensions: The ResNet bottleneck module (RNM), Pyramid Pooling Module (PPM) and Relations Module (RM). Numbers signify output feature map sizes, and kernel sizes. 1) RNM setup: 'ReLU' blocks are Rectified Linear Unit layers [4], 'BN' blocks are Batch Normalisation [5] and 'Conv' are convolution layers. This module is repeated 3 times, each time doubling the amount of output feature maps. 2) PPM setup: consists of pooling, convolution and upsampling layers. Results from all streams are concatenated at the end. 3) RM setup: composed of a Channel Relation Module (CRM) and a Spatial Relation Module (SRM) connected in parallel and concatenated.

3. Relation Modules (represented by Module 3 in Fig. 4.4): Spatial Relations module (Fig. 4.6) helps in capturing global spatial relations, and Channel Relation module (Fig. 4.7) helps in capturing contextual relations.

Notably each of these modules has a skip connection, so that the original signal is not modified but rather amplified.

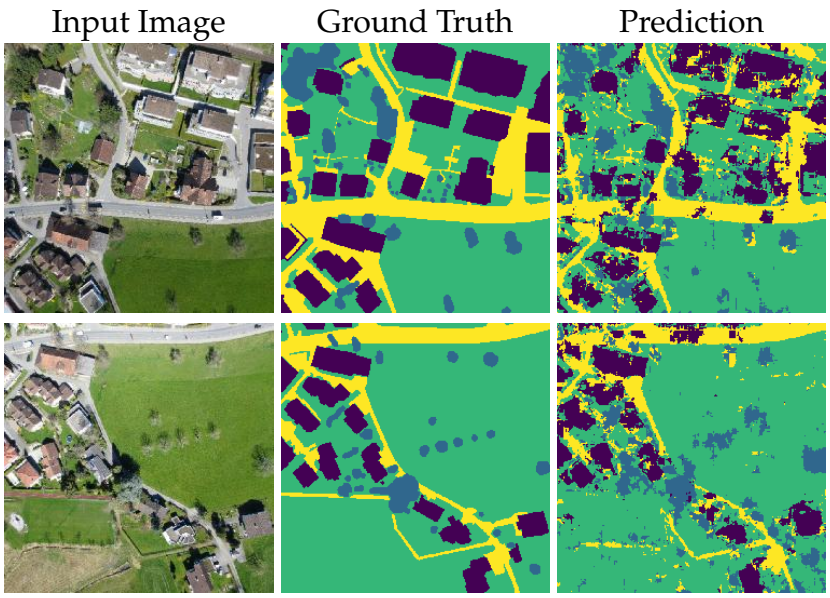


FIGURE 4.5: Sample CNN segmentation results on UAV snapshots

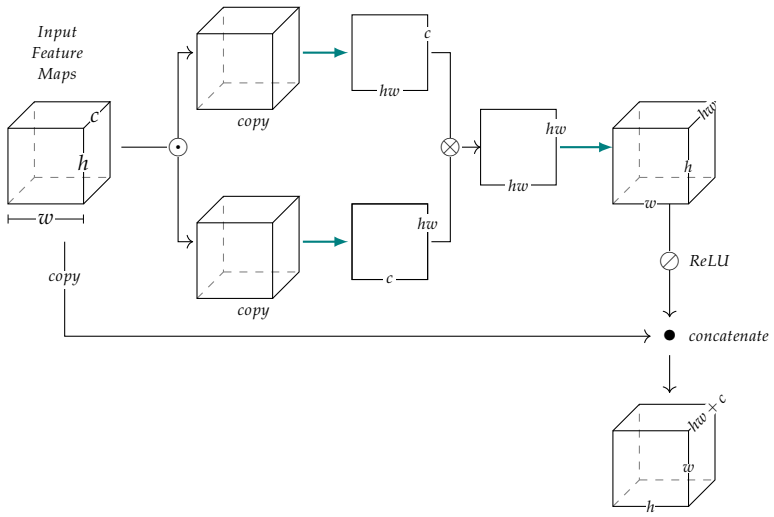


FIGURE 4.6: Spatial Relation Module (SRM). ➝ signifies reshape. ⊗ is matrix multiplication. ⊙ is 1×1 convolution. ⊘ is Rectified Linear Unit (ReLU). ● is concatenation.

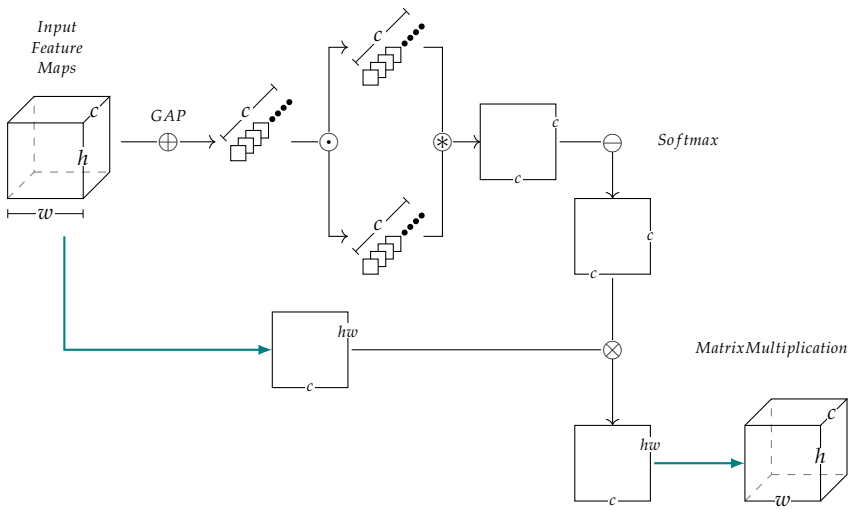


FIGURE 4.7: Channel Relation Module (CRM). $\oplus$ denotes Global Average Pooling. $\odot$ denotes 1×1 convolution. $\circledast$ denotes outer production. $\ominus$ denotes Softmax operation. $\otimes$ denotes matrix multiplication. $\twoheadrightarrow$ signifies reshape.

### 4.3.2 Global Particle Filtering

In this section a K-Nearest Neighbour based Particle Filtering algorithm is described. It will be used to localise a series of segmented top-down UAV snapshots on a segmented Google Earth map. Our goal is to track the path drawn by the snapshots on the map, therefore localise the most recent snapshot location on the map, which corresponds to the absolute location of the UAV. In general the inspiration of this algorithm is [198], similar logic is followed in this proposal but without relying on a ground truth map.

Our particle filtering (PF) framework, also known as Sequential Monte Carlo (SMC) [54, 220], is a nonlinear state estimation approach that approximates the posterior by a finite set of random samples. It is inspired by a template matching algorithm [221]. The input measurements to the PF are the angle and the distance between each two snapshots, both of which are derived from VO.

The map is segmented using the CNN described in Section 4.3.1 before flight and it is composed of 4 different categories: grass, houses, roads and trees. Each snapshot is then segmented in real time using the same CNN into the same categories. All input snapshots to the algorithm are resized to 800x600. The localisation steps are as follows:

1. Create the segmented map look-up table (LUT) $H_n = \{h_1, h_2, ..., h_n\}$ using a sliding window (see Fig. 4.2). Each entry $h_i$ in this table corresponds to the *histogram* of a circular region with radius $r$ centred at a unique location on the map. The radius $r$ is equal to $I/2$, where $I = 576$ pixels, the height of the input image to the CNN. A window stride equivalent to 5 meters in chosen in both column and row directions to reduce the size of LUT; this number will define the maximum localisation accuracy that can be achieved. Each histogram is a list of 4

percentage values that sum to 100 corresponding to 4 categories: grass, houses, roads and trees.

2. PF initialisation: Create a set of particles $P_c = \{p_1, p_2, ..., p_c\}$. The particle count is chosen to be $c = 10,000$. Randomly spread the particles $P_c$ globally across the whole map (see Fig. 4.8).

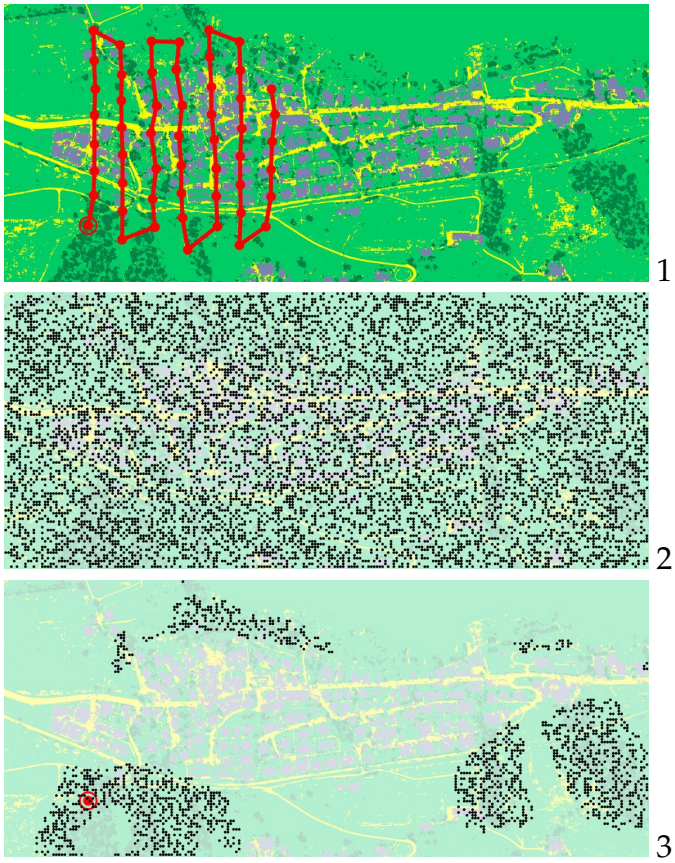


FIGURE 4.8: (1) Ground truth path as recorded by GPS stamps on photos (in red). (2) Global spreading of particles. (3) K-NN particle matches for the 1st snapshot.

3. Segment the first snapshot $S_t$ and take its histogram $S'_t$, then find its K-Nearest Neighbour [222] in the LUT: $K_t = \{Y : d(Y, H_i) < d(H_i, H_j), i \neq j\}$ where $Y$ is a small set of histograms closest to $S'_t$; and $d$ is the Euclidean distance function: $d(x, y) = \sqrt{\sum_{i=1}^{4}(x_i - y_i)^2}$.

4. Segment the second snapshot $S_{(t+1)}$ and take its histogram $S'_{(t+1)}$. Find $K_{(t+1)}$ then allocate an initial movement direction for each particle by finding the nearest neighbours to $K_t$. This is done by finding the nearest neighbours of 8 equally dispersed points around $K_t$ (see Fig. 4.9).

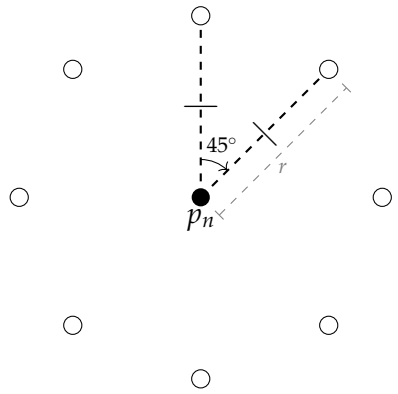


FIGURE 4.9: Particle $p_n$ with 8 possible movement directions of radius $r$, each is measured at 45° angle of the next.

5. Acquire the following snapshot $S_{(t+2)}$, then calculate the amount of yaw rotation $\psi$ with the previous snapshot $S_{(t+1)}$. Since the UAV movement is planar and the altitude is nearly constant, a planar scene is assumed and $\psi$ is estimated by finding the Rigid (Euclidean) transformation matrix between two snapshots: First, the locations of the correspondences between the two snapshots are extracted using SURF [180]. They are centred and multiplied by each other: $M = Cor_n^T \times Cor_{(n-1)}$, where $Cor_n$ is the correspondences locations list of snapshot $n$. Then the Singular Value Decomposition (SVD) of M is calculated: $M = USV^T$, where $U$, $S$, $V$ are the factorised matrices resulting from SVD. The Euclidean rotation matrix can be then acquired using the equation:

$$R = V \times \left|V \times U^T\right| \times U^T, \tag{4.1}$$

where $R$ is a counter-clockwise rotation matrix which takes the shape:

$$R_{2\times 2} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}. \tag{4.2}$$

After calculating $R$, the euclidean distance $d$ between the two snapshots can be calculated from the difference of their corresponding centres.

6. Scale the snapshot so that its height fits the input height $I$ required by the CNN. Crop the snapshot to a circle centred at the centre with a radius $r$ (following [198]). Segment the snapshot using our CNN, then calculate its histogram $S'_{(t+2)}$ (see Fig. 4.10).

FIGURE 4.10: Cropped snapshot $S$ with histogram: $S' = [55.3, 0.1, 2.3, 42.2]$ resembling percentages of: grass, houses, roads and trees.

7. Move all particles by an angle $\psi$, and a distance $d$ relative to their previous locations (see Fig. 4.11). The amount of movement of a particle $p_n = (x_n, y_n)$ is calculated using:

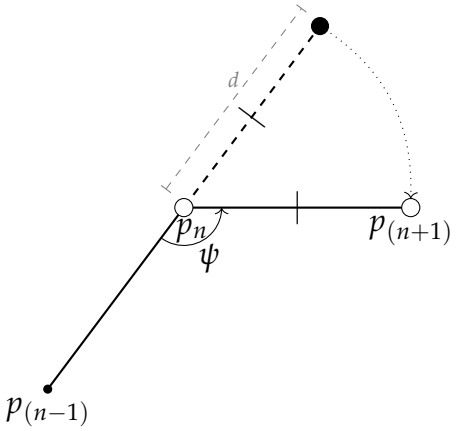


FIGURE 4.11: Particle $p_{(n+1)}$ after being moved from its previous location $p_n$ relative to $p_{(n-1)}$ by angle $\psi$ and distance $d$.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \begin{bmatrix} x'' \\ y'' \end{bmatrix}, \tag{4.3}$$

Where $R$ is substituted from eq. 4.2; and $(x'', y'')$ is the vector from the previous particle location to the current one $\overrightarrow{p_{(n-1)}p_n}$:

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} p_n.x - p_{(n-1)}.x \\ p_n.y - p_{(n-1)}.y \end{bmatrix}, \tag{4.4}$$

Finally the new particle location $p_{(n+1)}$ is calculated:

$$p_{(n+1)} = (p_n.x + x', p_n.y + y'). \tag{4.5}$$

8. After moving all particles, look up their equivalent histogram from the LUT. Then calculate the particle weights based on their KNN distance to the current snapshot histogram $S'_t$. Normalise all weights so that they sum to 1:

$$K'_i = \frac{K_i}{\sum K_i}. \tag{4.6}$$

9. Resampling: Sort the particle weights in a list starting with the small weights (weights reflect the KNN distance to the current measurement i.e. snapshot histogram, a smaller weight means a closer match). Then calculate the *cumulative sum* $C$ of all particle weights. Randomly sample $n$ particles in the first half of $C$. This will increase the probability of selecting particles that best agree with the measured snapshot histogram. Because the weights are sorted, cumulative sum is denser in the first half than in the second half (see Fig. 4.12).


FIGURE 4.12: The cumulative sum of particle weights, notice that it is denser at one edge than the other. Each pin represents an entry, entries at the dense edge are closer match to the measurement (snapshot histogram).

10. Calculate the PF estimate of the current snapshot location by taking the mean of all particle locations. Then repeat from step (5).

## 4.4 Performance Evaluation

The experimental setup is described first in Section 4.4.1, then the performance of both segmentation and localisation is evaluated in Sections 4.4.2 and 4.4.3 consecutively.

### 4.4.1 Experimental Setup

#### 4.4.1.1 Datasets

The proposed CNN was trained on Potsdam satellite semantic labelling image database authored by International Society for Photogrammetry and Remote Sensing (ISPRS) [3]. The dataset contains 38 images, each image has a resolution of 6000x6000 pixels and a Ground Sampling Distance (GSD) of 5cm. To train the network, the RGB orthographic versions of images are used (see bottom row in Fig. 4.3).

Seven photos are used for validation (photo IDs are: 2_10, 2_11, 2_12, 2_13, 2_14, 3_10 and 3_11), and the rest are used for training excluding photo

ID 4_12 (a total of 30 images). The corresponding ground truth images are given and they are labeled by the authors using 6 categories: building, road, tree, low vegetation (grass), artificial ground and cars. The CNN was trained using 4 labels, after combining cars category with road; and artificial ground with grass.

The main influential factor in training the CNN is data augmentation. Potsdam RGB images are resized from 6000x6000 to 1200x1200 resolution to match the scale of snapshots captured by the UAV. Then the augmented datasets are created by cropping each image into 5 images (the 4 corners and the center), then applying different rotation and flipping operations on the resulting images. Finally each photo is blurred using a gaussian (normal distribution) 3x3 kernel with $\sigma = 1$.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.7}$$

Where $\mu$ is the mean (controls the centre of the curve), and $\sigma$ is the standard deviation. $\sigma^2$ is the variance, which controls the width of the guassian curve. If the function is plotted, it produces a bell shaped curve, this is why it is called the bell curve.

The main UAV snapshots dataset used to evaluate the system is a publicly available Sensefly Small Village dataset, Merlischachen [2]. The dataset was captured in April 2013, and contains 297 high resolution (4608x3456) top-down snapshots taken by a fixed-wing UAV (see top row in Fig. 4.3). The UAV flight height is 162 meters relative to the ground, and the absolute height ranges from 609-616m above sea level. Each snapshot has a reported GSD of 5.32cm. 53 snapshots starting from photo ID 0928 are selected for testing our algorithm, each having an overlap between 49-72% with the next. The average distance between consecutive snapshot centres is between 50-60m, and the average time gap is 5 seconds. The total navigated distance is 2.85km with a flight time of 4 minutes and 39 seconds. Another dataset (Willisau Swiss Gravel Quarry [6]) captured in April 2013, was also used to compare our system with [7].

The maps were acquired from Google Earth. A map of the navigated village (Merlischachen) was acquired at an altitude that corresponds to the same scale of UAV snapshots. Two versions of this map were tested, one acquired in June 2015, and another in May 2012 (see 2nd & 3rd columns in Fig. 4.13). Both vary in the amount of changes from the state of the environment at flight time (which was in April 2013, see 1st column in Fig. 4.13). The segmented ground truth for 2013 and 2015 maps were manually created and used later for evaluation. Another map for Willisau [6] was also acquired on November 2014 (see Fig. 4.14).

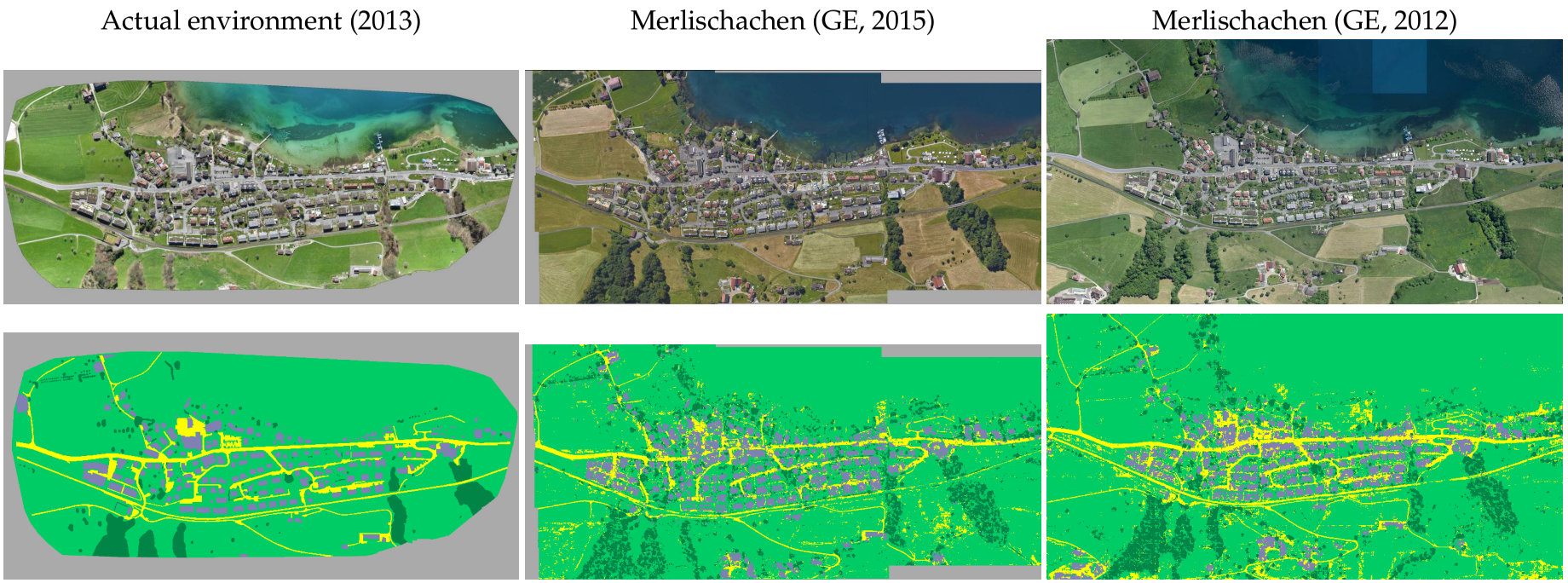


FIGURE 4.13: **1st Column:** The actual Merlischachen village environment as it looks at the time of the flight (April 2013) of the first dataset [2], along with its manually labeled ground truth (below). **2nd & 3rd Columns:** Merlischachen Google Earth map (acquired on June 2015, and May 2012 respectively) with the corresponding CNN segmented versions.

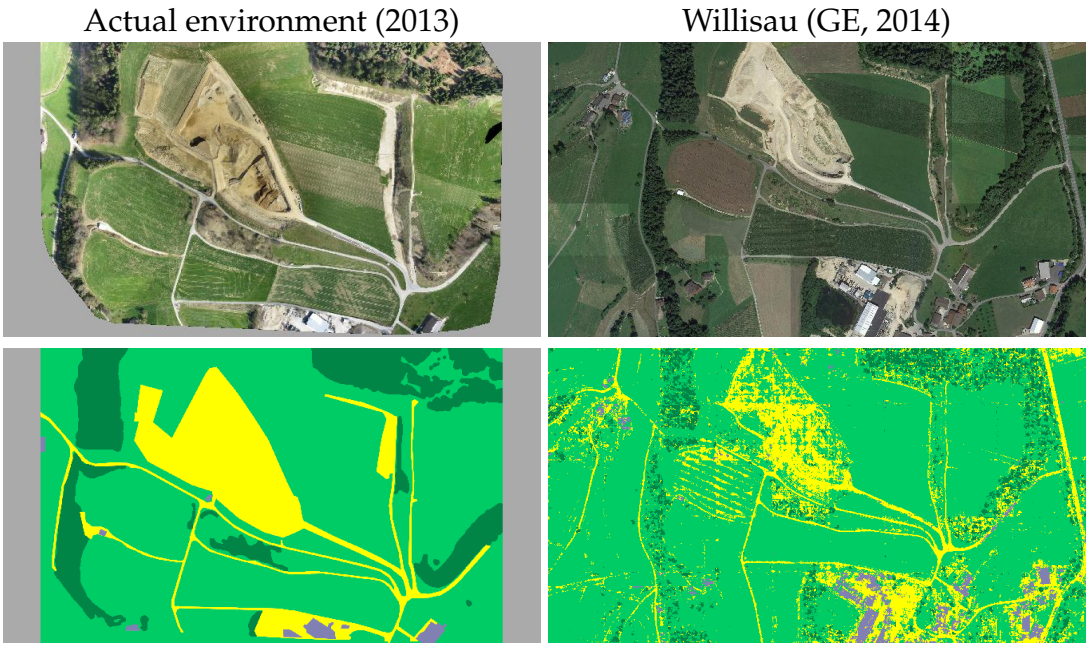


FIGURE 4.14: The map used in the second dataset [6], acquired from GE in November 2014. Along with the corresponding location at the time of the flight (Dec. 2013).

A few of the photos (whether from the snapshots or the map), contain a different category belonging to sea. But to our advantage there was no confusion about classifying this category as the network learned to classify it consistently as 'grass' across all photos.

#### 4.4.1.2 CNN Implementation

The final network contained 13m weight parameters. It was implemented using Tensorflow and trained using an Nvidia Tesla K80. The encoder was frozen so that its weights did not change. The Adam optimiser was used and trained until the network could no longer improve on the validation set. The network input image is a square image with height $I = 576$ pixels, and the output is a segmented image of 4 labels with the same resolution. Because transfer learning and a lightweight network were used, training the CNN took around 8 hours, using a batch size of 16, and 20 epochs. The corresponding training and validation loss can be seen in left of Fig.4.15, and the resulting accuracy on the right of the same figure.

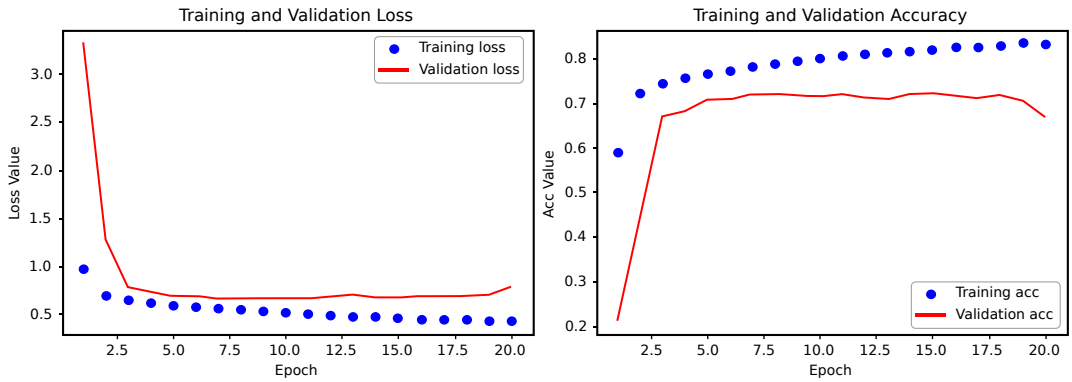


FIGURE 4.15: Loss (left) and accuracy (right) CNN training curves.

## 4.4.2 Segmentation Performance

### 4.4.2.1 Evaluation Metric

To evaluate the segmentation performance Merlischachen dataset was used along with the corresponding 2015 Google Earth map (2nd column in Fig. 4.13). The segmentation grou nd truth for both is not available so one third of Merlischachen testing snapshots (17 out of 53) were manually hand labeled, as well as all of the 2015 Google Earth map.

The segmentation confusion matrices are calculated. The amounts of correctly classified pixels (true positives) and the incorrectly classified pixels (false negatives) are shown for each label. Bottom Table 4.1 shows the confusion matrix of Merlischachen snapshots, and Top Table 4.1 shows the confusion matrix of Merlischachen Google Earth map.

| → Reference<br>↓ Prediction | Houses | Trees | Grass | Roads |
|---|---|---|---|---|
| Houses | **51.24** | 5.65 | 20.78 | 22.33 |
| Trees | 0.27 | **62.85** | 36.07 | 0.81 |
| Grass | 2.23 | 10.04 | **83.00** | 4.73 |
| Roads | 3.38 | 2.56 | 33.70 | **60.35** |

| → Reference<br>↓ Prediction | Houses | Trees | Grass | Roads |
|---|---|---|---|---|
| Houses | **69.13** | 1.44 | 11.53 | 17.89 |
| Trees | 1.32 | **51.52** | 45.60 | 1.56 |
| Grass | 3.33 | 6.82 | **85.84** | 4.00 |
| Roads | 7.71 | 6.33 | 22.89 | **63.07** |

TABLE 4.1: Segmentation accuracy confusion matrices for 2015 Merlischachen Map (top), and Merlischachen snapshots [2] (bottom). Higher values (percentages) on the diagonal line mean higher segmentation accuracy for each individual category (true positives).

Even though that the incorrectly labelled pixels (false negatives) can amount up to 45% on a single category; or up to 22% on average; but the majority of these incorrectly labelled pixels contribute towards the Overall Percentage Accuracy (OPA) metric, which is the measure which was used to evaluate matching the segmented snapshots to the map.

The Overall Percentage Accuracy (OPA) is defined as the overall pixel count of each predicted label (whether true positives or false negatives) compared to the ground truth counts. To calculate the OPA, Algorithm 3 is followed:

**Algorithm 3** Calculation of Overall Percentage Accuracy (OPA)

**for** snapshot $s \leftarrow 1, n$ **do**
  **for all** label $i \in \{houses, trees, grass, road\}$ **do**
    $A \leftarrow$ percentage of $i$ in GT
    $B \leftarrow$ percentage of $i$ in prediction
    $Acc_i \leftarrow A - B$
    **if** $Acc_i < 0$ **then**
      $Acc_i \leftarrow 0$
  $OPA_s = 100 - \sum Acc_i$
$OPA = (\sum_{s=1}^{n} OPA_s)/n$

#### 4.4.2.2 Results and Discussion

The advantage of our navigation algorithm is that it is not necessary to have a state-of-the-art segmentation performance, whether with regard to true positives or accurate boundary segmentation, as misclassified pixels (false negatives) can count towards the OPA metric.

What matters for us is to have ratios of labels in each snapshot as close as possible to the ground truth ratios. This information is represented using the Overall Percentage Accuracy (OPA) metric. This means that the misclassified pixels can count towards the overall ratios.

In table 4.2, the total percentages of true positives and false negatives can be seen, along with the corresponding Overall Percentage Accuracy (OPA).

| → Reference<br>↓ Measure | Snapshots | Map |
|---|---|---|
| True Positives (%) | 80.40 | 77.32 |
| False Negatives (%) | 19.60 | 22.68 |
| Overall Percentage Accuracy (OPA) (%) | **95.46** | **90.48** |
| CNN Inference Time (s) | 0.55 | 0.41 |

TABLE 4.2: Segmentation performance on Merlischachen snapshots and map. Inference time is per single processed photo.

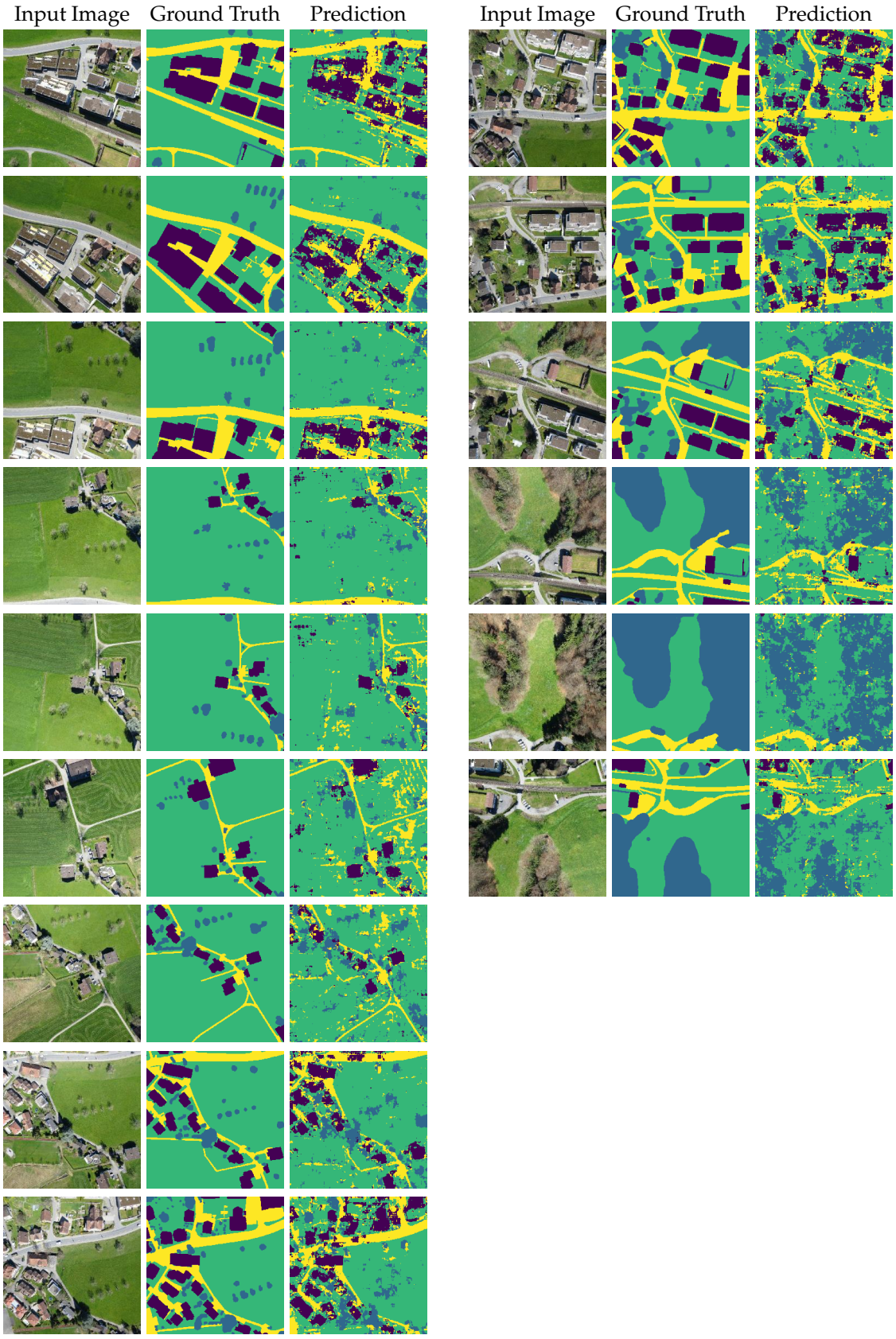


FIGURE 4.16: CNN segmentation results on first 15 snapshots from Merlischachen dataset.

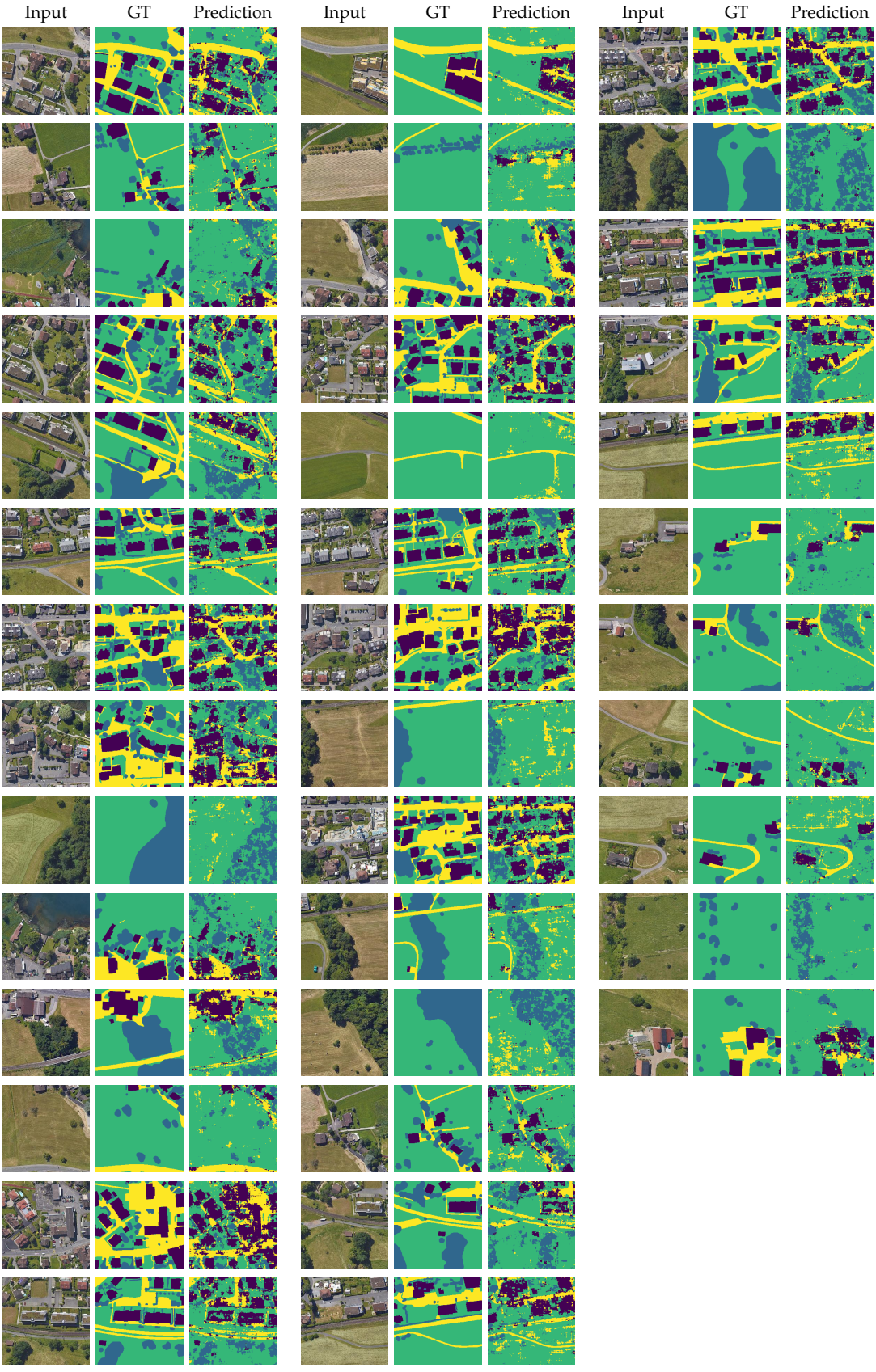


FIGURE 4.17: Sample CNN segmentation results on GE map.

### 4.4.3 Localisation Performance

#### 4.4.3.1 Evaluation Metric

An accuracy test was conducted to find the lowest possible Overall Percentage Accuracy (OPA) on both map and snapshots, which is required for successful localisation (i.e. for the PF to maintain tracking and to eventually converge). This test would guide us in designing the CNN that would enable this localisation in real time.

The starting point is the segmented ground truth for both Merlischachen snapshots and map, then noise is incrementally added. First the algorithm is tested after adding noise to the snapshots alone, then to the map alone, then to both snapshots and map (see Table 4.3).

In the case of the map, uniform noise is distributed across the map, and the look-up table $H_n$ is recalculated. In the case of the snapshots, either static or random noise value is either added or subtracted from the histogram.

| Noise tolerance | **-** **static noise** | **+** **static noise** | **+/-** **random noise *** |
|---|---|---|---|
| **Snapshots** | up to 30% | up to 30% | up to 30% |
| **Map** | up to 25% | up to 35% | up to 30% |
| **Both** | - ** | - ** | up to 15% |

TABLE 4.3: Noise tolerance rates for both the snapshots and the map. * The random noise has an upper limit and a lower limit of the percentage specified. ** Repeating the same static operation on both map and snapshots has no effect.

The process started with a bare U-Net, the modules described in section 4.3.1 are incrementally added, and the network is fine tuned until an OPA accuracy higher than 85% is achieved, the required minimum according to the conducted test. Both OPA segmentation accuracies in Table 4.2 match this requirement. It is important to note that a bare U-Net without the additional modules achieves considerably poorer localisation results.

#### 4.4.3.2 Results and Discussion

The inference time of the CNN is an average of 0.55 seconds for UAV snapshots, and 0.42 seconds for map photos tested on an Intel Core i3-6100T CPU. The maximum time required to calculate the Rigid transform after extracting SURF features and matching them (as mentioned in section 4.3.2) is 0.22 seconds. This means that more than enough time is available to do the processing in real time (a maximum of 0.77 seconds per snapshot), as the capture time difference between each two snapshots is between 4-5 seconds. This is much better than the method presented in [7], whose optimisation alone takes an average of 6.8 seconds per snapshot on the same CPU.

In Fig. 4.18 the proposed global localisation algorithm is evaluated using 53 snapshots from the Sensefly Merlischachen dataset [2] on two Google Earth maps. Using the first map (captured on June 2015), it was possible to

estimate the location of the UAV with an average drift amount of 35m since convergence on snapshot 16. This is within the visual range of a single snapshot which has a radius of around 80m. The second map was captured on May 2012, where the PF converged on snapshot 11, and with an average drift of 39m. The average drift for VO was 66m. The total area of each map is $600m^2$, and the total navigated distance is 2.85km. A video recording of the system's performance on the first dataset (GE, 2015) is available online [1]. It can also be seen in Fig.4.21.

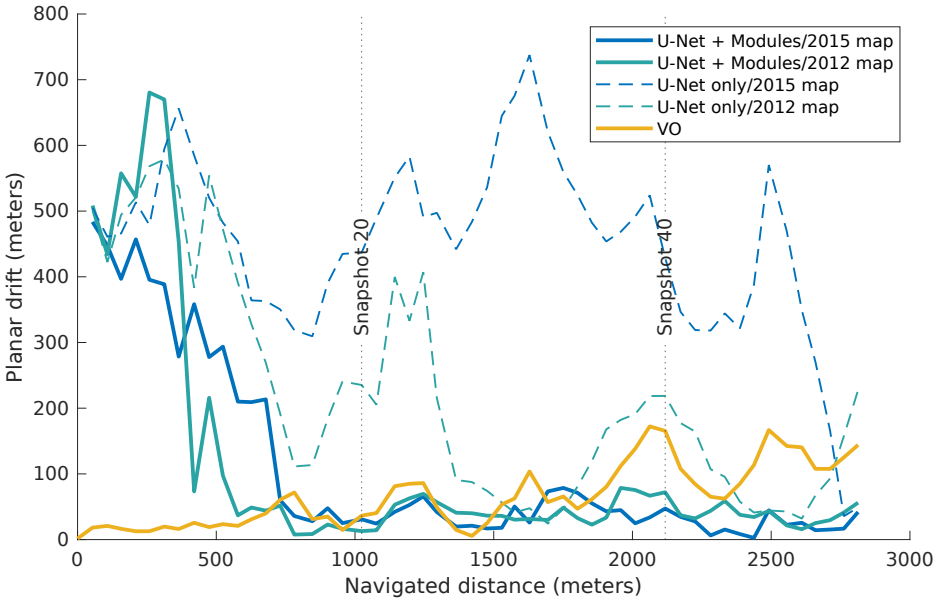


FIGURE 4.18: Our global absolute localisation performance compared to Visual Odometry relative localisation tested on 53 snapshots from Merlischachen dataset. On a Google Earth map captured in May 2012, the PF converges on snapshot 11, and with an average drift of 35m. On a different map date (June 2015) our PF converges on snapshot 16, with an average drift of 39m. The U-Net behaves poorly without the additional modules. The planar drift is measured from the centre of the localised snapshot to the actual GPS recorded location of the UAV in the air. [7] is not capable of absolute localisation.

In comparison with recent state-of-the-art algorithms, that the proposed solution achieves close results. The proposed algorithm is applied on the same snapshot sequences and maps used in [7]. The first dataset is a sequence of 17 snapshots (from Sensefly Merlischachen dataset [2]) which was captured on April 2013, was used along a Google Earth map of the same area which was captured earlier on May 2012. This is a time gap of 11 months, which is much shorter than the more challenging version which was used in our global localisation test above (26 months). The proposed system achieves a relative localisation with an average drift of 20.3m is achieved, but even

[1] Online: https://doi.org/10.15131/shef.data.16589621

though it is less accurate than the state-of-the-art average of 7.06m on the same map (see Fig. 4.19); it has the advantage of being capable of global localisation.

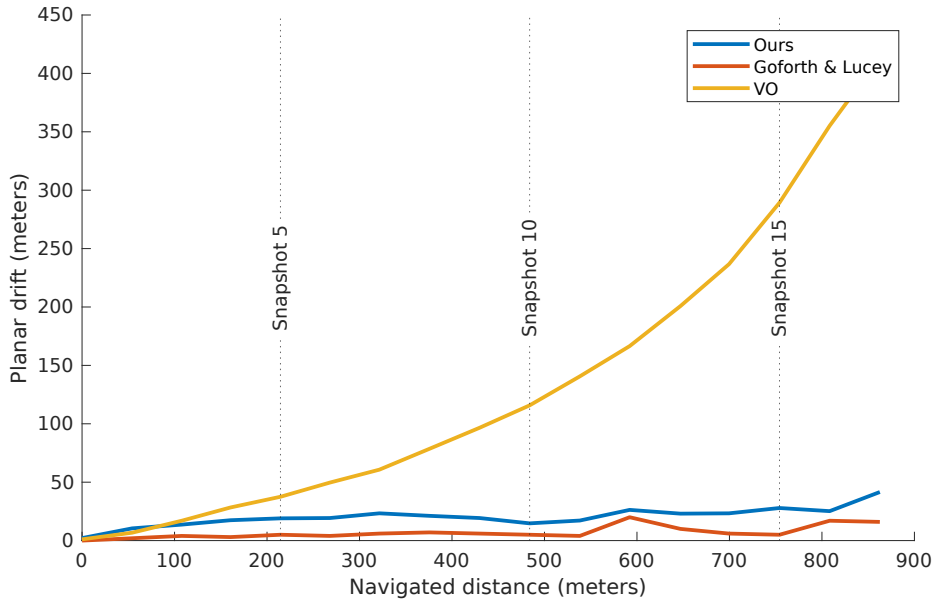


FIGURE 4.19: Comparison with the first map in [7] (relative localisation). Sequence is 17 snapshots long.

The second map used in [7] is more challenging as the houses category is nearly absent. There is a time gap of 19 months between flight date (April 2013) and map capture date (November 2014). The sequence used is 12 snapshots long from Sensefly Willisau dataset [6]. An average drift amount of 38m was achieved, close to the state-of-the-art of 25m (see Fig. 4.20).

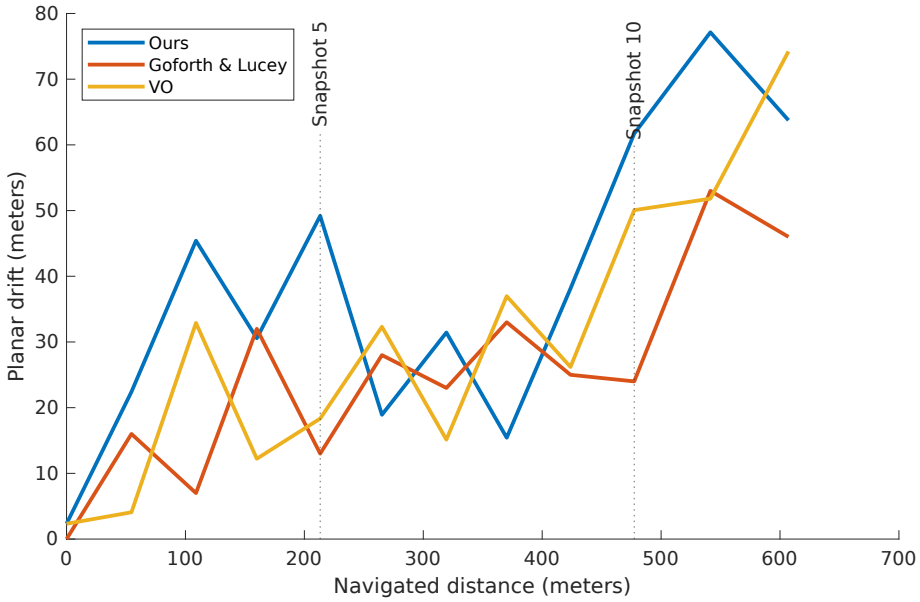


FIGURE 4.20: Comparison with the second map in [7] (relative localisation). Sequence is 12 snapshots long.

## 4.5 Conclusions

In this chapter a vision based absolute global localisation system that uses Google Earth had been designed. The proposed solution is a CNN segmentation based algorithm which uses a particle filter for localisation. The key to the success of the algorithm is taking advantage of features common to both maps and aerial photographs, namely the categories of objects and the relations between them. By taking advantage of those relations it was demonstrated that it is possible to localise a UAV using sparse snapshots and a map given a lightweight CNN which is trained on a popular aerial dataset [3] achieving modest segmentation results on both snapshots and map.

It was demonstrated that the proposed solution could achieve localisation performance close to state-of-the-art algorithms, with the additional advantage of being capable of global localisation. More accuracy would have been achieved if a higher frame rate was available, or if other measurements from other sensors such as an IMU were acquired. In those cases, it would be possible to *fuse* the estimate from our system with other estimates coming from the extra sensors in a *sensor fusion* mechanism. The proposed solution has the potential to enable many safety measures that can be used in emergencies such as homing (i.e. return back to take-off position) and safe landing.

A natural question would arise: Is it possible to improve the system further? Today, deep learning is becoming the state-of-the-art solution in many diverse areas. Therefore a logical question imposes itself: Is it possible to

integrate more deep learning into the system? Therefore in the coming chapter, the possibility of integrating more deep learning into this system will be investigated.

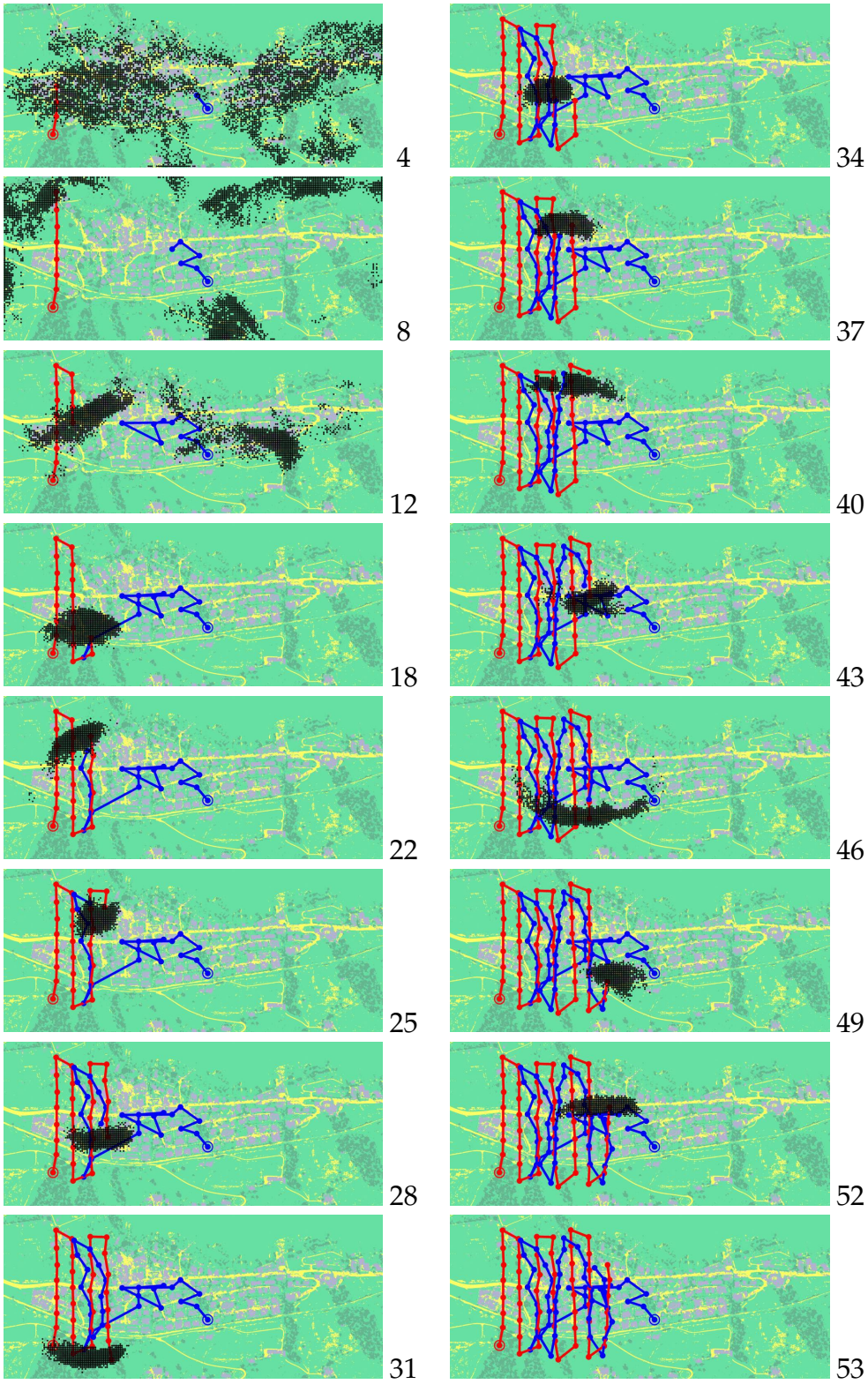


FIGURE 4.21: System performance continued from Figure 4.8. The red path is the ground truth, the blue path is the estimated UAV location. The PF converges on snapshot 16. Notice that before convergence the estimated location is inaccurate because its location is estimated by averaging the locations of all particles.

# Chapter 5

# Deep Image Registration and Visual Odometry

The previous chapter proposed a UAV localisation and homing solution which was completely independent from GNSS. The main component which enabled this capacity was a CNN which could segment both the captured snapshots and the map without being trained on datasets derived from neither the map nor the snapshots. The proposed algorithm could localise the snapshots globally on the map. The current chapter deals with the question: Can deep learning be extended to other parts of the system which are currently solved using non-learning algorithms, namely, the motion estimation (visual odometry) component, and would it make any improvements?

## 5.1 Introduction

In the previous chapter it was shown how image matching against a map using deep learning can be competing to traditional methods for localisation. It can even be superior to them at other tasks (such as global localisation). Given the success of deep learning in many applications, it is natural to wonder about the possibility of getting performance improvements by extending the deep learning into other parts of the system. By looking at the sketch of the localisation system introduced in the previous chapter (Figure 4.1), it can be noticed that the motion estimation (VO) operation is completely independent from the segmentation. It used the input frames directly to find the motion between two frames. But can this architecture be modified to allow the integration of more learning into the system (Figure 5.1)?

The aim of this chapter is to investigate the possibility of converting the motion estimation (VO) algorithm introduced in Chapter 3 and used in Chapter 4, into a learning solution. This task is going to be addressed by investing the segmented snapshots which were acquired by the segmentation network of the previous chapter in motion estimation, through deep registration.

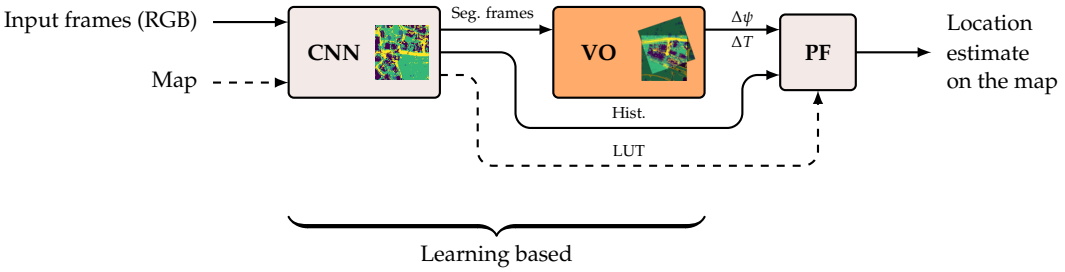


FIGURE 5.1: Localisation system diagram. The segmentation CNN was proposed in Chapter 4. In Chapter 3, frame to frame motion estimation (VO) was proposed (either using homographies or PnP algorithm). In this chapter, the aim is to convert the motion estimation (VO) segment to become learning based.

Traditionally, the transformation between two frames is found by extracting specific point features from both frames, then matching them. This is possible in monocular RGB images (see top row in Figure 5.2), and this is how it was done in Chapter 3. But if the pixel intensities of the segmented frames are to be considered, considerable amounts of useful information which are currently being ignored, could be invested in matching (see bottom row in Figure 5.2).

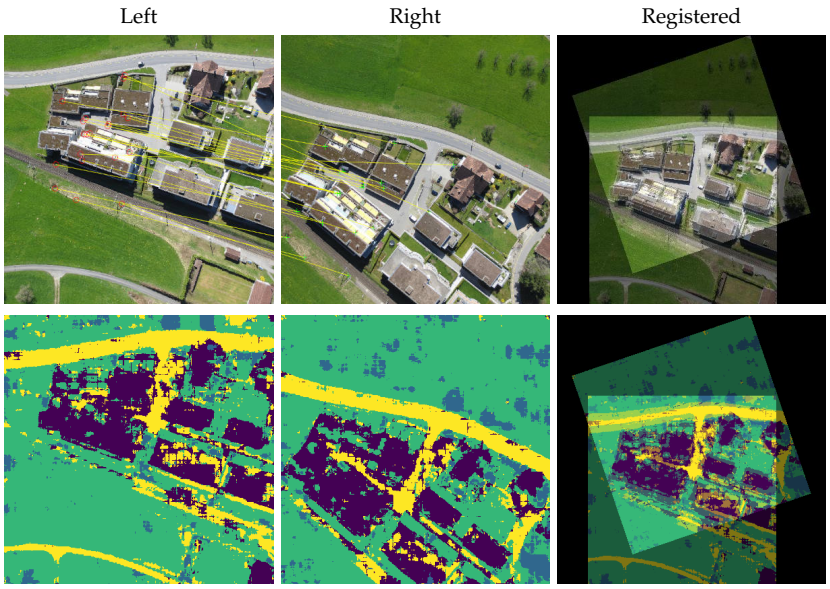


FIGURE 5.2: Top row: Traditional image registration through feature detection and matching. Bottom row: Global pixel intensity based registration.

Two main challenges arise at this point against doing deep registration:

1. It is well known [44] that global matching methods which rely on pixel intensities for matching are less accurate than feature based methods.

2. It was shown in many researches how traditional (non-learning) registration and matching methods outperform learning based ones. This

is mainly because 1) the ground truth of learning methods is acquired from traditional methods, and 2) the capacity of the deep network to generalise beyond the training dataset is very limited on greyscale photos.

However, the argument of using the segmented snapshots as the object of learning has stronger incentives:

1. In the case of fixed-wing UAV navigation introduced in this thesis, no frames bear to be lost. As there is a single line of frames which draws the UAV path. Therefore, due to the sensitivity of motion estimation from feature correspondences, it would be advantageous to switch to deep matching when the RANSAC correspondence algorithm does not meet a good consensus threshold [185].

2. Existing deep registration networks fail to register greyscale photos with 256 labels not belonging to the dataset used for training. It would be interesting to know if this case still applied to segmented photos composed of only 4 labels instead of the normal 256 greyscale colours range. Does the reduction in labels make the registration task easier to address?

3. Since traditional registration methods outperform deep ones, it was suggested that it is preferred to use the deep VO methods only in case of temporary failures of traditional methods [223]. This argument adds to the redundancy of the localisation system.

Many of the available registration convolutional neural networks follow a siamese architecture. Given that the purpose of the deep VO networks is to find a relationship between two frames, it is reasonable that the siamese approach is taken. The same types of features (whether points or regions) have to be extracted from the two input frames. These features are eventually matched and used to find a relationship between the two input frames and to produce the corresponding translation and rotation transformations.

The rest of the chapter is organised as follows: In Section 5.2 a review of existing state-of-the-art deep registration/VO methods is presented. The solution is proposed in Section 5.3 and evaluated in Section 5.4. Finally a conclusion of this work is presented in Section 5.5.

## 5.2 State Of The Art

As previously defined, Visual Odometry is done by estimating the changes in location and orientation of the agent across frames. In simple terms, it is done by estimating the pose changes of the agent from one frame to another using a user defined function. When all of these poses are combined, they amount to what is called visual odometry. Motion estimation is done by matching (or registering) the frames to each other. Traditional (non-learning) image registration techniques can be classified into two categories:

1. Feature based methods: Which first do feature detection on two input frames using algorithms such as SIFT, SURF, ORB.. And then matching those features from one frame to another. Using the correspondences it becomes possible to find a relationship between frames.

2. Intensity based methods (also called global, or wholistic methods): Optimisation based methods which follow an iterative approach so as to minimise a cost function or a similarity measure.

In deep learning, pose estimation follows the same approach as traditional methods. They take an input of two images, and produce the transformation between them. The only difference is in the way the function is estimated: it is not defined by the user, rather by a learning based neural network.

Existing deep VO methods imitate both of these techniques: feature based and intensity based; in addition to a third technique not normally applied in traditional methods which is based on optic-flow. Available deep image registration systems, like other deep image processing networks, rely on CNNs. And just as some conventional image registration methods are iterative, such as, Mutual Information (MI), deep methods are iterative as well, as the CNN is trained iteratively, but eventually the prediction is done in a single iteration.

Given a fixed image $f$ and a moving image $m$, existing deep VO and registration networks can be classified into the following 3 categories:

1. Rigid registration: Also called *linear* or *affine* registration. In this kind of registration all pixels from image $m$ are transformed by the same amounts of translation, rotation and scaling, so as to maximise a similarity measure with $f$. This is the kind of registration that is most related to the application of this chapter.

2. Non-Rigid registration: Also referred to in literature as *non-linear*, *warping*, *deformable* and *morphing* registration. Here, an irregular transformation field or an optic flow field is predicted so as to transform the pixels in $m$ to match ones in $f$, maximising a similarity measure. Each pixel in $m$ is transformed separately.

3. Spatial Transformation Units (STNs): is a kind of modular network that does not predict transformation parameters (nor a registration field), rather it applies such parameters to the moving image $m$, which in turn is compared to $f$ using a similarity measure.

Available rigid registration methods are used in two main kinds of systems: Deep Visual Odometry, and deep image registration (medical and aerial images). These methods are classified in Table 5.1, each category is discussed in detail in the following sub-sections.

| | Supervised | Un-Supervised |
|---|---|---|
| **Rigid** | [224, 225, 226, 227, 228, 229, 230] | [121, 231] |
| **Non-Rigid** | | [231, 121] |
| **Prior Concat.** | [224, 225, 226] | [121, 232, 233] |
| **Ensuing Concat.** | [227, 228, 229, 230] | [231] |
| **MSE-Loss** | [228, 229, 226, 230, 225, 224, 234] | [235] |
| **NCC-Loss** | | [231, 121, 236, 237, 235] |
| **Other Loss** | Geodesic [225] | MAE [233], Custom [232] |

TABLE 5.1: Classification of registration methods.

### 5.2.1 Deep Visual Odometry Networks

Deep VO methods accomplish the task of visual odometry by predicting the camera poses across frames. This is done repetitively by matching each frame at time $t$ with the following frame at time $t+1$, until final frame at time $n$ is reached. Deep VO networks can be classified into two categories: networks that do concatenation at the beginning (prior concatenation), and ones that do concatenation at the end (ensuing concatenation) (see Figure 5.3). All of deep VO networks do rigid registration.

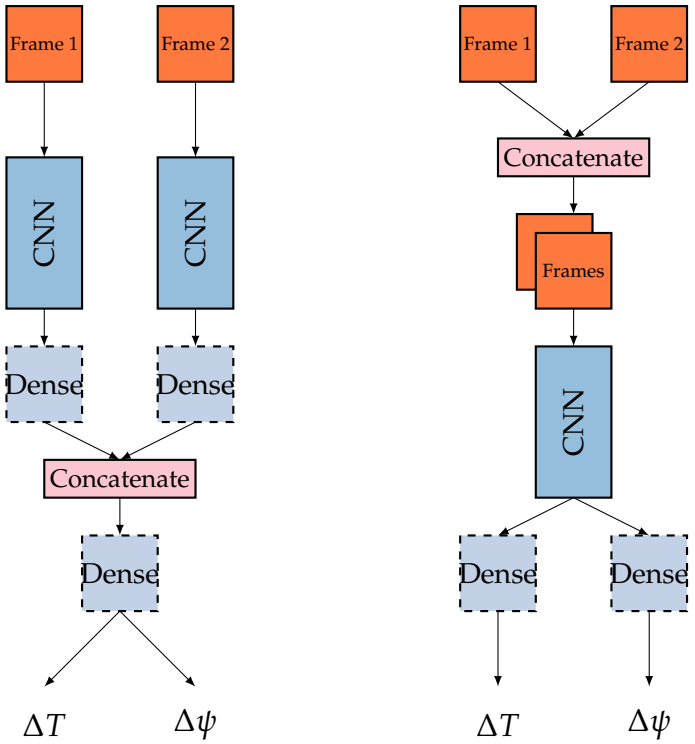


FIGURE 5.3: Deep VO network types: concatenation is either done at the end (ensuing concatenation), on the left, or at the beginning (prior concatenation), on the right. Each dense block is composed of 1 or more fully connected layers, which are used to regress the rotation and translation changes.

**Ensuing Concatenation:** The general procedure followed by this kind of networks is to extract features from two frames at times $t$ and $t+1$, then match those features to find the pose. The general network architecture is composed of two branches (following a siamese architecture), each branch is a shared CNN which is used for feature extraction. Then the final outputs of the two CNNs are concatenated and connected to a layer or more of fully connected layers, which does the task of feature matching, eventually regressing the required rigid parameters (rotation, translation and/or scaling). This kind of networks belong to the ensuing concatenation category, and such networks generally follow the architecture described in left of Figure 5.3.

The first learning based framework that was proposed to do pose estimation was DeepVO [234]. The network is siamese, each branch consists of a bare (untrained) AlexNet [100] and two fully connected layers. The two branches are then concatenated and connected to 3 fully connected layers, the final output of which regressed 3 values: two for translation on $\Delta x$ and $\Delta y$ axes, and one for rotation $\Delta\psi$. The network was trained on the KITTI autonomous driving dataset [110].

This class of networks was later improved by using pre-trained CNN models such as, VLocNet [238] and VLocNet++ [125]. This network has two ResNet-50 branches which are concatenated and connected to two fully connected layers to regress 6 variables representing the 6 degree-of-freedom global pose. Many other networks with similar architectures were also proposed [227, 228, 229, 230, 231]. Other variations include using optic flow maps [239], passing detected features along with input images [234], doing segmentation along with VO [125] and predicting depth along with VO [240].

**Prior Concatenation:** Appearing after the ensuing concatenation models, this kind of networks does not follow the siamese architecture. Rather the two input frames are concatenated, and they are processed by a single CNN branch (as described in the right of Figure 5.3). For example UnDeepVO [232], which relies on VGG network architecture [102], and ends with two branches of fully connected layers to regress 6 values representing changes in rotation $(\Delta\phi, \Delta\psi, \Delta\psi)$ and translation $(\Delta x, \Delta y, \Delta z)$.

Numerous variations have been proposed [124, 240, 224, 225, 121, 226, 233, 124] which integrate different components, for example, the integration of Long Short-Term Memory (LSTM) modules after the CNN, such as DR-CNN [241, 242]. Other networks pass features instead of whole images [243].

DeepVIO [233] uses separate networks to estimates the optic-flow between two frames, then it is integrated with IMU data using another network to calculate the trajectory of a stereo camera. Other networks integrate depth like [244, 240], others predict depth which is passed to a separate VO network like [245, 244].

### 5.2.2 Spatial Transformer Networks

Spatial Transformer Networks (STNs) are modular networks which can be inserted into other existing networks, and trained along with them using

backpropagation. They take in an input image or feature map, and produce a transformed version of it. This transformation can be rigid or non-rigid, and this flexibility makes them applicable on both rigid and non-rigid registration. In the case of rigid registration, the STN receives an input image, along with transformation parameters (translation $x, y$ and rotation $\psi$), and it applies those parameters to every pixel in the image. In the case of non-rigid registration, the STN receives a transformation field $\Phi$ and it applies it to corresponding pixels in the input image.

They were first introduced in [246], and their advantage is that they efficiently learn how to adjust the spatial information within a network, so as to maximise the final optimisation objective done by the network. For example they help shift and rotate an input image so as to best match a template image by employing a similarity measure. They help the network invest spatially diverse information instead of ignoring such information which is what CNNs normally do. An STN is depicted in Figure 5.4, and it is composed of the following parts:

1. Localisation network: which is a small CNN which regresses the transformation parameters (whether rigid or non-rigid).
2. Sampling grid: calculates the new location of each transformed feature point (pixel) from the input image into the output image.
3. Sampling (bilinear interpolation): calculates the intensity values of each pixel or feature point in the new location by considering the intensity values of surrounding pixels.

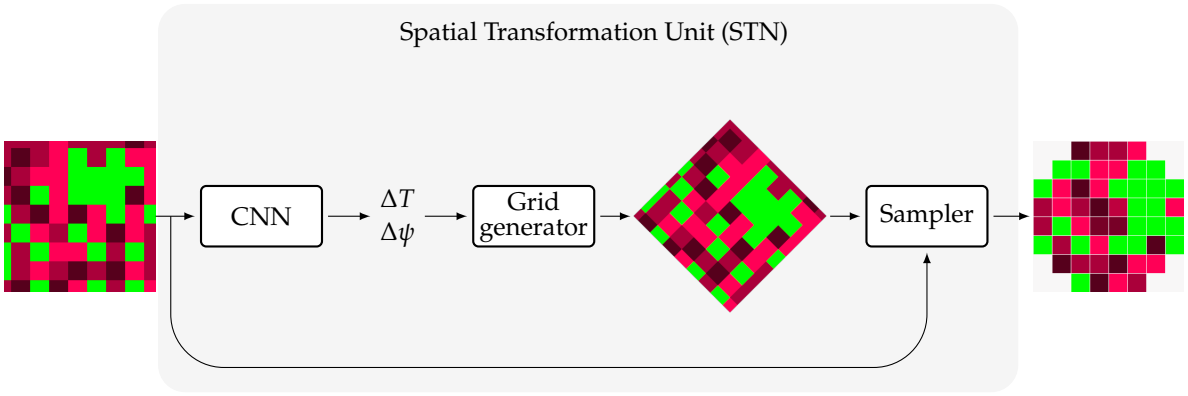


FIGURE 5.4: A Spatial Transformation Unit (STN) which shows its application on a rigid transformation.

Since their introduction, STNs have been successfully invested in many medical image registration networks [231, 121] which are discussed in the following section. It is worthwhile to mention some notable modifications to STNs. In [227] a new module was introduced called Image Transformer Network (ITN) which can be added prior to an STN forming Image and Spatial Transformer Network (ISTN). It was applied on the rigid registration of segmented vessel images. It is composed of a mini CNN which learns how

to optimise the input image received by the STN, maximising the registration accuracy of STN given that a Structures-of-Interest (SoI) regions are provided (such as a segmented map). Another notable work is IC-STN [247] which addresses the boundaries effects produced by STNs due to cropping, by modifying the warping parameters instead of the input image.

### 5.2.3 Deep Image Registration Networks

A different category of networks is called deep image registration, such techniques are mostly applied on medical and aerial images. A big portion of these systems follow non-rigid registration, but some of them follow a rigid registration approach which is the focus of this chapter.

Deep registration networks aim to accomplish a similar task as Deep VO methods, but instead of predicting a camera pose, they attempt to register one image on top of the other. They match a moving image $m$ to a fixed image $f$. Despite the fact that both of these methods are used to solve different problems, but both of them serve the objective addressed by this chapter.

#### 5.2.3.1 Aerial Image Registration

Also called remote sensing image registration. Existing aerial image registration systems' final aim is to overlay one image on top of the other. In general, aerial image registration systems can be classified into 3 categories:

1. Feature matching registration: follow the same architecture as Deep VO, such as, [248, 249, 250, 251, 252, 253, 254, 255, 256, 257]. They extract features using a CNN, then match them either using a separate algorithm, or using another CNN. Even if some of these methods do not predict the registration parameters (such as rotation and translation), in theory, it is possible to calculate them from the correspondences.

2. Homography prediction: Predict the locations of the 4 corners of moving image $m$ in fixed image $f$, such as, [258, 259]. This is not necessarily useful for the applications of this chapter, as the 4 corners have to adhere to rigid transformation constraints.

3. Global registration: The only kind which is non-rigid. The pixel intensities in each image are considered as a whole, and a CNN is trained to compare two images by maximising a similarity measure like Normalised Cross Correlation (NCC), Mutual Information (MI) or Mean Square Error (MSE) [260].

#### 5.2.3.2 Medical Image Registration

Deformable registration networks are applied on medical image registration with the purpose of finding the segmentation map of one of the two images being registered. This is done by registering a segmentation ground

truth $A'$ of image $A$, to another unsegmented image $B$. This results in an optic flow field which maps the pixels of image $A$ to image $B$. This optic flow field is eventually applied on the segmented map $A'$, to get $B'$.

In essence, deformable registration morphs a moving image $m$ to match another fixed image $f$, using a transformation field which contains different transformation values for each pixel [121, 236, 261, 230]. This is different than finding rigid transformation parameters which applies the same transformation to all image pixels.

A few of the available medical image registration systems focus on rigid registration, and they follow the same architecture as deep VO systems. For example, [230, 228, 229] follow the later concatenation model, others follow the prior concatenation model [226, 225, 224]. Some are capable of doing both rigid and non-rigid registration [121, 231]. Others prove that registering feature points is more accurate than registering pixel intensities [262].

The majority of medical image registration systems are non-rigid, a seminal work which is a good representative of state-of-the-art medical deformable networks is called DIRNet [237]. DIRNet is an unsupervised network which receives two images: moving $m$ and fixed $f$, and composed of a fully CNN that predicts a transformation field $\Phi$, which is applied to $m$ using an STN producing $m(\Phi)$. Then $m(\Phi)$ is compared to $f$ using a similarity metric, optimising a normalised cross correlation (NCC) objective function (loss) and providing backpropagation to the CNN.

Most of existing deformable networks follow similar architectures as [237], for example the recent state-of-the-art unsupervised VoxelMorph [236, 235]. VoxelMorph contains a U-Net CNN, which concatenates the two input images and predicts a transition field. This field is sent to an STN [246] which warps one of the input image, which is then compared to the other. There is no need for ground truth labels, hence the training is unsupervised. Its advantage is the flexibility in calculating the loss, as in addition to calculating the NCC loss, it introduces another smoothing loss term to supervise the changes in the transformation field. Many later works were built on top of VoxelMorph such as ADMIR [121].

A recent breakthrough which is based on VoxelMorph is the introduction of SynthSeg [263] which recognises the weak ability of CNNs to generalise to images with contrast values different than those used for training. Therefore instead of requiring ground truth segmentations for training data, it uses samples of existing labelled data and produces new data for training using a gaussian generative model. This forces the network to learn to segment different shapes and intensities during training. The most notable registration work built on it is SynthMorph [264, 265], which recognises a disadvantage of all existing deformable networks to generalise to new data with intensities non-existent in training data. It proposes to use randomly synthesised training images, and it proves that this method beats state of the art networks such as [235].

## 5.3 Proposed Method

The overall system architecture is described in Figure 5.1. The difference between this architecture and the one introduced in the previous chapter, is that the VO component no longer needs to do traditional feature detection and matching on input RGB snapshots. Instead, it works on the segmented snapshots coming from the segmentation CNN, using 4 greyscale values instead of 256 which is normally used. In addition, it is now learning based.

The new proposed VO module is another CNN which receives 2 input snapshots captured at times $t$ and $t+1$. The snapshots are segmented and they are composed of 4 labels. The CNN predicts two numerical values: the amounts of yaw rotation $\Delta\psi$ and translation $\Delta T$ between the two input frames.

The main challenge that the research in this chapter faced is the lack of research in rigid segmented image registration, which especially have big amounts of rotation and translation. It was unclear from the beginning of the research which direction had to be taken which would enable this kind of registration. The planned CNN could follow one of numerous different architectures, as most of the existing works seemed plausible to begin with. Therefore, a series of experiments had to be done so as to reach the optimum network architecture. The progress towards the optimum solution can be summarised as:

1. Deep VO: Experimentations started with deep VO network architectures (prior and ensuing concatenation, as will be explained in the following section), and their results were evaluated on the proposed dataset.

2. Deformable networks: A semi-supervised learning solution is proposed. The network is derived from deformable network registration.

3. Randomised generation: A promising work about adding randomness to the training images is evaluated for better addressing the problem.

The results and discussion of each one of these experiments are described in the following section. Conclusions are finally made in Section 5.5.

## 5.4 Performance Evaluation

In this section the dataset setup is described first. Then the experiments listed in the previous section are described, each with its own corresponding results and discussion.

### 5.4.1 Experimental Setup

The main dataset used for training is ISPRS Potsdam [3], which is the same one used to train the segmentation CNN in Chapter 4. This dataset

contains 38 high resolution aerial images along with their ground truth segmentations. Each segmented image contains 6 different categories (impervious surfaces, buildings, low vegetation, tree, car and clutter). For the purposes of this work, the 'car' label is combined with 'impervious surfaces' to create a 'road' label, and 'clutter' is combined with 'low vegetation' to create a new 'grass' label. This results in images which contain 4 categories, the same number of categories predicted by the segmentation CNN. Figure 5.5 outlines sample photos from Potsdam dataset.

FIGURE 5.5: Sample Potsdam DS photos used for augmentation. First line: RGB Potsdam photos. Second line: Original labels. Third line: The combined labels used for augmenting the required dataset.

Using the high resolution images in Potsdam dataset (a total of 38 images, each with the size of 6000x6000 pixels), an augmented dataset was created. The augmented datset consists of pairs of images with 60% minimum overlap. To produce those pairs, each one on the 38 images was augmented as follows:

Let a Potsdam photo be called *im*, each photo is augmented as follows:

1. For each photo *im* of 38 photos:

2. Resize *im* to a size that matches the scale of the standard fixed-wing UAV altitude.

3. Randomly rotate *im* in an angle between -179° and +179°.
4. Crop a square inside the result to acquire the first snapshot.
5. Crop a random square inside the unrotated *im*, to acquire the second snapshot. Cropping should be done so that there is a minimum overlap of 60%, a chosen initial value.
6. Repeat 190 times for each photo.

This procedure produces 7220 augmented pairs of images (see Figures 5.6 & 5.7 for sample photos).

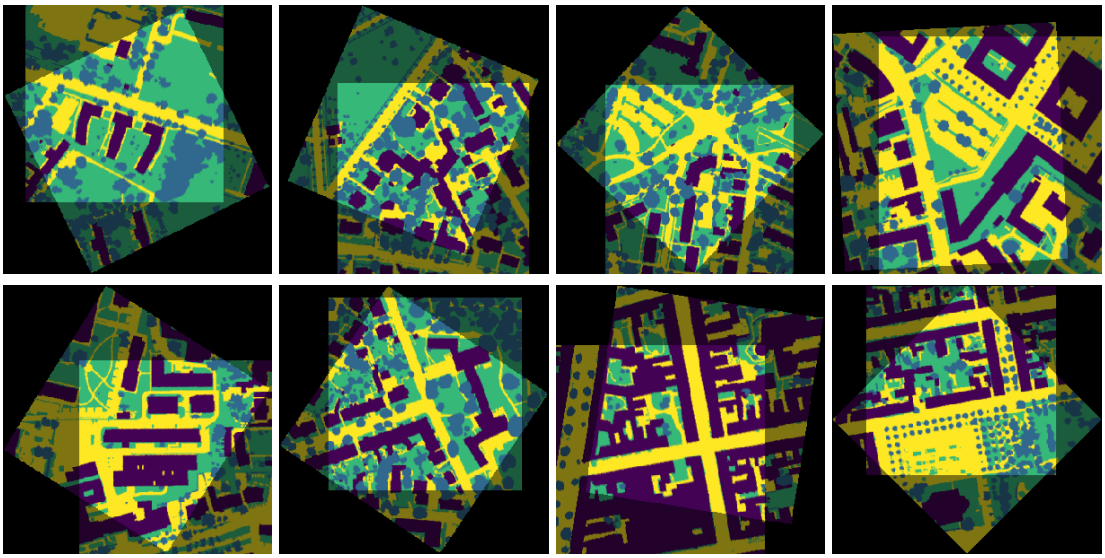

FIGURE 5.6: Samples of augmented Potsdam DS photos used for training the registration CNN.

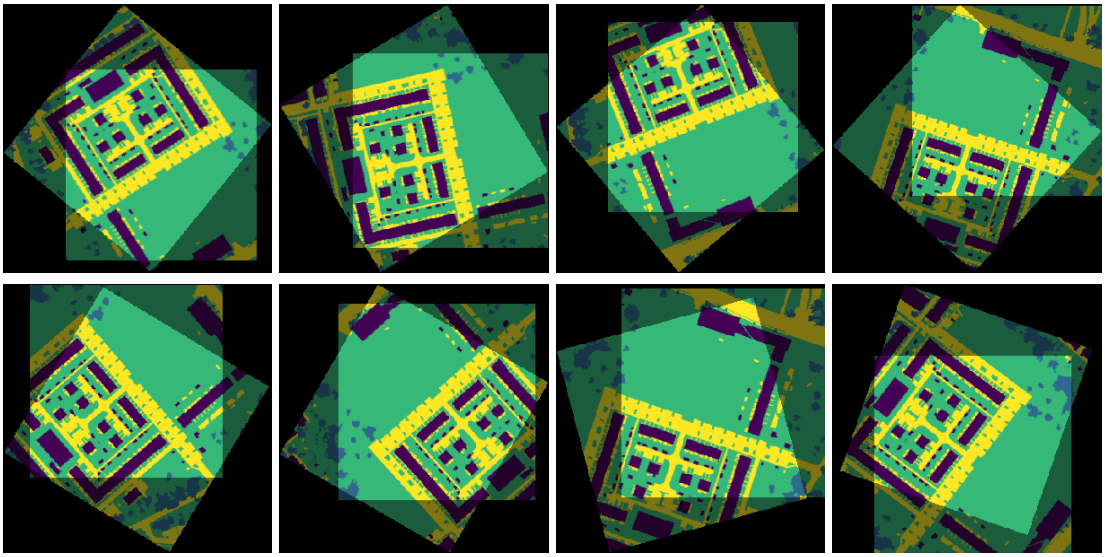

FIGURE 5.7: Samples from a single augmented Potsdam photo.

### 5.4.2 Rigid Deep VO: Prior Concatenation

Research started by replicating the *deep VO* network architectures. It seemed reasonable that such architectures would work. As existing works solve a similar problem to the one being addressed: the inputs are two frames, and the objective is to find a relationship between them, which is a transformation from one frame to the other.

An initial model was created which followed the Prior Concatenation architecture described in Figure 5.3. The model relied on a pre-trained MobileNetV2 [105] network base, the same light weight network that was chosen for the segmentation network in Chapter 4. The base was connected to fully connected layers, the last of which regressed the translation $\Delta T$, which is composed of two values $(\Delta x, \Delta y)$; and rotation $\Delta \psi$ values.

#### 5.4.2.1 Results

Numerous architecture settings were tested, for example, numerous base CNN architectures were tested (bare such as AlexNet [100] or pre-trained like MobileNetV2 [105]), the number and width of dense layers, dropout, optimisation algorithm, learning rate.. etc. But none gave good results, in all tested settings no matter how the network structure and its parameter configurations were changed, the network would not learn successfully, it would keep overfitting, i.e. memorising the input images, but not finding a generalisable relationship between them.

#### 5.4.2.2 Discussion

After deeper investigation into existing deep VO networks, it becomes easy to notice that they are applied on video frames captured at high frame rate, the same kinds of videos used to do SLAM. Which means that the two input frames at times $t$ and $t+1$ have a very high overlap. And when they are concatenated on top of each other, one of the two frames needs a small deformation to match the other. This deformation can be represented by an optic flow field (see Figure 5.8):

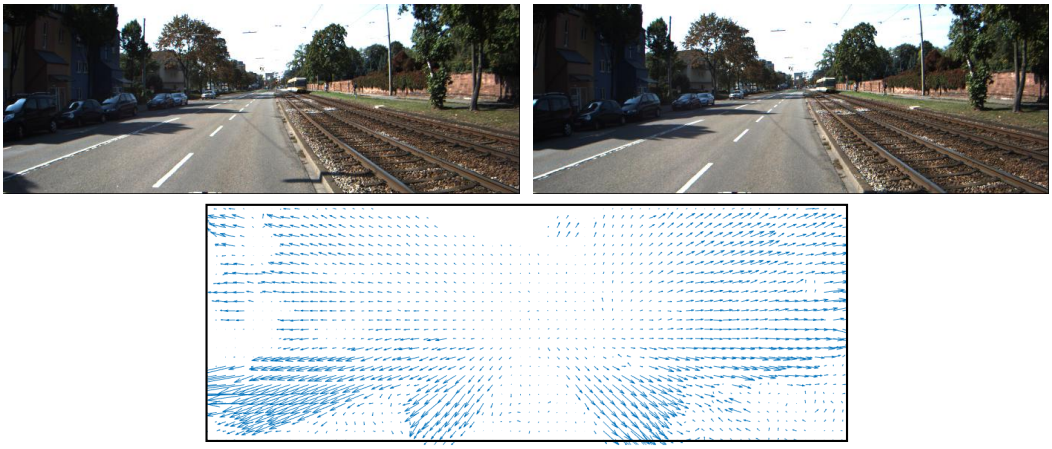

FIGURE 5.8: Top: Two input frames at times $t$ and $t+1$ from KITTI driving dataset. Bottom: The resulting optic flow map which captures the movement from one frame to the other.

In prior concatenation deep VO networks, the two frames are concatenated at the outset, and later they are convolved by the CNN. Since the difference between the two frames is minimal, it becomes possible to capture the feature transitions from frame $t$ to frame $t+1$ within a single kernel. Figure 5.9 shows how it is possible to map two diagonal edges using a $5 \times 5$ convolution kernel:

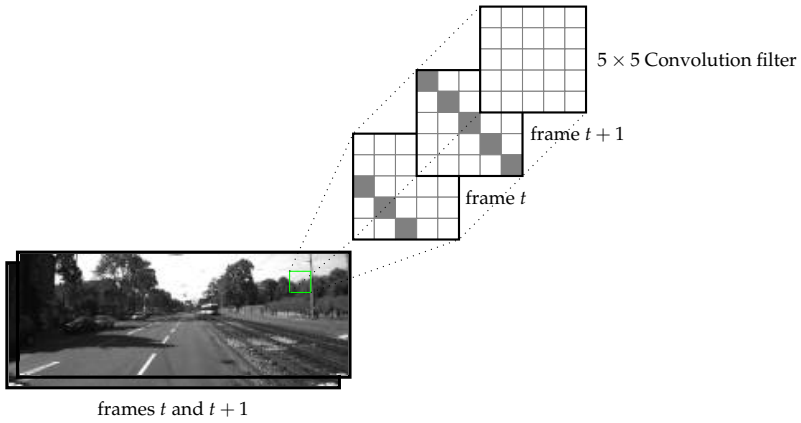


FIGURE 5.9: High overlap between two photos means that the convolution filters of a CNN can easily find a mapping between the pixels of two concatenated images.

It can be understood that the prior concatenation networks first find an optic flow map, then with the help of fully connected layers, they average the whole map and convert it into a transformation matrix. This is precisely the reason why such architectures failed in the case of aerial photos with little overlap. Because since the overlap was not big enough, it was not possible to find the initial optic flow field using convolution filters. By comparison to Figure 5.9, the second layer of pixels $t+1$ contained pixels which are neighbouring to the corresponding pixels in the first layer $t$. Such neighbouring pixels do not exist in the case of aerial images with limited overlap. Even

when it was attempted to increase the size of the convolution filters to exceptionally big sizes, e.g., 20 or 30, overfitting remained and generalisation was unsuccessful. Therefore a different path had to be taken.

### 5.4.3 Rigid Deep VO: Ensuing Concatenation

Since concatenating the two input frames at the outset (before being convolved by the network) did not produce good results, another network was created following the ensuing concatenation architecture. The base CNN remained similar, with the exception of converting the architecture into a Siamese (double branched) network, and adding fully connected layers at the end. Each branch received an image which would be convolved by the CNN. The processed feature maps would be concatenated, then finally regressed to do a regressive prediction of 3 values: two values representing changes in surface translation $(\Delta x, \Delta y)$, and one representing change in yaw angle rotation $\Delta\psi$.

#### 5.4.3.1 MobileNetV2 Results

The following is the network architecture which was adopted when a pre-trained MobileNetV2 was used as the base CNN. The weights of MobileNetV2 were frozen (in preparation for transfer learning). The output feature maps from layer *block_16_project* were taken from MobileNetV2 (this layer which has low resolution is immediately behind the last output layer of MobileNetV2). Then the feature maps were passed to a fully connected layer, which were then concatenated and eventually regressed through a series of dense (fully connected) layers. The architecture can be seen in Figure 5.10.

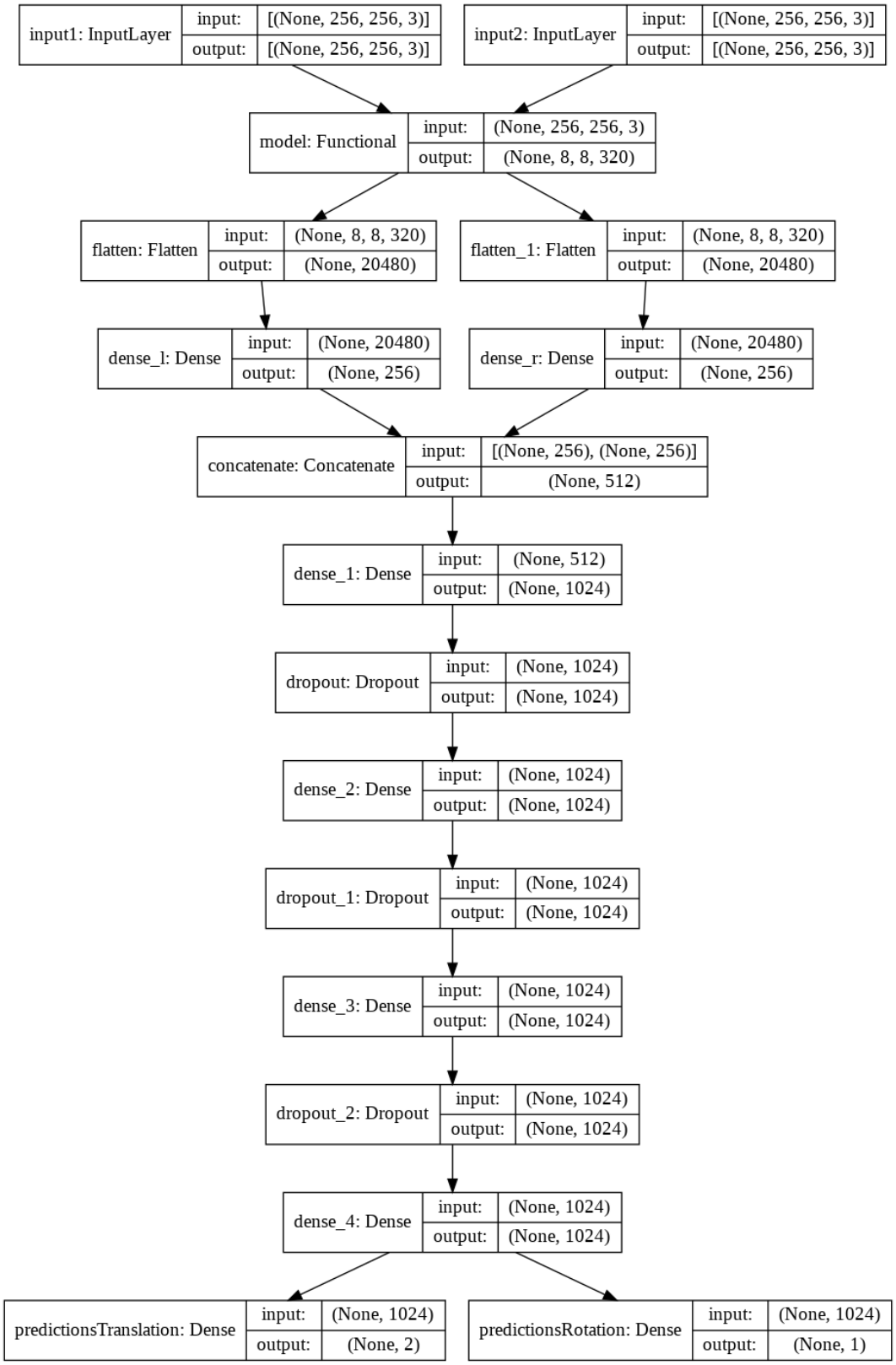


FIGURE 5.10: Ensuing concatenation network structure. The 'Functional' module is the MobileNetV2 CNN. The 'Dense' layers are fully connected layers.

The network contained 16m parameters, and it was trained on an Nvidia

V100 GPU for 500 epochs. Adagrad optimiser was used (which does not require any manual tuning of the learning rate) with MSE loss for the translation, and MAE loss for rotation (scaled by 20 factors to compensate for the low values). MSE and MAE losses are commonly used for regression problems. The progress of the resulting loss showed good learning. The loss curves showed that the network was learning well as its performance on the validation set was as good as the training set. Both curves were decreasing hand in hand, as can be seen in Figure 5.11:

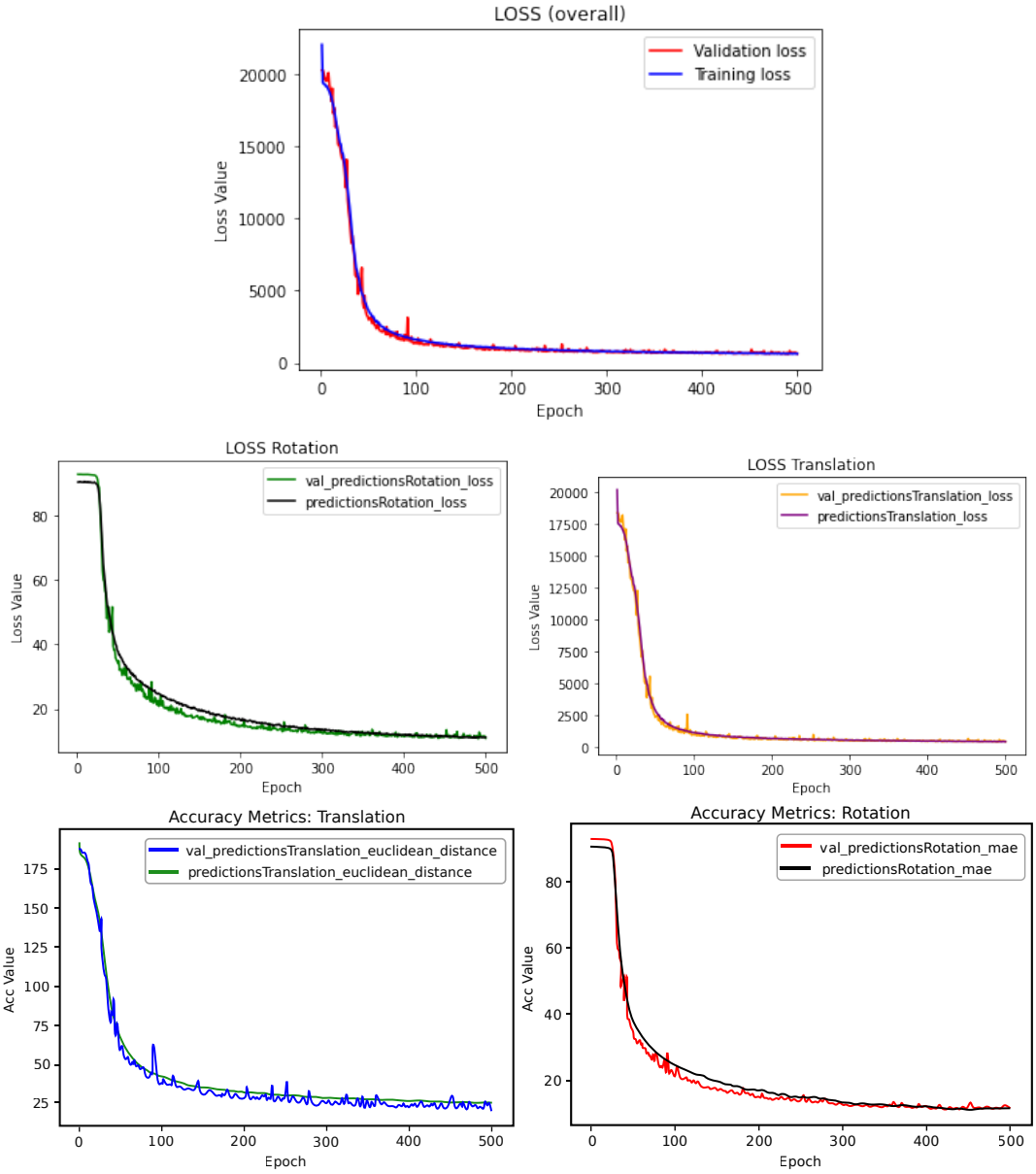


FIGURE 5.11: Loss and accuracy training curves resulting from using MobileNetV2 CNN transfer learning.

When the trained network was tested on the testing set, the results were as follows (Table 5.2):

| Prediction | Average accuracy |
|---|---|
| Rotation MAE (yaw angle) | 11° |
| Translation (Euclidean distance) | 19 pixels |

TABLE 5.2: Results from using MobileNetV2 in ensuing concatenation architecture.

#### 5.4.3.2 AlexNet Results

Another experiment was performed, where the same network architecture was maintained, with the exception of replacing MobileNetV2 pre-trained CNN with an untrained AlexNet [100] CNN. The overall network structure can be seen in Figure 5.12:

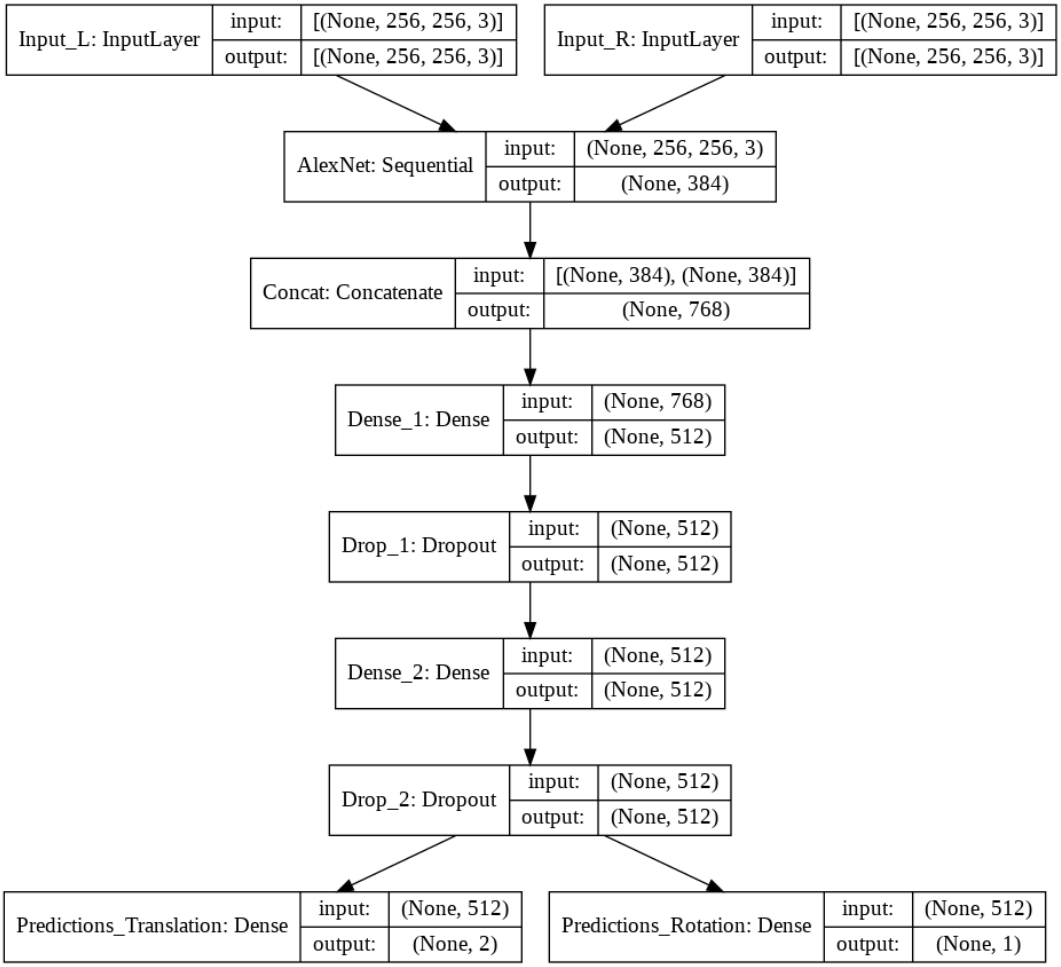


FIGURE 5.12: Network structure. The 'Sequential' model is AlexNet CNN.

The AlexNet CNN architecture used can be seen in Figure 5.13.

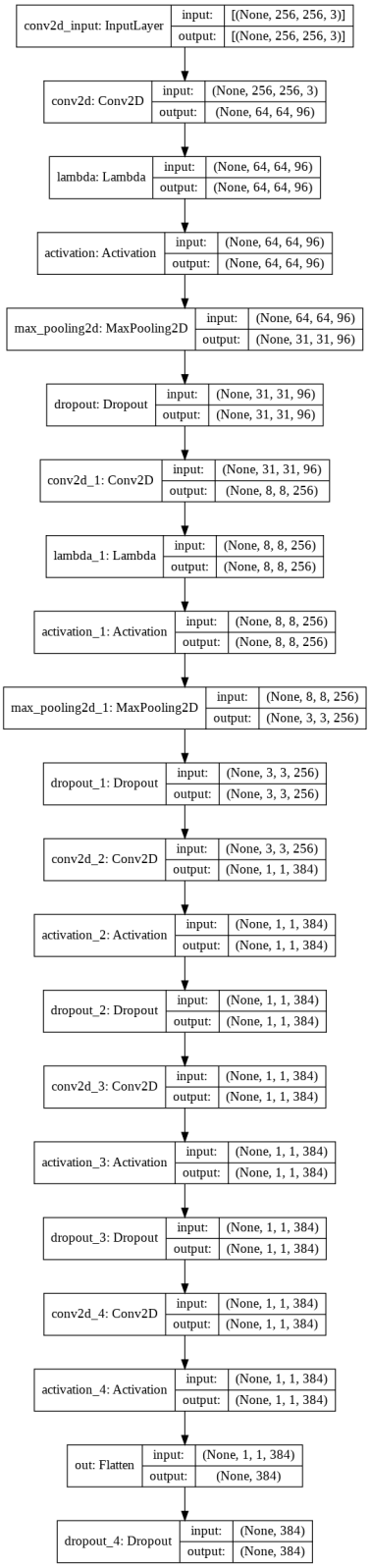


FIGURE 5.13: The modified AlexNet structure which was used.

The final network contained 4.8m weight parameters. It was trained on an Nvidia Tesla P100 GPU for 300 epochs. Adam optimiser was used with MSE loss for translation, and MAE loss for rotation. The rotation loss was

scaled by 10 factors to compensate for low rotation values. The resulting loss curves showed good learning behaviour. They are shown in Figure 5.14:

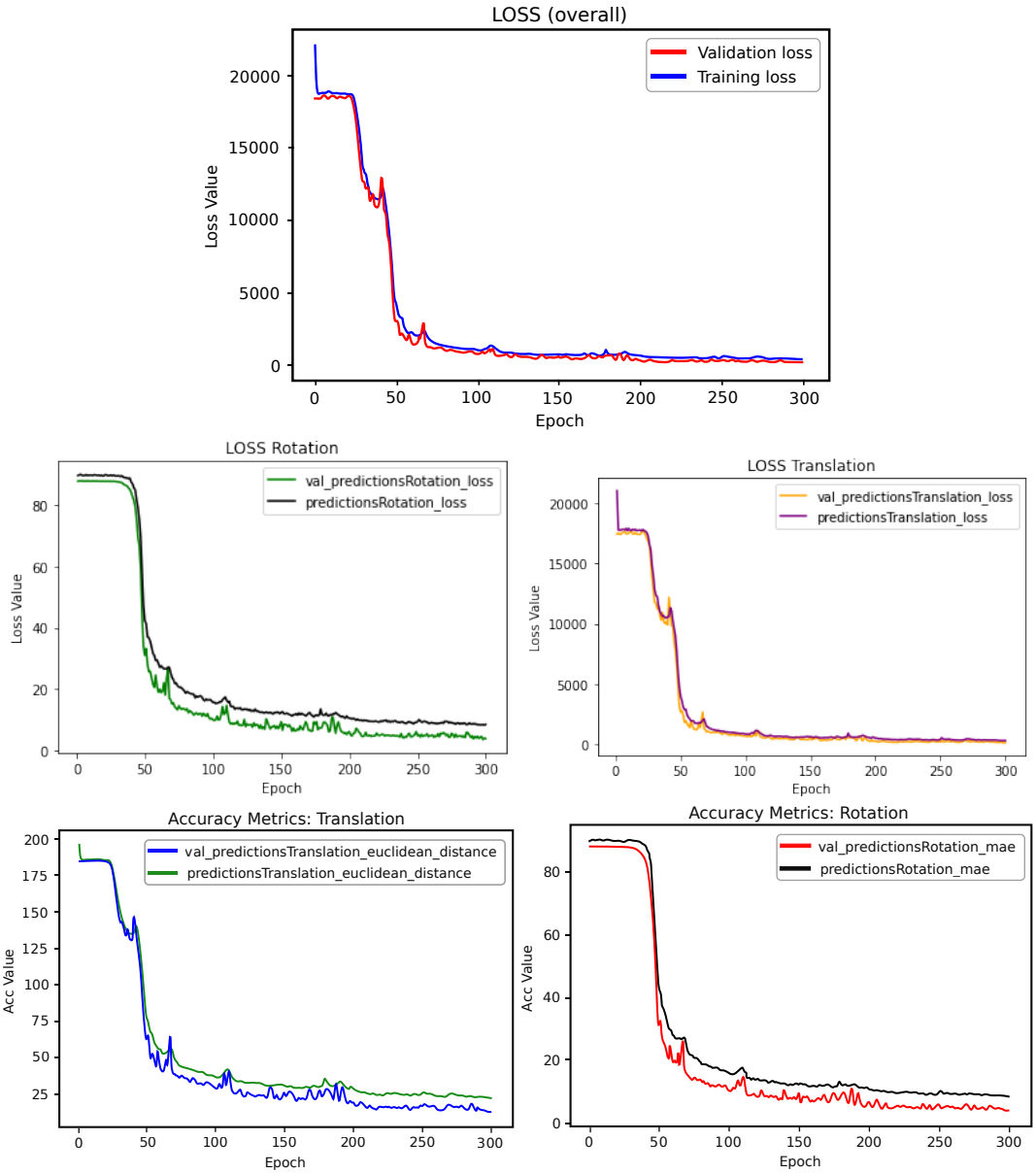


FIGURE 5.14: Loss and accuracy training curves.

The final prediction results on the testing set are as follows (Table 5.3):

| **Prediction** | **Average accuracy** |
|---|---|
| Rotation MAE (yaw angle) | 4° |
| Translation (Euclidean distance) | 12 pixels |

TABLE 5.3: Results from using bare (untrained) AlexNet in ensuing concatenation architecture.

The results above were acquired by training on Potsdam dataset. All images in the dataset were augmented, the augmented results were shuffled

and the resulting set was split into training/validation/testing subsets (see top of Figure 5.15). According to this setting, the network worked well and gave the results above.

However, when the network was tested on segmented images belonging to a different dataset, e.g., the hand labelled versions of Sensefly Merlishachen dataset [2], the results were not as expected. In those cases, the network performed much worse and it was incapable to learn and generalise its knowledge. Surprisingly, similar results were acquired when Potsdam images were split at the outset, i.e., instead of shuffling all photos then splitting, the photos were split into different subsets, then each subset was augmented to produce the full sized subsets (see bottom of Figure 5.15).

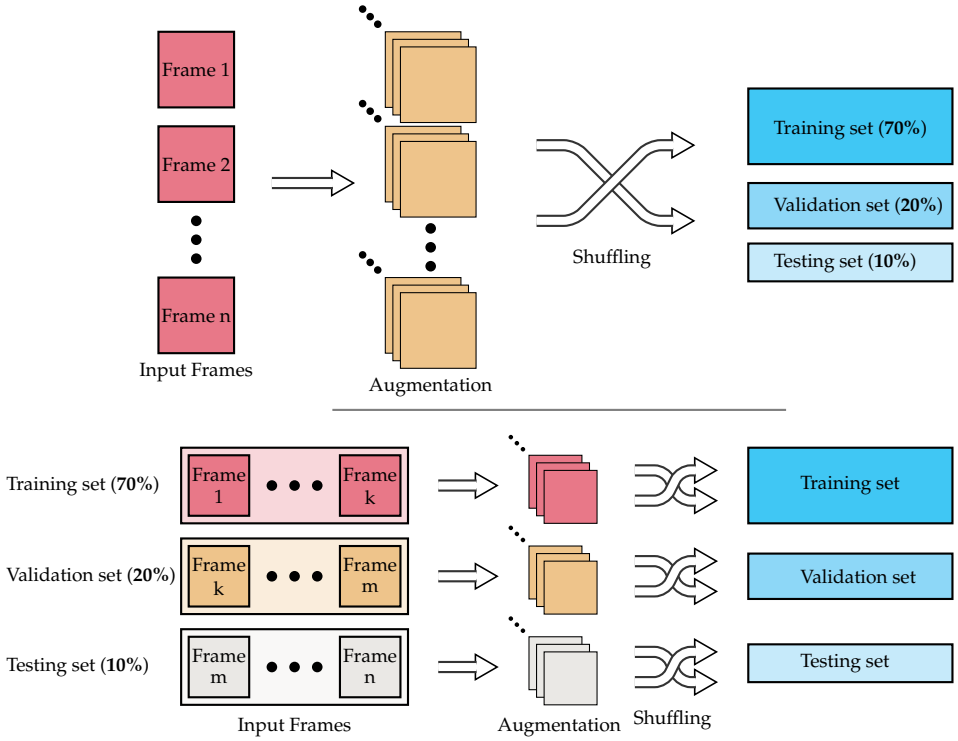


FIGURE 5.15: Dataset splitting strategies: Either augment all photos (all Potsdam DS photos), shuffle then split into 3 subsets (top), or separate photos before augmentation and shuffling into 3 subsets (bottom).

The outcome that a bare network performed slightly better than a pre-trained network is discussed in the following section, where the results above are investigated.

#### 5.4.3.3 Discussion

The ensuing concatenation architecture proved more successful than the prior concatenation, as the network learned successfully and it did not overfit. Also, it was not a surprise that a bare un-trained network such as AlexNet performed much better than using a pre-trained network such as MobileNetV2. As it was shown in [266], using a pre-trained network (transfer learning) does

not help very much if the objective is localisation, as opposed to classification. This is because the majority of the currently available pre-trained networks are trained on on a single dataset: ImageNet [95], with a classification objective.

The acquired results were consistent with this conclusion, as after many trials it was found that an untrained AlexNet gave better results than MobileNetV2. It can be understood that the network works as follows: first, the two branches of the CNN performed feature extraction on both input frames, then the fully connected layers helped find a relationship between the extracted features, in effect doing feature matching. Which is identical to the traditional feature extraction and matching workflow.

Despite the fact that it is well known that neural networks perform well only on datasets on which they are trained, the original assumption which was made in this work was that it would be easier to register segmented aerial photos due to their much simpler nature: they are not RGB images, rather, they are segmented into only 4 categories. I.e., instead of dealing with 256 colours, only 4 colours are needed to be dealt with (see Figure 5.16).

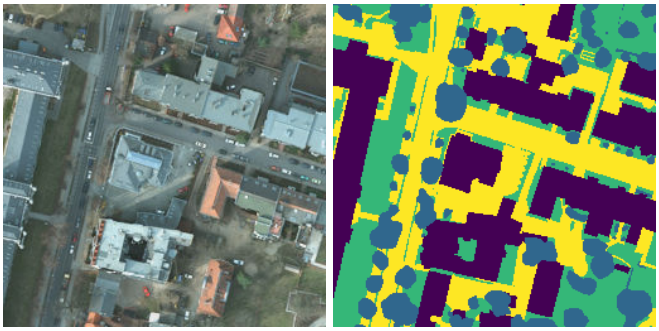

FIGURE 5.16: Comparison between an RGB image, and a segmented image composed of 4 colours.

Therefore, it seemed reasonable at the outset that a network trained on a subset of such images, would be able to generalise its knowledge to register other similar images. However, after proper testing and evaluation, this assumption was proved to be invalid. The conclusion to be made from this is that the number of categories did not matter, rather, the patterns and distributions of those categories played the main role. For the network to learn well, it needed to be trained on patterns which were similar to the testing set. This is when research took a new path, and another category of solutions had to be investigated: The deformable networks.

### 5.4.4 Deformable Networks

Deformable networks are a kind of networks which were designed for medical image registration. They morph a moving image so as to match a fixed image. Applying deformable transformation on aerial images is unhelpful in the case that is being addressed in this chapter (see Figure 5.17). This is because the eventual aim of the desired registration is not only to map

pixels of one image to another, but to find the rigid transformation parameters. However, since they use STNs (which can operate on regressed rotation and translation parameters), it would be possible to modify them to make them suitable for rigid registration.

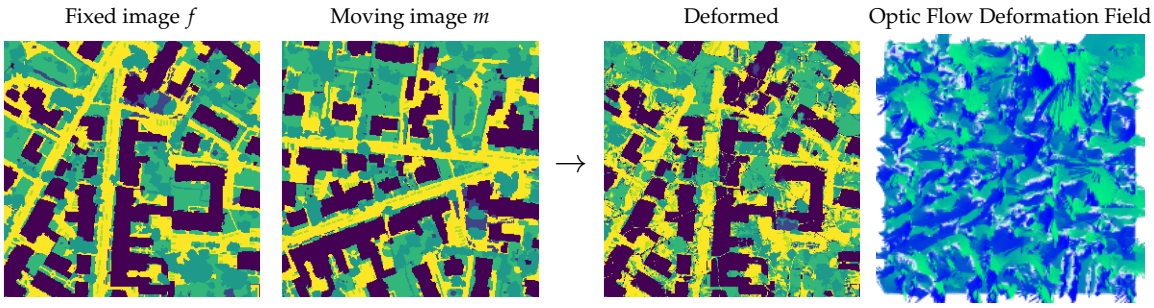


FIGURE 5.17: Deformable registration results on aerial snapshots.

The state-of-the-art work in this field is a network called VoxelMorph[236, 235]. The network receives two images: A fixed image $f$, and a moving image $m$. And it is composed of a U-Net which learns to predict an optic flow registration field $\phi$, which is applied on $m$ using a Spatial Transformation Unit (STN), and then compared to the fixed image $f$. The network architecture was modified as follows (refer to Figure 5.18):

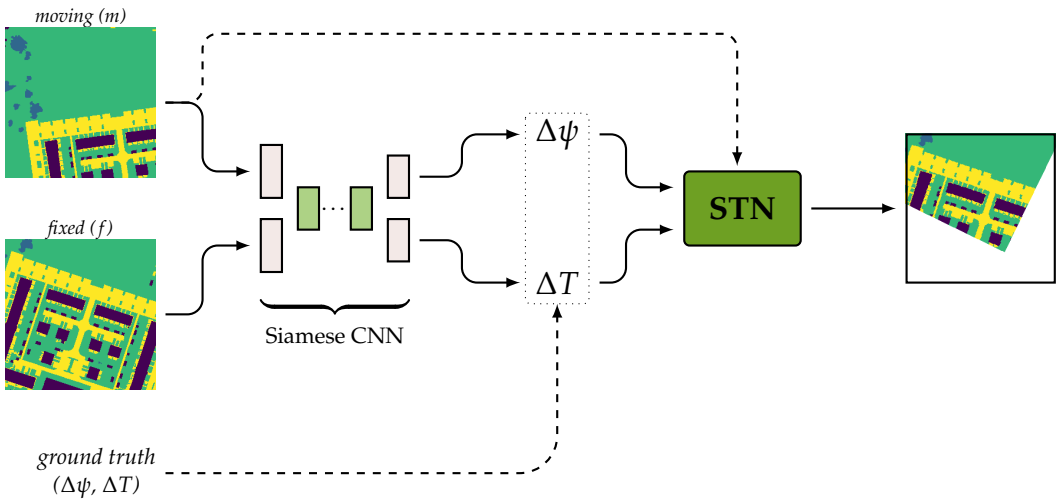


FIGURE 5.18: Proposed rigid semi-supervised architecture. STN is a Spatial Transformation Unit.

1. The U-Net was replaced with a rigid transformation network which regressed rotation $\Delta\psi$ and translation $(\Delta x, \Delta y)$ resembled by $\Delta T$. Instead of concatenating input images, both images were provided concurrently to the best performing siamese model which was achieved before. Which has an ensuing concatenation siamese architecture with untrained AlexNet base (Figure 5.12).

2. Instead of predicting a registration field $\phi$, the network now regressed rotation and translation values. But since the ground truth for these values was already available, they were used as means to provide partial supervision to the network. If the original unsupervised scheme was used, the chances of getting stuck at local minimums become big, as the aerial images have large rotation/translation difference. This is why ground truth rotation & translation values were provided as means to regularise the network. I.e. replace the $L_{smooth}$ loss with a new *semi-supervised* loss. This made the network *semi-supervised*.

3. Two STNs were used sequentially: one to apply rotation, another is to apply translation to the moving image $m$.

The ADAM [267] optimisation algorithm was used, MSE loss was used for regressing the translation, and MAE for regressing the rotation.

#### 5.4.4.1 Results

The resulting architecture can be seen in Figure 5.19.

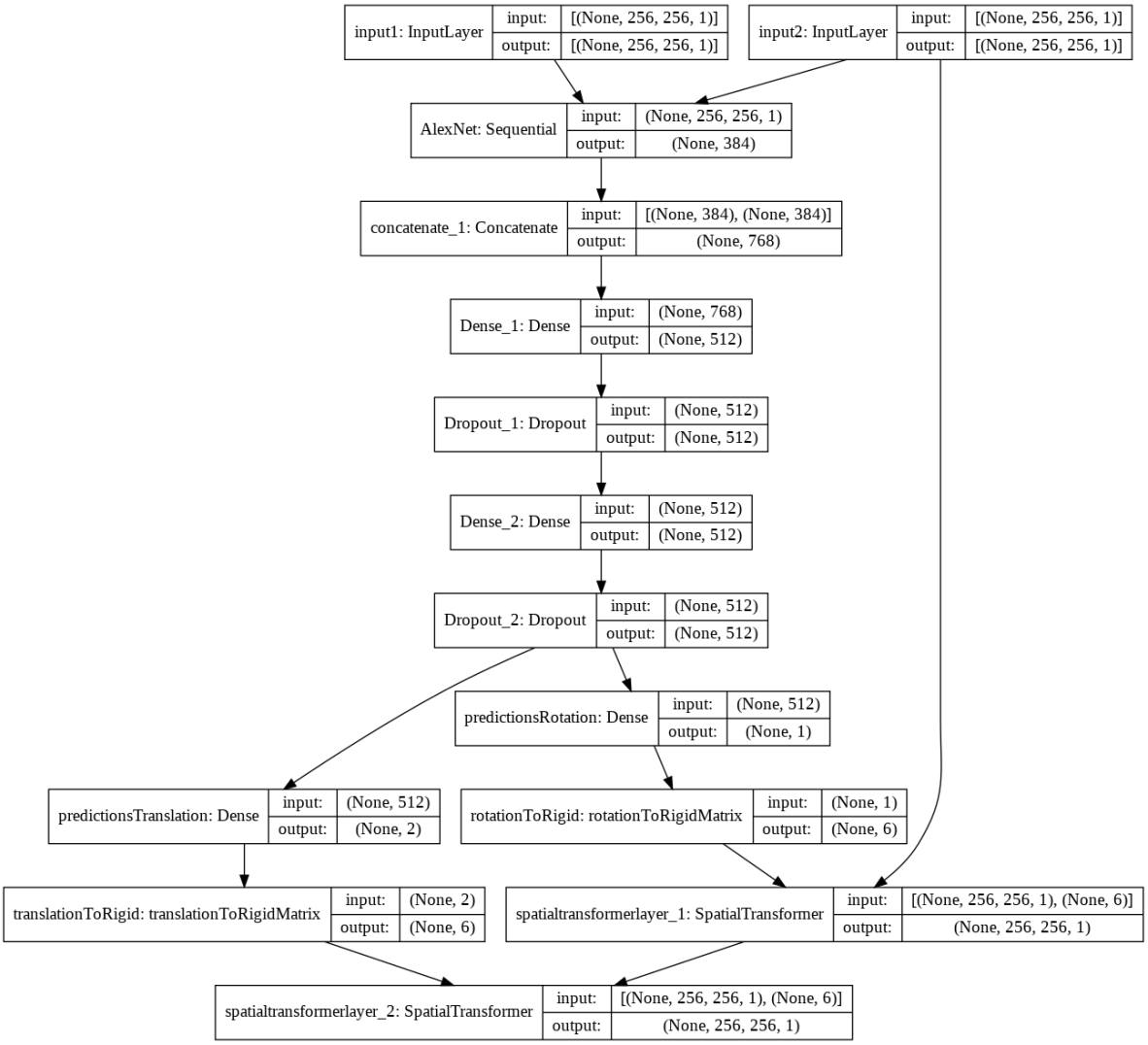


FIGURE 5.19: Modified deformable network structure.

The final network contained 4.8m parameters. It was trained on Nvidia Tesla P100 for 300 epochs. Adam optimiser was used along MSE loss for both spatial transformers and translation regression. MAE loss was used for rotation regression. The resulting loss curves from this network proved much better than previous networks (Figure 5.20):

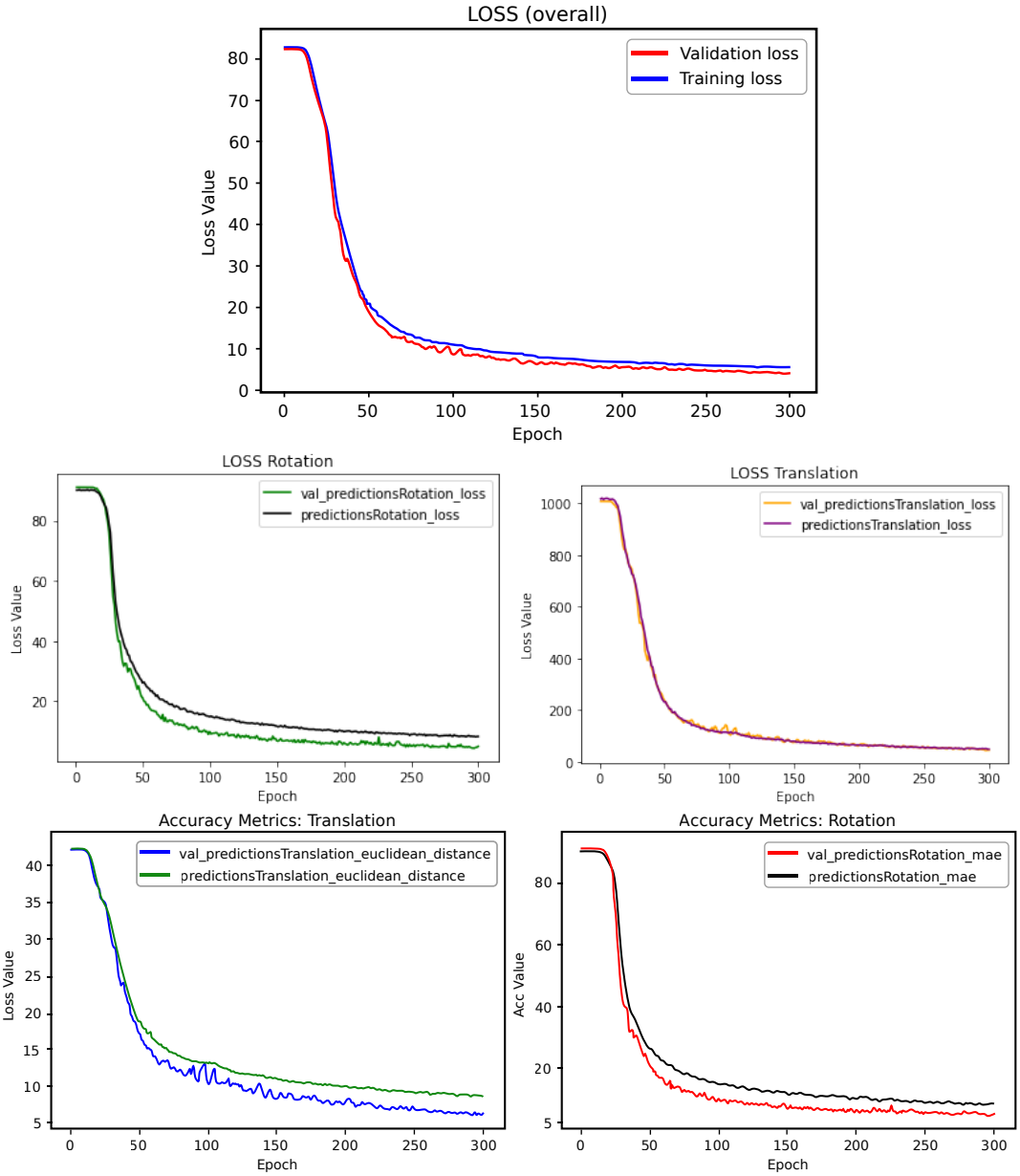


FIGURE 5.20: Loss and accuracy training curves.

Resulting predictions on the testing set were as in Table 5.4:

| Prediction | Average accuracy |
|---|---|
| Rotation MAE (yaw angle) | 4° |
| Translation (Euclidean distance) | 5.8 pixels |

TABLE 5.4: Results from using the proposed network architecture.

#### 5.4.4.2 Discussion

Instead of augmenting and shuffling all Potsdam dataset images, then splitting them into training/validation/testing sets, they were split before

augmentation, and each set was augmented. But this failed, which meant that the network was not learning to generalise to photos it had not seen before.

It is important to note that existing rigid transformation works [230, 228, 229, 226] use a single dataset. I.e. they use similar kinds of photos with identical layout but different rotations and translations to produce the training/validation/testing sets. Not only this, but the augmentations made are minimal, for example:

1. In [226] the translation in the augmented images ranged from [-5, +5] pixels, and the rotation was between [-5, +5] degrees only. The image sizes were 71x61 pixels.

2. In [229] they had a standard deviation of 10 pixels and 10 degrees i.e. 68% of training values were within this range. The images were sized at 156x300 pixels.

3. In [230] rotations were between [-15, +15] and translations between [-30, +30], with 256x256 pixels sized images.

4. The network proposed in [224] was trained for rough registration using rotations of range $\pm 30$, and for fine registration with range $\pm 5$.

This clearly showed that other existing networks were simply learning the current task only, and their generalisation capacity with other datasets was not good at all. This made it required to pursue a different direction, particularly a recent work which showed good prospects.

### 5.4.5 Randomised Generation

There is evidence from a recent work that training on randomly generated images makes the network adapt to the registration of images unseen before [264]. So a final attempt to learn the regression task was made using randomly generated images (Figure 5.21).

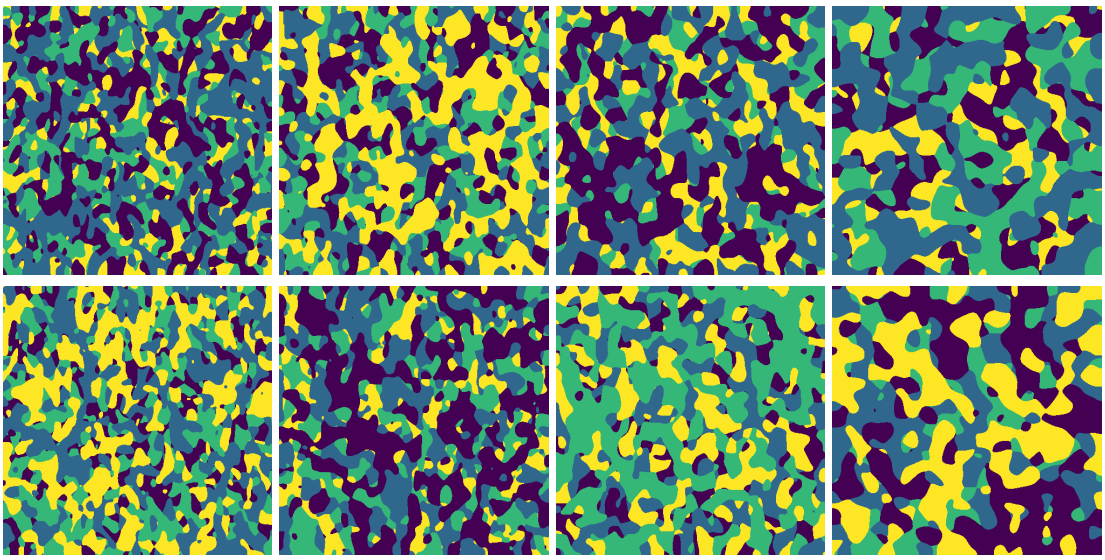

FIGURE 5.21: Samples of generated randomised images.

The images were generated like so:

1. Generate 4 Perlin Gaussian noise images $p_j(j \in 1, 2, ..., J)$. Perlin noise is a kind of randomly generated n-dimensional gradient noise [268], whose values have smooth transitions. As an example, see left of Figure 5.22. The number of generated images $J$ equals the number of categories (labels) required, for this thesis, $J = 4$.

2. Apply Gaussian blurring to each noise image. See right of Figure 5.22.

3. To get the final labelled image, calculate the *argmax* across all pixels of all generated noise images $p_j$: $r_k = \arg\max(p_j)$

4. Repeat for $k$ required random images $k \in 1, 2, ..., K$. For this work, $K =$ 8900 images were generated for training.

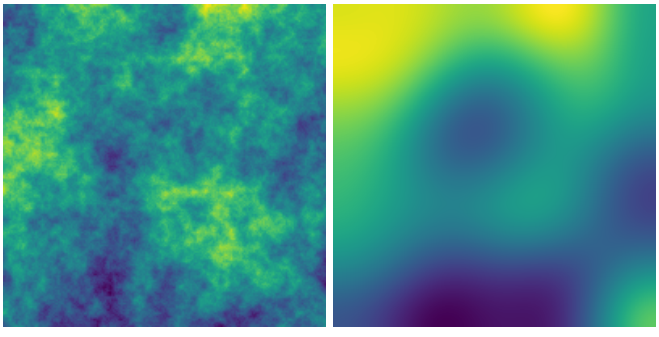

FIGURE 5.22: Perlin Gaussian noise image before smoothing (left), and after smoothing (right).

#### 5.4.5.1 Results

This procedure follows the process presented in [264, 265], except that no random deformation field is applied to the noise images. After training using the randomly generated dataset, the network could not learn to generalise well, and it was overfitting no matter how the parameters were changed (refer to Figures 5.23 & 5.24).

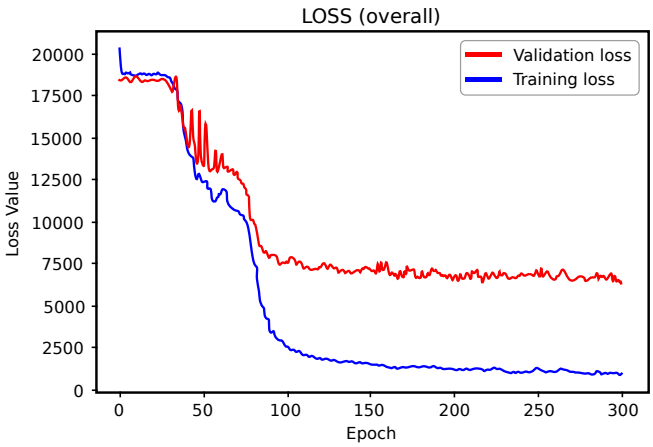


FIGURE 5.23: Resulting loss after training on the network architecture with AlexNet base.

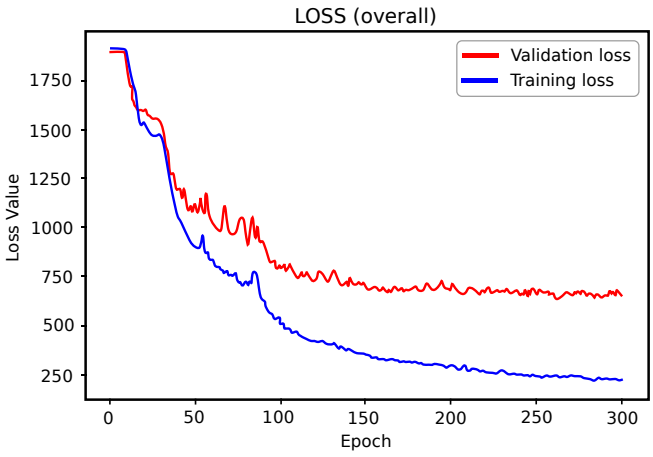


FIGURE 5.24: Resulting loss after training on the modified deformable architecture.

#### 5.4.5.2 Discussion

It has been recognised by many different papers such as [264] that CNNs in general work only on images they have seen before. Therefore the CNN does not learn how to work with unfamiliar data. This is understandable, as if the CNN learns to extract particular features belonging to a set of images, there would be no reason to think it would be able to extract those unique features in images belonging to a different datasets.

But we have a couple of reasons to postulate that this rule does not apply for aerial images, especially top-down ones, and more especially that they are segmented. Most aerial images share similar components (roads, trees, houses, streets). And they share a lot of common features, for example it is not possible to find round houses, as houses are mostly square shaped, roads extend from one end to another and are not isolated.. etc.

We have relied on this principle to produce the segmentation CNN which enabled the localisation system introduced in Chapter 4. It was assumed that it would work for registration as well as segmentation. I.e. by following the same principle, training a network to register pairs of segmented aerial photos from one dataset, should give the network enough knowledge to be able to generalise its behaviour on different datasets.

In addition, it has been proved in some works [265] that brute forcing the network to learn new features dramatically improved its generalisation ability. Nevertheless, none of these techniques were able to give the proposed network the capacity to generalise beyond training dataset. That is, they fail to register new aerial photos, not only photos belonging to the same dataset which the training set was taken from. The reason why randomised generation worked in the original paper [265] might be that the generated images in it were morphing examples. Which means that they already had big overlap, and it can be understood that the network 'nudged' edges to match each other. The case is different in the case of this thesis. As the photos have big rotation variance, an act of 'nudging' them means the possibility of falling into local minimums. Therefore the colours in the image need to be used for matching. In Appendix B a simple test was conducted to validate this. This test demonstrates how CNNs fail to recognise, or 'see' colour, perhaps due to their convolutional nature (convolutions are only sensitive at the edges, not on blank colours).

## 5.5 Conclusion

In conclusion, the search done in this chapter went through a series of stages so as to reach a sound solution. In the experiments done in the first stage, it was shown how both kinds of deep VO networks (prior and ensuing concatenation) did not work for the problem of top-down segmented registration with big rotation and translation differences. In the second stage, a different solution taken from deformable networks was proposed. The architecture of a state-of-the-art deformable network was changed, and a new semi-supervised network capable of doing the required registration task was proposed.

However, the proposed architecture failed to register novel photos not derived from the same dataset from which the training set was taken. Therefore, in the third stage, another experiment inspired from a promising research was conducted. The research proved that training a CNN with randomly generated images improved its generalisation capacity. The proposed network was trained on randomly generated images which represented the

input segmented photos. They contained 4 labels with identical colour spreading to the dataset. However the generalisation capacity was not improved.

The main conclusion to be drawn from this is that the failure to generalise registration to novel aerial images belonging to new environments other than the trained dataset, is not due to a shortcoming in the proposed network. As it can correctly predict the transformations between images as long as they are taken from the same dataset (as in Figure 5.15). But rather this failure is due to the convolutional nature of CNNs. A small experiment was made that exposes this shortcoming, it shows how CNNs fail to distinguish colour. The research made in this chapter highlights this gap in CNNs, and shows the need to address it.

# Chapter 6

# Conclusions

In this thesis the localisation of a fixed-wing UAV using a single monocular camera was studied. The main focus was on addressing the safety concerns of flying Small Unmanned Aircraft (SUA) under the weight of 20kg, using computer vision techniques. The SUA flies at a constant height up to a few hundred meters. Throughout the thesis, a computer vision localisation solution was developed.

Two visual odometry (VO) algorithms based on traditional feature detection and matching were proposed for solving the problem of localisation of a UAV. The first was based on image stitching, and the second was based on camera motion estimation, Perspective-from-n-Points (PnP). This was addressed in Chapter 3. Both algorithms were evaluated, but as it was shown, due to accumulated inaccuracies between the frames, big amounts of drift resulted at the end of the navigated paths. On two long datasets (1080m and 5300m), the angular drift was $10.3^{\circ}$ and $10.2^{\circ}$ respectively. And a Euclidean drift distance of 203m and 298m respectively.

With the aim of addressing the big drift resulting from traditional techniques, it was suggested to use existing resources of information which is especially available for fixed-wing UAVs, such as Google Earth. A segmentation CNN was proposed (Chapter 4) which was trained on an independent dataset, but was able to segment new unseen datasets, such as the navigated environment and the corresponding Google Earth map. It was shown how integrating this CNN with a particle filter, in addition to the VO module presented earlier which caused drift if used alone, offered big improvements. It enabled the localisation of the UAV on the map, which solved the large drift amounts and made it possible to do homing, which is to return back to the initial take off location.

The proposed solution could achieve localisation accuracy close to state-of-the-art algorithms, with the additional advantage of being capable of global localisation. The only disadvantage of the proposed system is that its accuracy is highly tied to the frame rate. Given the probabilistic nature of the particle filter, its accuracy can be significantly improved by having higher frame rate, which was not available in the datasets used.

Aiming to refine the system further, and given the success of using deep learning in a navigation system, an investigation about the possibility of extending the deep learning to other parts of the system was pursued. The focus was especially the motion estimation component. Many existing works

were evaluated, eventually proposing a rigid based semi-supervised registration system which could estimate the rotation and translation between two segmented snapshots (Chapter 5). Despite its limited generalisation capacity, it was able to learn and predict well but only using photographs identical to the dataset on which it was trained.

The proposed solution is cheap to implement. It requires a single camera with moderate on-board processing, which makes it applicable to consumer lightweight fixed-wing drones under 20kg. Possible applications include:

1. The proposed localisation system is cheap to implement. All it requires is a top down camera and moderate on-board processing. And it can be used to enable homing on Small Aerial Vehicles (SAVs), especially cheap drones sold off the shelf. This property increases safety of operation and it can possibly count towards demonstrating that safe flight can be performed beyond visual line of sight (BVLOS). Such demonstration is a legal requirement in the UK [8]. If it can be demonstrated that the system is safe enough, it can deregulate flying an SAV.

2. The system can be deployed on professional UAVs with state-of-the-art GNSS navigation technologies. This helps in adding an extra layer of redundancy which is crucial in every aerial navigation system. Numerous redundant localisation systems are used on civilian aeroplanes, but unfortunately, not all of them are available for SAVs. Therefore having cheap redundant localisation systems which can be cheaply implemented on SAVs is very important for reducing risks and improving safety. This was argued in detail in Chapter 1.

3. The system can also be extended to be used on other large scale GNSS denied environments specifically on other solar planets which lack satellite navigation technologies like Mars.

## 6.1 Future work

The work presented in this thesis can be carried forward and developed in many directions, for example:

1. State estimation and visual odometry are problems which have been studied well over the past two decades. They have efficient traditional mathematical and geometric solutions. However, due to the advent of deep learning techniques, the right balance and optimum integration between deep learning and traditional methods across different fields and applications is yet to be found. The research done in this thesis highlights this need, and shows how the two can be combined to achieve superior end results.

2. Localisation accuracy: It is possible to integrate an Inertial Measurement Unit (IMU) inside the UAV. The IMU measurements should be recorded alongside the captured snapshots. It is possible to merge these

measurements with the location estimate predicted by the system using a Kalman filter, in what is known as the Extended Kalman Filter (EKF) sensor fusion. Other clues which can possibly help in localising the UAV can also be integrated such as the location of the sun, which can be inferred from the building shadow directions in each of the snapshots. The application of recent neuromorphic imaging sensors on fixed-wing UAVs can also be explored.

3. Improve safety: Another path which can be pursued is to classify the safe landing regions on the map and attempt to use them as backup landing sites in case of emergency. It is sometimes important to recognise populated areas and avoid flying over them all together. This requires improving the classification CNN, and to develop path planning algorithms. One important challenge in this regard is to be able to correctly classify safe landing sites as they share the same categories as other sites.

4. CNN architecture: The fifth chapter revealed a shortcoming about CNNs and provides pointers to possible directions which can be taken. It shows a need to deeply explore how CNNs process information on a lower level, with the purpose of enabling CNNs to have better colour sensitivity on blank and untextured images (refer to Appendix B).

# Appendix A

# Matrix and Vector Properties

Some of the equations in Chapter 3 rely on the following vector and matrix properties:

## A.1 Vector Operations

### A.1.1 Cross Product

- The cross product of two vectors, is another vector which is perpendicular to both. The resulting vector has a magnitude (length) which is equal to the parallelogram formed by the two vectors.
- A cross product of a vector with itself is zero:

$$\vec{a} \times \vec{a} = 0,$$

- Cross product between two equal n-dimentional vectors is equal to:

$$\vec{a_{3\times1}} \times \vec{b_{3\times1}} = \lfloor a_x \rfloor b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}_{3\times3} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}_{3\times1},$$

  Where $\lfloor a_x \rfloor$ is called the skew symmetric matrix form of the vector $\vec{a}$. A skew symmetric matrix is a matrix which satisfies the condition:

$$A^T = -A.$$

### A.1.2 Dot Product

- The dot product of two vectors is a scalar which can be calculated as:

$$\vec{a}.\vec{b} = |a||b| \sin(\theta),$$

- A dot product of a vector with itself is one:

$$\vec{a}.\vec{a} = 1,$$

- The dot product of two perpendicular vectors is zero:

$$\vec{a}.\vec{b} = 0,$$

- Dot product of two matrices:

$$A.B = A^T B,$$

  Where $A^T$ is the transpose of $A$.

## A.2 Matrix Properties

- Transpose of two multiplied matrices is:

$$(AB)^T = B^T A^T.$$

- Matrix multiplication is associative:

$$A * B = B * A$$

- Matrix *rank* is the number of unique rows *or* columns in that matrix. I.e. it is the number of rows which are not a multiplication, addition, division or subtraction of other rows. If the rank of a matrix is equal to the number of variables in it, it becomes possible to find a unique solution. If the matrix is square, it is possible to find the rank by finding the *determinant*. A non-zero determinant means that matrix is full rank (all rows and columns are independent).

- Matrix *determinant* can be found only for square matrices:

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

# Appendix B

# CNN blindness to colour

The following experiment was done to test the colour sensitivity of the registration CNNs presented in Section 5.4.3. A greyscale chequered image was randomly generated using a range of 4 colours (representing the 4 main categories existing in the aerial datasets used in the thesis). See Figure B.1.

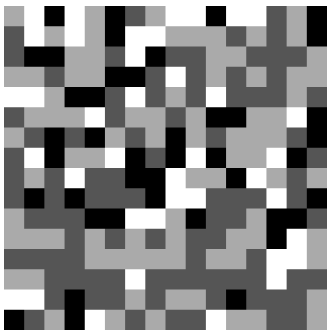

FIGURE B.1: Randomly generated image with 32x32 blocks, using 4 colours.

Each generated image was randomly cropped into two snapshots differing only in translation, and each containing 8x8 blocks. Each generated image had the resolution equivalent to input an input aerial photo to the CNN (see Figure B.2).

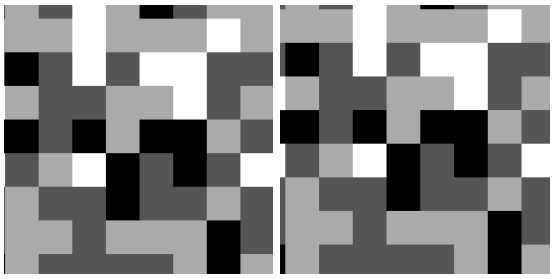

FIGURE B.2: Two 8x8 cropped snapshots from a generated image.

This operation was repeated until a whole dataset of left and right images was populated, and used to train a siamese CNN proposed in Section 5.4.3.

## B.1 Results

The experimented was first attempted with low resolution shapes (Figure B.3), then with ones containing more blocks:

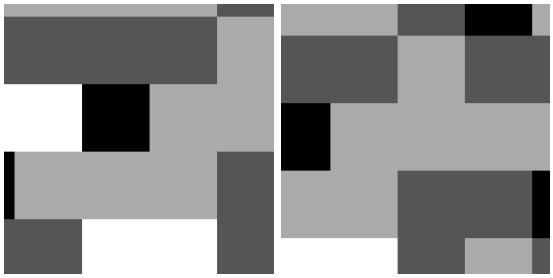

FIGURE B.3: Two 4x4 cropped snapshots from a generated image.

But no matter how much block resolution the images had, the registration CNN would not learn to map these photos to each other. A sample resulting loss curve is shown in Figure B.4.

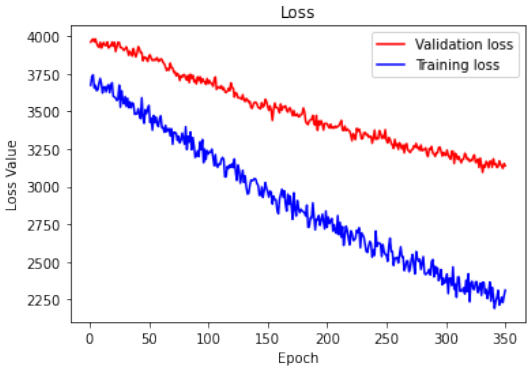


FIGURE B.4: Sample loss result of training using the checkered squares.

However, surprise came when a simple non symmetric star shape was imposed on the snapshots (Figure B.5):

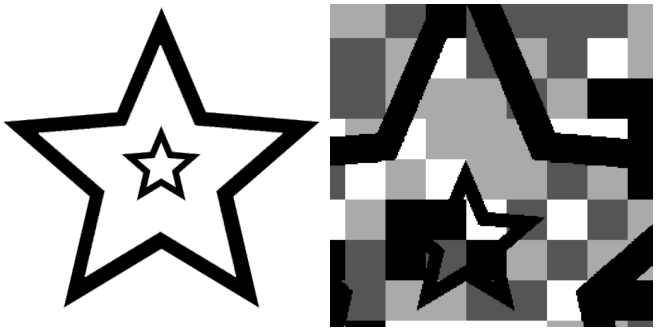

FIGURE B.5: Left: Imposed star. Right: Sample resulting frame

The network learned to register the unsymmetrical images easily as opposed to images containing symmetrical coloured blocks. The resulting loss curve is shown in Figure B.6.

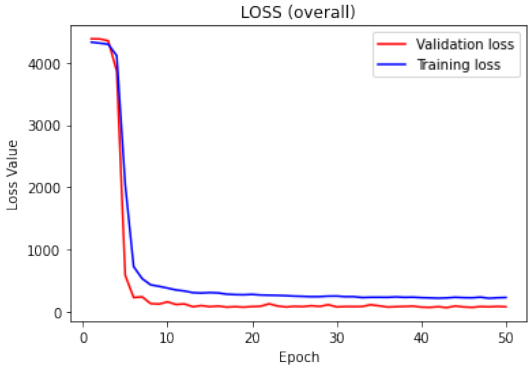


FIGURE B.6: Sample loss result of training using the checkered squares with the added star.

This experiment shows clearly how CNNs have a disadvantage when it comes to colour sensitivity. As the network only learned to register the images when the unsymmetrical shapes were added to the training images. Which lets us draw the conclusion that due to the convolutional nature of CNNs, they learn from edges despite the various colours existing in the images. This experiment sheds the light on an inherent shortcoming about CNNs. The research done in Chapter 5 shows the need to address this shortcoming.

# Bibliography


[1] T. S. Collett and M. Collett, "Memory Use In Insect Visual Navigation," *Nature Reviews Neuroscience*, vol. 3, no. 7, p. 542, 2002.

[2] "Sensefly Merlishachen Swiss Village Dataset," Accessed: 2021-08-25. [Online]. Available: https://www.sensefly.com/education/datasets/?dataset=1419

[3] "2D Semantic Labeling Contest - Potsdam Dataset," 2013, International Society for Photogrammetry and Remote Sensing, Accessed: 2021-08-25. [Online]. Available: http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

[4] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *ICML*, 2010.

[5] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training By Reducing Internal Covariate Shift," *arXiv preprint arXiv:1502.03167*, 2015.

[6] "Sensefly Willisau Swiss Gravel Quarry Dataset," Accessed: 2021-08-25. [Online]. Available: https://www.sensefly.com/education/datasets/?dataset=1421

[7] H. Goforth and S. Lucey, "GPS-Denied UAV Localization Using Pre-existing Satellite Imagery," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2974–2980.

[8] "Civil Aviation Authority: Unmanned Aircraft System Operations in UK Airspace - Guidance and Policy CAP 722," 2019, Accessed: 2020-06-25. [Online]. Available: http://publicapps.caa.co.uk/modalapplication.aspx?appid=11&mode=detail&id=415

[9] D. C. Tsouros, S. Bibi, and P. G. Sarigiannidis, "A Review On UAV-Based Applications For Precision Agriculture," *Information*, vol. 10, no. 11, p. 349, 2019.

[10] S. Lahoti, A. Lahoti, and O. Saito, "Application Of Unmanned Aerial Vehicle (UAV) For Urban Green Space Mapping In Urbanizing Indian Cities," in *Unmanned Aerial Vehicle: Applications in Agriculture and Environment*. Springer, 2020, pp. 177–188.

[11] F. Nex and F. Remondino, "UAV For 3D Mapping Applications: A Review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[12] M. Erdelj, E. Natalizio, K. R. Chowdhury, and I. F. Akyildiz, "Help From The Sky: Leveraging UAVs For Disaster Management," *IEEE Pervasive Computing*, vol. 16, no. 1, pp. 24–32, 2017.

[13] M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge, "Multipurpose UAV For Search And Rescue Operations In Mountain Avalanche Events," *Geomatics, Natural Hazards and Risk*, vol. 8, no. 1, pp. 18–33, 2017.

[14] A. Goodchild and J. Toy, "Delivery By Drone: An Evaluation Of Unmanned Aerial Vehicle Technology In Reducing CO2 Emissions In The Delivery Service Industry," *Transportation Research Part D: Transport and Environment*, vol. 61, pp. 58–67, 2018.

[15] E. Frachtenberg, "Practical Drone Delivery," *Computer*, vol. 52, no. 12, pp. 53–57, 2019.

[16] NASA, "Urban Air Mobility (UAM) Market Study," 2018, Accessed: 2021-10-01. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/uam-market-study-executive-summary-v2.pdf

[17] G. Brunner, B. Szebedy, S. Tanner, and R. Wattenhofer, "The Urban Last Mile Problem: Autonomous Drone Delivery To Your Balcony," in *2019 international conference on unmanned aircraft systems (icuas)*. IEEE, 2019, pp. 1005–1012.

[18] P. Finnegan, "2020/2021 World Civil Unmanned Aerial Systems Market Profile & Forecast," Teal Group, Tech. Rep., 2020.

[19] "Department For Transport New Technology Grants; Innovation Challenge Fund Webinar," 2017, Accessed: 2018-09-22. [Online]. Available: https://www.dft.gov.uk/innovation-grants/innovation-grants/icf/

[20] "Sensefly Photogrammetric Imaging," Accessed: 2021-08-25. [Online]. Available: https://www.sensefly.com/drone/ebee-x-fixed-wing-drone/

[21] "Wingtra," Accessed: 2018-05-25. [Online]. Available: https://www.wingtra.com/drone/

[22] O. Mian, J. Lutes, G. Lipa, J. J. Hutton, E. Gavelle, and S. Borghini, "Direct Georeferencing On Small Unmanned Aerial Platforms For Improved Reliability And Accuracy Of Mapping Without The Need For Ground Control Points," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-1/W4, pp. 397–402, 2015. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-1-W4/397/2015/

[23] K. W. Williams, "A Summary of Unmanned Aircraft Accident/Incident Data: Human Factors Implications," Federal Aviation Administration (FAA) Civil Aerospace Medical Institute (CAMI) Oklahoma City, Tech. Rep., 2004.

[24] D. A. Wiegmann and S. A. Shappell, *A Human Error Approach To Aviation Accident Analysis: The Human Factors Analysis And Classification System*. Routledge, 2017.

[25] A. Dempster, "How Vulnerable Is GPS?" *Transportation*, 2001.

[26] B. Van den Bergh and S. Pollin, "Keeping UAVs Under Control During GPS Jamming," *IEEE Systems Journal*, vol. 13, no. 2, pp. 2010–2021, 2018.

[27] M. Harris, "Military Tests that Jam and Spoof GPS Signals are an Accident Waiting to Happen," *IEEE Spectrum*, vol. 58, no. 2, pp. 22–27, 2021.

[28] E. P. Marcos, A. Konovaltsev, S. Caizzone, M. Cuntz, K. Yinusa, W. Elmarissi, and M. Meurer, "Interference And Spoofing Detection For GNSS Maritime Applications Using Direction Of Arrival And Conformal Antenna Array," in *Proceedings of the 31st International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2018)*, 2018, pp. 2907–2922.

[29] C. Palmer, "The Boeing 737 Max Saga: Automating Failure," *Engineering*, vol. 6, no. 1, pp. 2–3, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809919308641

[30] J. A. Bergstra and M. Burgess, "Candidate Software Process Flaws For The Boeing 737 Max MCAS Algorithm And Risks For A Proposed Upgrade," *arXiv preprint arXiv:2001.05690*, 2020.

[31] J. Herkert, J. Borenstein, and K. Miller, "The Boeing 737 MAX: Lessons For Engineering Ethics," *Science and engineering ethics*, vol. 26, no. 6, pp. 2957–2974, 2020.

[32] J. A. Volpe, "Vulnerability Assessment Of The Transportation Infrastructure Relying On The Global Positioning System," US National Transportation Systems Center, Tech. Rep., 2001.

[33] "International Federation for Robotics: Executive Summary World Robotics 2017 Industrial Robots," 2017, Accessed: 2019-01-28. [Online]. Available: https://ifr.org/downloads/press/Executive_Summary_WR_2017_Industrial_Robots.pdf

[34] K. Ballenger, "Unmanned Aircraft Systems - General Overview," *American Institute of Aeronautics and Astronautics*, vol. 26, 2013.

[35] "Flyability Elios Drone," Accessed: 2018-09-13. [Online]. Available: https://www.flyability.com/elios/

[36] "Createc Radiation Imaging," Accessed: 2018-05-25. [Online]. Available: https://www.createc.co.uk/challenging-problems/radiation-imaging/

[37] "Amazon Prime Air," https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011, Accessed: 2021-08-25.

[38] K. Ikeuchi and D. Miyazaki, *Digitally Archiving Cultural Objects*. Springer Science & Business Media, 2008.

[39] "KittyHawk Cora," Accessed: 2018-05-30. [Online]. Available: https://cora.aero/

[40] "Verity Studios," Accessed: 2018-06-01. [Online]. Available: https://veritystudios.com/

[41] "Responsible Robotics Report: Drones In The Service Of Society," Accessed: 2018-08-22. [Online]. Available: https://responsiblerobotics.org/2018/06/05/report-drones-in-the-service-of-society/

[42] S. Schoenung, "Overview Of UAV Status And Challenges For Remote Sensing," *Proceedings of 30th International Symposium on Remote Sensing of Environment: Information for Risk Management and Sustainable Development, November 10-14, 2003, Honolulu, Hawai'i*, pp. 652–655, 2003.

[43] D. Nistér, O. Naroditsky, and J. Bergen, "Visual Odometry For Ground Vehicle Applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3–20, 2006.

[44] D. Scaramuzza and F. Fraundorfer, "Visual Odometry Part I: The First 30 Years And Fundamentals," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[45] F. Fraundorfer and D. Scaramuzza, "Visual Odometry Part II: Matching, Robustness, Optimization, And Applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[46] O. Amidi, "An Autonomous Vision-Guided Helicopter," Ph.D. dissertation, Carnegie Mellon University, 1996.

[47] O. Amidi, T. Kanade, and K. Fujita, "A Visual Odometer For Autonomous Helicopter Flight," *Robotics and Autonomous systems*, vol. 28, no. 2-3, pp. 185–193, 1999.

[48] A. Volkova and P. W. Gibbens, "Extended Robust Feature-Based Visual Navigation System for UAVs," in *Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on*. IEEE, 2016, pp. 1–7.

[49] G. Conte and P. Doherty, "An Integrated UAV Navigation System Based On Aerial Image Matching," in *Aerospace Conference, 2008 IEEE*. IEEE, 2008, pp. 1–10.

[50] J. R. Braga, H. F. Velho, G. Conte, P. Doherty, and É. H. Shiguemori, "An Image Matching System For Autonomous UAV Navigation Based On Neural Network," in *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on*. IEEE, 2016, pp. 1–6.

[51] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization And Mapping: Part I," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[52] T. Bailey and H. Durrant-Whyte, "Simultaneous Localization And Mapping (SLAM): Part II," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[53] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, And Future Of Simultaneous Localization And Mapping: Toward The Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[54] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[55] J. Zhang and S. Singh, "Enabling Aggressive Motion Estimation At Low-Drift And Accurate Mapping In Real-Time," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 5051–5058.

[56] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," in *European Conference On Computer Vision*. Springer, 2014, pp. 834–849.

[57] D. Gálvez-López and J. D. Tardos, "Bags Of Binary Words For Fast Place Recognition In Image Sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[58] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile And Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[59] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual Simultaneous Localization And Mapping: A Survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[60] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-SLAM–Based Navigation For Autonomous Micro Helicopters In GPS-Denied Environments," *Journal of Field Robotics*, vol. 28, no. 6, pp. 854–874, 2011.

[61] J. Engel, J. Sturm, and D. Cremers, "Scale-Aware Navigation Of A Low-Cost Quadrocopter With A Monocular Camera," *Robotics and Autonomous Systems*, vol. 62, no. 11, pp. 1646–1656, 2014.

[62] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[63] A. J. Davison and D. W. Murray, "Simultaneous Localization And Map-Building Using Active Vision," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 7, pp. 865–880, 2002.

[64] M. Montemerlo and S. Thrun, *FastSLAM: A Scalable Method For The Simultaneous Localization And Mapping Problem In Robotics*. Springer, 2007, vol. 27.

[65] M. Basiri, F. Schill, P. U. Lima, and D. Floreano, "Robust Acoustic Source Localization Of Emergency Signals From Micro Air Vehicles," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4737–4742.

[66] H. Strasdat, J. M. Montiel, and A. J. Davison, "Visual SLAM: Why Filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[67] G. Klein and D. Murray, "Parallel Tracking And Mapping For Small AR Workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE, 2007, pp. 225–234.

[68] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart, "Onboard IMU And Monocular Vision Based Control For MAVs In Unknown In-And Outdoor Environments," in *Robotics and automation (ICRA), 2011 IEEE international Conference on*. IEEE, 2011, pp. 3056–3063.

[69] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision Based MAV Navigation In Unknown And Unstructured Environments," in *Robotics and automation (ICRA), 2010 IEEE international Conference on*. IEEE, 2010, pp. 21–28.

[70] R. Mur-Artal and J. D. Tardós, "Orb-SLAM2: An Open-Source SLAM System For Monocular, Stereo, And RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[71] Z. Zhang, C. Forster, and D. Scaramuzza, "Active Exposure Control For Robust Visual Odometry in HDR Environments," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3894–3901.

[72] D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip *et al.*, "Vision-Controlled Micro Flying Robots: From System Design To Autonomous Navigation And Mapping In GPS-Denied Environments," *IEEE Robotics & Automation Magazine*, vol. 21, no. 3, pp. 26–40, 2014.

[73] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? Combining Events, Images, And IMU For Robust Visual SLAM In HDR And High-Speed Scenarios," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 994–1001, 2018.

[74] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-Direct Monocular Visual Odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 15–22.

[75] M. Pizzoli, C. Forster, and D. Scaramuzza, "REMODE: Probabilistic, Monocular Dense Reconstruction In Real Time," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2609–2616.

[76] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous On-Board Monocular-Vision-Based Elevation Mapping Applied To Autonomous Landing Of Micro Aerial Vehicles," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 111–118.

[77] "Pix4D Photogrammetry Software," Accessed: 2018-09-22. [Online]. Available: https://pix4d.com/

[78] D. Nistér, O. Naroditsky, and J. Bergen, "Visual Odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1. Ieee, 2004, pp. I–I.

[79] D. Törnqvist, T. B. Schön, R. Karlsson, and F. Gustafsson, "Particle Filter SLAM With High Dimensional Vehicle Model," *Journal of Intelligent and Robotic Systems*, vol. 55, no. 4-5, pp. 249–266, 2009.

[80] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, Vision-Based Flight And Live Dense 3D Mapping With A Quadrotor Micro Aerial Vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[81] D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip *et al.*, "Vision-Controlled Micro Flying Robots: From System Design To Autonomous Navigation And Mapping In GPS-Denied Environments," *IEEE Robotics & Automation Magazine*, vol. 21, no. 3, pp. 26–40, 2014.

[82] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library For Visual, Visual-Inertial, and Multimap SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[83] D. Scaramuzza, "1-Point-RANSAC Structure From Motion For Vehicle-Mounted Cameras By Exploiting Non-Holonomic Constraints," *International journal of Computer Vision*, vol. 95, no. 1, pp. 74–85, 2011.

[84] W. Rawat and Z. Wang, "Deep Convolutional Neural Networks For Image Classification: A Comprehensive Review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[85] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew, "A Review Of Semantic Segmentation Using Deep Neural Networks," *International journal of multimedia information retrieval*, vol. 7, no. 2, pp. 87–93, 2018.

[86] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan *et al.*, "Natural TTS Synthesis By Conditioning WaveNet On Mel Spectrogram Predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4779–4783.

[87] Z. Zhang, J. Geiger, J. Pohjalainen, A. E.-D. Mousa, W. Jin, and B. Schuller, "Deep Learning For Environmentally Robust Speech Recognition: An Overview Of Recent Developments," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 5, pp. 1–28, 2018.

[88] G. Guo and N. Zhang, "A Survey On Deep Learning Based Face Recognition," *Computer vision and image understanding*, vol. 189, p. 102805, 2019.

[89] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "FairMOT: On The Fairness Of Detection And Re-Identification In Multiple Object Tracking," *arXiv preprint arXiv:2004.01888*, 2020.

[90] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[91] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations By Back-Propagating Errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[92] A. Halevy, P. Norvig, and F. Pereira, "The Unreasonable Effectiveness Of Data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

[93] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering The Game Of Go Without Human Knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[94] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving Neural Networks By Preventing Co-Adaptation Of Feature Detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[95] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *2009 IEEE Conference on Computer Vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[96] D. Held, S. Thrun, and S. Savarese, "Learning To Track At 100 FPS With Deep Regression Networks," in *European conference on computer vision*. Springer, 2016, pp. 749–765.

[97] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied To Handwritten Zip Code Recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied To Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[99] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[100] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification With Deep Convolutional Neural Networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[101] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2015, pp. 1–9.

[102] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[103] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning For Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2016, pp. 770–778.

[104] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks For Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[105] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetv2: Inverted Residuals And Linear Bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2018, pp. 4510–4520.

[106] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks For Biomedical Image Segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[107] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[108] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *2009 IEEE Conference on Computer Vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[109] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic Object Classes In Video: A High-Definition Ground Truth Database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[110] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision Meets Robotics: The KITTI Dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[111] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks For Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2015, pp. 3431–3440.

[112] H. Noh, S. Hong, and B. Han, "Learning Deconvolution Network For Semantic Segmentation," in *Proceedings of the IEEE international Conference on Computer Vision*, 2015, pp. 1520–1528.

[113] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A Deep Convolutional Encoder-Decoder Architecture For Image Segmentation," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[114] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, And Fully Connected CRFs," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[115] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder With Atrous Separable Convolution For Semantic Image Segmentation," in *Proceedings of the European Conference On Computer Vision (ECCV)*, 2018, pp. 801–818.

[116] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies For Accurate Object Detection And Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2014, pp. 580–587.

[117] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE international Conference on Computer Vision*, 2017, pp. 2961–2969.

[118] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international Conference on Computer Vision*, 2015, pp. 1440–1448.

[119] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[120] Y. Chen, Y. Tian, and M. He, "Monocular Human Pose Estimation: A Survey Of Deep Learning-Based Methods," *Computer Vision and Image Understanding*, vol. 192, p. 102897, 2020.

[121] K. Tang, Z. Li, L. Tian, L. Wang, and Y. Zhu, "ADMIR–Affine And Deformable Medical Image Registration For Drug-Addicted Brain Images," *IEEE Access*, vol. 8, pp. 70 960–70 968, 2020.

[122] S. Lathuilière, P. Mesejo, X. Alameda-Pineda, and R. Horaud, "A Comprehensive Analysis Of Deep Regression," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 9, pp. 2065–2081, 2019.

[123] S. Zagoruyko and N. Komodakis, "Learning To Compare Image Patches Via Convolutional Neural Networks," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2015, pp. 4353–4361.

[124] X. Wang and H. Zhang, "Deep Monocular Visual Odometry For Ground Vehicle," *IEEE Access*, vol. 8, pp. 175 220–175 229, 2020.

[125] N. Radwan, A. Valada, and W. Burgard, "Vlocnet++: Deep Multitask Learning For Semantic Visual Localization And Odometry," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4407–4414, 2018.

[126] X. Liu, L. Song, S. Liu, and Y. Zhang, "A Review Of Deep-Learning-Based Medical Image Segmentation Methods," *Sustainability*, vol. 13, no. 3, p. 1224, 2021.

[127] I. Kotaridis and M. Lazaridou, "Remote Sensing Image Segmentation Advances: A Meta-Analysis," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 173, pp. 309–322, 2021.

[128] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[129] S. Lathuilière, P. Mesejo, X. Alameda-Pineda, and R. Horaud, "A Comprehensive Analysis Of Deep Regression," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 9, pp. 2065–2081, 2019.

[130] S. Minaee, P. Luo, Z. Lin, and K. Bowyer, "Going Deeper Into Face Detection: A Survey," *arXiv preprint arXiv:2103.14983*, 2021.

[131] C. Zheng, W. Wu, T. Yang, S. Zhu, C. Chen, R. Liu, J. Shen, N. Kehtarnavaz, and M. Shah, "Deep Learning-Based Human Pose Estimation: A Survey," *arXiv preprint arXiv:2012.13392*, 2020.

[132] Y. Fu, Y. Lei, T. Wang, W. J. Curran, T. Liu, and X. Yang, "Deep Learning In Medical Image Registration: A Review," *Physics in Medicine & Biology*, vol. 65, no. 20, p. 20TR01, 2020.

[133] M. O. Franz and H. A. Mallot, "Biomimetic Robot Navigation," *Robotics and autonomous Systems*, vol. 30, no. 1-2, pp. 133–153, 2000.

[134] R. Wehner, B. Michel, and P. Antonsen, "Visual Navigation In Insects: Coupling Of Egocentric And Geocentric Information," *Journal of Experimental Biology*, vol. 199, no. 1, pp. 129–140, 1996.

[135] A. Wystrach, M. Mangan, and B. Webb, "Optimal Cue Integration In Ants," *Proc. R. Soc. B*, vol. 282, no. 1816, p. 20151484, 2015.

[136] R. Wehner, "The Architecture Of The Desert Ants Navigational Toolkit (Hymenoptera: Formicidae)," *Myrmecol News*, vol. 12, no. September, pp. 85–96, 2009.

[137] M. Müller and R. Wehner, "Wind And Sky As Compass Cues In Desert Ant Navigation," *Naturwissenschaften*, vol. 94, no. 7, pp. 589–594, 2007.

[138] K. Steck, B. S. Hansson, and M. Knaden, "Smells Like Home: Desert Ants, Cataglyphis Fortis, Use Olfactory Landmarks To Pinpoint The Nest," *Frontiers in Zoology*, vol. 6, no. 1, p. 5, 2009.

[139] P. Graham and K. Cheng, "Ants Use The Panoramic Skyline As A Visual Cue During Navigation," *Current Biology*, vol. 19, no. 20, pp. R935–R937, 2009.

[140] M. Müller and R. Wehner, "Path Integration In Desert Ants, Cataglyphis Fortis," *Proceedings of the National Academy of Sciences*, vol. 85, no. 14, pp. 5287–5290, 1988.

[141] A. Philippides, A. Dewar, A. Wystrach, M. Mangan, and P. Graham, "How Active Vision Facilitates Familiarity-Based Homing," in *Conference on Biomimetic and Biohybrid Systems*. Springer, 2013, pp. 427–430.

[142] T. Stone, M. Mangan, A. Wystrach, and B. Webb, "Rotation Invariant Visual Processing For Spatial Memory In Insects," *Interface Focus*, vol. 8, no. 4, p. 20180010, 2018.

[143] D. Floreano, J.-C. Zufferey, M. V. Srinivasan, and C. Ellington, *Flying Insects And Robots*. Springer, 2009, ch. 7: Visual homing in insects and robots, pp. 87–98.

[144] J. F. Cheeseman, C. D. Millar, U. Greggers, K. Lehmann, M. D. Pawley, C. R. Gallistel, G. R. Warman, and R. Menzel, "Way-Finding In Displaced Clock-Shifted Bees Proves Bees Use A Cognitive Map," *Proceedings of the National Academy of Sciences*, p. 201408039, 2014.

[145] A. Cheung, M. Collett, T. S. Collett, A. Dewar, F. Dyer, P. Graham, M. Mangan, A. Narendra, A. Philippides, W. Stürzl *et al.*, "Still No Convincing Evidence For Cognitive Map Use By Honeybees," *Proceedings of the National Academy of Sciences*, vol. 111, no. 42, pp. E4396–E4397, 2014.

[146] D. Floreano, J.-C. Zufferey, M. V. Srinivasan, and C. Ellington, *Flying Insects and Robots*. Springer, 2009.

[147] F. Colonnier, A. Manecy, R. Juston, H. Mallot, R. Leitel, D. Floreano, and S. Viollet, "A Small-Scale Hyperacute Compound Eye Featuring Active Eye Tremor: Application To Visual Stabilization, Target Tracking, And Short-Range Odometry," *Bioinspiration & biomimetics*, vol. 10, no. 2, p. 026002, 2015.

[148] A. Vardy and R. Moller, "Biologically Plausible Visual Homing Methods Based On Optical Flow Techniques," *Connection Science*, vol. 17, no. 1-2, pp. 47–89, 2005.

[149] A. Briod, J.-C. Zufferey, and D. Floreano, "Optic-Flow Based Control Of A 46g Quadrotor," in *Workshop on Vision-based Closed-Loop Control and Navigation of Micro Helicopters in GPS-denied Environments, IROS 2013*, no. EPFL-CONF-189879, 2013.

[150] C. M. Sabo, A. Cope, K. Gurney, E. Vasilaki, and J. Marshall, "Bio-Inspired Visual Navigation For A Quadcopter Using Optic Flow," in *AIAA Infotech@ Aerospace*, 2016, p. 0404.

[151] A. J. Cope, C. Sabo, K. Gurney, E. Vasilaki, and J. A. Marshall, "A Model For An Angular Velocity-Tuned Motion Detector Accounting For Deviations In The Corridor-Centering Response Of The Bee," *PLoS computational biology*, vol. 12, no. 5, p. e1004887, 2016.

[152] A. J. Cope, C. Sabo, E. Vasilaki, A. B. Barron, and J. A. Marshall, "A Computational Model Of The Integration Of Landmarks And Motion In The Insect Central Complex," *PloS one*, vol. 12, no. 2, p. e0172325, 2017.

[153] W. Hübner and H. A. Mallot, "Metric Embedding Of View-Graphs," *Autonomous Robots*, vol. 23, no. 3, pp. 183–196, 2007.

[154] D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, and R. Wehner, "A Mobile Robot Employing Insect Strategies For Navigation," *Robotics and Autonomous systems*, vol. 30, no. 1-2, pp. 39–64, 2000.

[155] D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, and M. Milford, "OpenRatSLAM: An Open Source Brain-Based SLAM System," *Autonomous Robots*, vol. 34, no. 3, pp. 149–176, 2013.

[156] M. J. Milford and G. F. Wyeth, "Mapping A Suburb With A Single Camera Using A Biologically Inspired SLAM System," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1038–1053, 2008.

[157] ——, "SeqSLAM: Visual Route-Based Navigation For Sunny Summer Days And Stormy Winter Nights," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1643–1649.

[158] E. Pepperell, P. I. Corke, and M. J. Milford, "All-Environment Visual Place Recognition With SMART," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1612–1618.

[159] D. D. Gaffin, A. Dewar, P. Graham, and A. Philippides, "Insect-Inspired Navigation Algorithm For An Aerial Agent Using Satellite Imagery," *PloS one*, vol. 10, no. 4, p. e0122077, 2015.

[160] M. Müller, S. Lupashin, and R. D'Andrea, "Quadrocopter Ball Juggling," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 5113–5120.

[161] M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching Using Invariant Features," *International journal of Computer Vision*, vol. 74, no. 1, pp. 59–73, 2007.

[162] M. Leordeanu and M. Hebert, "A Spectral Technique For Correspondence Problems Using Pairwise Constraints," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 2. IEEE, 2005, pp. 1482–1489.

[163] S. Park and S.-K. Park, "Spectral Scan Matching And Its Application To Global Localization For Mobile Robots," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1361–1366.

[164] S. Park, S.-K. Park, and M. Hebert, "Fast And Scalable Approximate Spectral Matching For Higher Order Graph Matching," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 36, no. 3, pp. 479–492, 2014.

[165] J. Zeil, "Visual Homing: An Insect Perspective," *Current opinion in neurobiology*, vol. 22, no. 2, pp. 285–293, 2012.

[166] D. Ortin and J. M. M. Montiel, "Indoor Robot Motion Based On Monocular Images," *Robotica*, vol. 19, no. 3, pp. 331–342, 2001.

[167] Y. Cheng, M. Maimone, and L. Matthies, "Visual Odometry On The Mars Exploration Rovers," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 1. IEEE, 2005, pp. 903–910.

[168] F. Andert, N. Ammann, J. Püschel, and J. Dittrich, "On The Safe Navigation Problem For Unmanned Aircraft: Visual Odometry And Alignment Optimizations For UAV Positioning," in *2014 International conference on unmanned aircraft systems (ICUAS)*. IEEE, 2014, pp. 734–743.

[169] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning High-Speed Flight In The Wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[170] D. Scaramuzza, F. Fraundorfer, and R. Siegwart, "Real-Time Monocular Visual Odometry For On-Road Vehicles With 1-Point RANSAC," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. Ieee, 2009, pp. 4293–4299.

[171] "Wingtra Datasets," Accessed: 2018-08-22. [Online]. Available: https://wingtra.com/downloads/

[172] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory And Practice*. Princeton university press, 2012.

[173] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm For Model Fitting With Applications To Image Analysis And Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[174] Z. Zhang, R. Deriche, O. Faugeras, and Q.-T. Luong, "A Robust Technique For Matching Two Uncalibrated Images Through The Recovery Of The Unknown Epipolar Geometry," *Artificial intelligence*, vol. 78, no. 1-2, pp. 87–119, 1995.

[175] H. C. Longuet-Higgins, "A Computer Algorithm For Reconstructing A Scene From Two Projections," *Nature*, vol. 293, no. 5828, p. 133, 1981.

[176] R. Hartley and A. Zisserman, *Multiple View Geometry In Computer Vision*. Cambridge university press, 2003.

[177] R. Szeliski, *Computer Vision: Algorithms And Applications*. Springer Science & Business Media, 2010.

[178] D. Nistér, "An Efficient Solution To The Five-Point Relative Pose Problem," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[179] C. Harris, M. Stephens *et al.*, "A Combined Corner And Edge Detector," in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[180] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *European Conference On Computer Vision*. Springer, 2006, pp. 404–417.

[181] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An Efficient Alternative To SIFT Or SURF," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[182] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary Robust Invariant Scalable Keypoints," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2548–2555.

[183] E. Rosten and T. Drummond, "Machine Learning For High-Speed Corner Detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[184] Z. Zheng, Y. Ma, H. Zheng, J. Ju, and M. Lin, "UGC: Real-Time, Ultra-Robust Feature Correspondence Via Unilateral Grid-Bbased Clustering," *IEEE Access*, vol. 6, pp. 55 501–55 508, 2018.

[185] J. Bian, W.-Y. Lin, Y. Matsushita, S.-K. Yeung, T.-D. Nguyen, and M.-M. Cheng, "GMS: Grid-Based Motion Statistics For Fast, Ultra-Robust Feature Correspondence," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4181–4190.

[186] P. H. Torr and A. Zisserman, "MLESAC: A New Robust Estimator With Application To Estimating Image Geometry," *Computer Vision and image understanding*, vol. 78, no. 1, pp. 138–156, 2000.

[187] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization For Mobile Robots," *Artificial intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[188] P. Martínez-Carricondo, F. Agüera-Vega, F. Carvajal-Ramírez, F.-J. Mesas-Carrascosa, A. García-Ferrer, and F.-J. Pérez-Porras, "Assessment Of UAV-Photogrammetric Mapping Accuracy Based On Variation Of Ground Control Points," *International journal of applied earth observation and geoinformation*, vol. 72, pp. 1–10, 2018.

[189] F. Agüera-Vega, F. Carvajal-Ramírez, and P. Martínez-Carricondo, "Assessment Of Photogrammetric Mapping Accuracy Based On Variation Ground Control Points Number Using Unmanned Aerial Vehicle," *Measurement*, vol. 98, pp. 221–227, 2017.

[190] A. Nassar, K. Amer, R. ElHakim, and M. ElHelw, "A Deep CNN-Based Framework For Enhanced Aerial Imagery Registration With Applications To UAV Geolocalization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1513–1523.

[191] D. Costea and M. Leordeanu, "Aerial Image Geolocalization From Recognition And Matching Of Roads And Intersections," *arXiv preprint arXiv:1605.08323*, 2016.

[192] M. Mantelli, D. Pittol, R. Neuland, A. Ribacki, R. Maffei, V. Jorge, E. Prestes, and M. Kolberg, "A Novel Measurement Model Based On abBRIEF For Global Localization Of A UAV Over Satellite Images," *Robotics and Autonomous Systems*, vol. 112, pp. 304–319, 2019.

[193] A. Cesetti, E. Frontoni, A. Mancini, A. Ascani, P. Zingaretti, and S. Longhi, "A Visual Global Positioning System For Unmanned Aerial Vehicles Used In Photogrammetric Applications," *Journal Of Intelligent & Robotic Systems*, vol. 61, no. 1-4, pp. 157–168, 2011.

[194] S. H. Choi and C. G. Park, "Robust Aerial Scene-Matching Algorithm Based On Relative Velocity Model," *Robotics and Autonomous Systems*, vol. 124, p. 103372, 2020.

[195] B. Patel, T. D. Barfoot, and A. P. Schoellig, "Visual Localization With Google Earth Images For Robust Global Pose Estimation Of UAVs," *IEEE Robotics and Automation Letters*, 2020.

[196] A. Yol, B. Delabarre, A. Dame, J.-E. Dartois, and E. Marchand, "Vision-Based Absolute Localization For Unmanned Aerial Vehicles," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 3429–3434.

[197] M. Shan, F. Wang, F. Lin, Z. Gao, Y. Z. Tang, and B. M. Chen, "Google Map Aided Visual Navigation For UAVs In GPS-Denied Environment," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2015, pp. 114–119.

[198] F. Lindsten, J. Callmer, H. Ohlsson, D. Törnqvist, T. B. Schön, and F. Gustafsson, "Geo-Referencing For UAV Navigation Using Environmental Classification," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 1420–1425.

[199] L. Mou, Y. Hua, and X. X. Zhu, "Relation Matters: Relational Context-Aware Fully Convolutional Network for Semantic Segmentation of High-Resolution Aerial Images," *IEEE Transactions on Geoscience and Remote Sensing*, 2020.

[200] G. Conte and P. Doherty, "Vision-Based Unmanned Aerial Vehicle Navigation Using Geo-Referenced Information," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, pp. 1–18, 2009.

[201] T. Wang, Y. Zhao, J. Wang, A. K. Somani, and C. Sun, "Attention-Based Road Registration For GPS-Denied UAS Navigation," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 4, pp. 1788–1800, 2020.

[202] V. Badrinarayanan, A. Handa, and R. Cipolla, "Segnet: A Deep Convolutional Encoder-Decoder Architecture For Robust Semantic Pixel-Wise Labelling," *arXiv preprint arXiv:1505.07293*, 2015.

[203] W. Wang, N. Yang, Y. Zhang, F. Wang, T. Cao, and P. Eklund, "A Review Of Road Extraction From Remote Sensing Images," *Journal of traffic and transportation engineering (english edition)*, vol. 3, no. 3, pp. 271–282, 2016.

[204] V. Iglovikov, S. Mushinskiy, and V. Osin, "Satellite Imagery Feature Detection Using Deep Convolutional Neural Network: A Kaggle Competition," *arXiv preprint arXiv:1706.06169*, 2017.

[205] V. Iglovikov and A. Shvets, "TernausNet: U-Net With VGG11 Encoder Pre-Trained On ImageNet For Image Segmentation," *arXiv preprint arXiv:1801.05746*, 2018.

[206] I. Demir, K. Koperski, D. Lindenbaum, G. Pang, J. Huang, S. Basu, F. Hughes, D. Tuia, and R. Raska, "DeepGlobe 2018: A Challenge To Parse The Earth Through Satellite Images," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2018, pp. 172–17 209.

[207] J. Sherrah, "Fully Convolutional Networks For Dense Semantic Labelling Of High-Resolution Aerial Imagery," *arXiv preprint arXiv:1606.02585*, 2016.

[208] J. Geng, H. Wang, J. Fan, and X. Ma, "Deep Supervised And Contractive Neural Network For SAR Image Classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 4, pp. 2442–2459, 2017.

[209] M. Volpi and D. Tuia, "Dense Semantic Labeling Of Subdecimeter Resolution Images With Convolutional Neural Networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 881–893, 2016.

[210] G. Chen, X. Zhang, Q. Wang, F. Dai, Y. Gong, and K. Zhu, "Symmetrical Dense-Shortcut Deep Fully Convolutional Networks For Semantic Segmentation Of Very-High-Resolution Remote Sensing Images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 5, pp. 1633–1644, 2018.

[211] S. Paisitkriangkrai, J. Sherrah, P. Janney, and A. Van Den Hengel, "Semantic Labeling Of Aerial And Satellite Imagery," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 7, pp. 2868–2881, 2016.

[212] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-To-Image Translation With Conditional Adversarial Networks," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2017, pp. 1125–1134.

[213] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks," in *Proceedings of the IEEE international Conference on Computer Vision*, 2017, pp. 2223–2232.

[214] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can Semantic Labeling Methods Generalize To Any City? The Inria Aerial Image Labeling Benchmark," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2017, pp. 3226–3229.

[215] D. Raposo, A. Santoro, D. Barrett, R. Pascanu, T. Lillicrap, and P. Battaglia, "Discovering Objects And Their Relations From Entangled Scene Representations," *arXiv preprint arXiv:1702.05068*, 2017.

[216] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A Simple Neural Network Module For Relational Reasoning," in *Advances in neural information processing systems*, 2017, pp. 4967–4976.

[217] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond Bags Of Features: Spatial Pyramid Matching For Recognizing Natural Scene Categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 2169–2178.

[218] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling In Deep Convolutional Networks For Visual Recognition," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[219] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid Scene Parsing Network," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2017, pp. 2881–2890.

[220] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel Approach To Nonlinear/Non-Gaussian Bayesian State Estimation," in *IEE proceedings F (radar and signal processing)*, vol. 140, no. 2. IET, 1993, pp. 107–113.

[221] H. Li, H.-B. Duan, and X.-Y. Zhang, "A Novel Image Template Matching Based On Particle Filtering Optimization," *Pattern Recognition Letters*, vol. 31, no. 13, pp. 1825–1832, 2010.

[222] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm For Finding Best Matches In Logarithmic Expected Time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[223] H. Zhan, C. S. Weerasekera, J.-W. Bian, and I. Reid, "Visual Odometry Revisited: What Should Be Learnt?" in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4203–4210.

[224] R. Liao, S. Miao, P. de Tournemire, S. Grbic, A. Kamen, T. Mansi, and D. Comaniciu, "An Artificial Agent For Robust Image Registration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[225] S. S. M. Salehi, S. Khan, D. Erdogmus, and A. Gholipour, "Real-Time Deep Pose Estimation With Geodesic Loss For Image-to-Template Rigid Registration," *IEEE transactions on medical imaging*, vol. 38, no. 2, pp. 470–481, 2018.

[226] Y. Xia, Y. Li, L. Xun, Q. Yan, and D. Zhang, "A Convolutional Neural Network Cascade For Plantar Pressure Images Registration," *Gait & posture*, vol. 68, pp. 403–408, 2019.

[227] M. C. Lee, O. Oktay, A. Schuh, M. Schaap, and B. Glocker, "Image-And-Spatial Transformer Networks For Structure-Guided Image Registration," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 337–345.

[228] S. Miao, Z. J. Wang, Y. Zheng, and R. Liao, "Real-Time 2D/3D Registration Via CNN Regression," in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 2016, pp. 1430–1434.

[229] S. Miao, Z. J. Wang, and R. Liao, "A CNN Regression Approach For Real-Time 2D/3D Registration," *IEEE Transactions On Medical Imaging*, vol. 35, no. 5, pp. 1352–1363, 2016.

[230] J. M. Sloan, K. A. Goatman, and J. P. Siebert, "Learning Rigid Image Registration-Utilizing Convolutional Neural Networks For Medical Image Registration," 2018.

[231] B. D. de Vos, F. F. Berendsen, M. A. Viergever, H. Sokooti, M. Staring, and I. Išgum, "A Deep Learning Framework For Unsupervised Affine And Deformable Image Registration," *Medical image analysis*, vol. 52, pp. 128–143, 2019.

[232] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular Visual Odometry Through Unsupervised Deep Learning," in *2018 IEEE international Conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 7286–7291.

[233] L. Han, Y. Lin, G. Du, and S. Lian, "DeepVIO: Self-Supervised Deep Learning Of Monocular Visual Inertial Odometry Using 3D Geometric Constraints," *arXiv preprint arXiv:1906.11435*, 2019.

[234] V. Mohanty, S. Agrawal, S. Datta, A. Ghosh, V. D. Sharma, and D. Chakravarty, "Deepvo: A Deep Learning Approach For Monocular Visual Odometry," *arXiv preprint arXiv:1611.06069*, 2016.

[235] G. Balakrishnan, A. Zhao, M. R. Sabuncu, J. Guttag, and A. V. Dalca, "VoxelMorph: A Learning Framework For Deformable Medical Image Registration," *IEEE Transactions On Medical Imaging*, vol. 38, no. 8, pp. 1788–1800, 2019.

[236] ——, "An Unsupervised Learning Model For Deformable Medical Image Registration," in *Proceedings of the IEEE Conference on Computer Vision and pattern recognition*, 2018, pp. 9252–9260.

[237] B. D. de Vos, F. F. Berendsen, M. A. Viergever, M. Staring, and I. Išgum, "End-To-End Unsupervised Deformable Image Registration With A Convolutional Neural Network," in *Deep learning in medical image analysis and multimodal learning for clinical decision support*. Springer, 2017, pp. 204–212.

[238] A. Valada, N. Radwan, and W. Burgard, "Deep Auxiliary Learning For Visual Localization And Odometry," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 6939–6946.

[239] T. Pandey, D. Pena, J. Byrne, and D. Moloney, "Leveraging Deep Learning For Visual Odometry Using Optical Flow," *Sensors*, vol. 21, no. 4, p. 1313, 2021.

[240] H. Zhan, R. Garg, C. Saroj Weerasekera, K. Li, H. Agarwal, and I. Reid, "Unsupervised Learning Of Monocular Depth Estimation And Visual Odometry With Deep Feature Reconstruction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 340–349.

[241] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards End-To-End Visual Odometry With Deep Recurrent Convolutional Neural Networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2043–2050.

[242] ——, "End-To-End, Sequence-To-Sequence Probabilistic Visual Odometry Through Deep Neural Networks," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 513–542, 2018.

[243] K. S. Krishnan and F. Sahin, "ORBDeepOdometry-A Feature-Based Deep Learning Approach To Monocular Visual Odometry," in *SoSE*, 2019, pp. 296–301.

[244] S. Li, X. Wang, Y. Cao, F. Xue, Z. Yan, and H. Zha, "Self-Supervised Deep Visual Odometry With Online Adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6339–6348.

[245] N. Yang, R. Wang, J. Stuckler, and D. Cremers, "Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction For Monocular Direct Sparse Odometry," in *Proceedings of the European Conference On Computer Vision (ECCV)*, 2018, pp. 817–833.

[246] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," *arXiv preprint arXiv:1506.02025*, 2015.

[247] C.-H. Lin and S. Lucey, "Inverse compositional Spatial Transformer Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2568–2576.

[248] S. Wang, D. Quan, X. Liang, M. Ning, Y. Guo, and L. Jiao, "A Deep Learning Framework For Remote Sensing Image Registration," *ISPRS Journal Of Photogrammetry And Remote Sensing*, vol. 145, pp. 148–164, 2018.

[249] G. Liu, Y. Chen, and H. Chen, "Remote Sensing Image Registration Based On Fusion Of Spatial Transformation And Dense Convolution,"

in *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*. IEEE, 2020, pp. 21–26.

[250] L. Ma, Y. Liu, X. Zhang, Y. Ye, G. Yin, and B. A. Johnson, "Deep Learning In Remote Sensing Applications: A Meta-Analysis And Review," *ISPRS Journal Of Photogrammetry And Remote Sensing*, vol. 152, pp. 166–177, 2019.

[251] Z. Yang, T. Dan, and Y. Yang, "Multi-Temporal Remote Sensing Image Registration Using Deep Convolutional Features," *IEEE Access*, vol. 6, pp. 38 544–38 555, 2018.

[252] A. Gupta, Y. Peng, S. Watson, and H. Yin, "Multitemporal Aerial Image Registration Using Semantic Features," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2019, pp. 78–86.

[253] D. Quan, S. Wang, M. Ning, T. Xiong, and L. Jiao, "Using Deep Neural Networks For Synthetic Aperture Radar Image Registration," in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2016, pp. 2799–2802.

[254] J.-H. Park, W.-J. Nam, and S.-W. Lee, "A Two-Stream Symmetric Network With Bidirectional Ensemble For Aerial Image Matching," *Remote Sensing*, vol. 12, no. 3, p. 465, 2020.

[255] F. Ye, Y. Su, H. Xiao, X. Zhao, and W. Min, "Remote Sensing Image Registration Using Convolutional Neural Network Features," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 2, pp. 232–236, 2018.

[256] Y. Liu, X. Gong, J. Chen, S. Chen, and Y. Yang, "Rotation-Invariant Siamese Network For Low-Altitude Remote-Sensing Image Registration," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 5746–5758, 2020.

[257] W. Ma, J. Zhang, Y. Wu, L. Jiao, H. Zhu, and W. Zhao, "A Novel Two-Step Registration Method For Remote Sensing Images Based On Deep And Local Features," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 7, pp. 4834–4843, 2019.

[258] L. Li, L. Han, M. Ding, Z. Liu, and H. Cao, "Remote Sensing Image Registration Based on Deep Learning Regression Model," *IEEE Geoscience and Remote Sensing Letters*, 2020.

[259] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Deep Image Homography Estimation," *arXiv preprint arXiv:1606.03798*, 2016.

[260] M. Papadomanolaki, S. Christodoulidis, K. Karantzalos, and M. Vakalopoulou, "Unsupervised Multistep Deformable Registration Of Remote Sensing Imagery Based On Deep Learning," *Remote Sensing*, vol. 13, no. 7, p. 1294, 2021.

[261] G. Haskins, U. Kruger, and P. Yan, "Deep Learning In Medical Image Registration: A Survey," *Machine Vision and Applications*, vol. 31, no. 1, pp. 1–18, 2020.

[262] L. Hansen and M. P. Heinrich, "Deep Learning Based Geometric Registration For Medical Images: How Accurate Can We Get Without Visual Features?" in *International Conference on Information Processing in Medical Imaging*. Springer, 2021, pp. 18–30.

[263] B. Billot, D. Greve, K. Van Leemput, B. Fischl, J. E. Iglesias, and A. V. Dalca, "A Learning Strategy For Contrast-Agnostic MRI Segmentation," *arXiv preprint arXiv:2003.01995*, 2020.

[264] M. Hoffmann, B. Billot, J. Eugenio Iglesias, B. Fischl, and A. V. Dalca, "Learning Image Registration Without Images," *arXiv e-prints*, pp. arXiv–2004, 2020.

[265] M. Hoffmann, B. Billot, J. E. Iglesias, B. Fischl, and A. V. Dalca, "Learning MRI Contrast-Agnostic Registration," in *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 2021, pp. 899–903.

[266] K. He, R. Girshick, and P. Dollár, "Rethinking ImageNet Pre-Training," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4918–4927.

[267] D. P. Kingma and J. Ba, "ADAM: A Method For Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[268] K. Perlin, "An Image Synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

# Index