POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Trifocal Tensor Estimation for n-view Deep Structure-from-Motion

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Leonardo Perelli**

Student ID: 967295
Advisor: Prof. Luca Magri
Co-advisors: Dr. Andrea Porfiri dal Cin, Prof. Giacomo Boracchi
Academic Year: 2021-22

Abstract

Structure-from-Motion (*SfM*) is the problem of recovering the 3D structure of a scene from n images of the same scene taken at different viewpoints. The 3D structure can be recovered by estimating the position of the cameras from which images were taken (*camera pose*) as well as depth measurements at each pixel of the images (*depth map*). SfM pipelines are useful in many applications spanning from autonomous driving to augmented reality.

The number n of images used to infer a single depth map is crucial to reduce noise in the depth estimates and better handle occlusions. Neural networks have been used throughout the SfM pipeline to improve performance and robustenss. However, in the general case of n images neural networks typically do not take advantage of well known 3D geometric constraints. All the solutions proposed to alleviate this are limited to the base case of $n = 2$ views.

In this work we propose an SfM pipeline for the general n-view case which efficiently couples neural networks with the use of 3D geometric constraints. The pipeline leverages the Trifocal tensor and presents a novel pose chaining algorithm to expand camera pose estimation to the general case of n images. We also provide a comparison between different Trifocal tensor estimation algorithms together with their implementation.

We empirically show that our pipeline outperforms previous state-of-the-art SfM pipelines on the KITTI dataset while displaying promising results on ETH3D.

Parole chiave: depth estimation, trifocal tensor, pose estimation, plane sweep

Sommario

Structure-from-Motion (*SfM*) è il problema che consiste nello stimare la struttura tridimensionale di una scena a partire da n immagini della stessa, scattate da diversi punti di vista. La struttura 3D può essere recuperata stimando la posizione delle fotocamere da cui sono state scattate le immagini (*posa della camera*) e stimando la profondità per ogni pixel (*mappa di profondità*). Gli algoritmi di SfM sono utili in molte applicazioni che spaziano dalla guida autonoma alla realtà aumentata.
Il numero n di immagini usate per ricavare una singola mappa di profondità è un parametro cruciale che permette di ridurre il rumore nelle stime di profondità e di gestire meglio le occlusioni. Le reti neurali sono state utilizzate in tutte le fasi degli algoritmi SfM per migliorarne prestazioni e robustezza. Tuttavia, nel caso generale di n immagini le reti neurali spesso fanno un uso limitato o inefficiente di noti vincoli geometrici tridimensionali. Tutte le soluzioni proposte per alleviare questi svantaggi sono limitate al caso base di $n = 2$ immagini.
In questo lavoro proponiamo un algoritmo SfM ad n immagini che accoppia in modo efficiente le reti neurali con l'uso di vincoli geometrici 3D. La nostra proposta sfrutta il tensore Trifocale e inlcude un nuovo algoritmo di concatenamento della posa per espandere la stima della posa delle camere al caso generale di n immagini. Oltre a ciò, forniamo anche un confronto tra diversi algoritmi di stima del tensore Trifocale insieme alla loro implementazione.
Infine, mostriamo empiricamente che il nostro algoritmo proposto supera i precedenti algoritmi stato dell'arte di SfM sul dataset KITTI e ottiene risultati promettenti su quello di ETH3D.

Parole chiave: stima della profondità, tensore trifocale, stima della posa, plane sweep

Contents

1 | Introduction

In this work we address the problem of Structure-from-Motion (*SfM*), a fundamental task in Computer Vision. The problem of SfM consists in recovering the 3D structure of a scene starting from n images taken from different viewpoints but with some degree of overlap in their content. The 3D structure of a scene can be represented in many different ways, for example as a point cloud expressed in the reference frame of a camera. In this work we focus on retrieving the position of the cameras from which images were taken (*camera pose*) as well as depth measurements at each pixel of the images (*depth maps*). This is sufficient to recover a point cloud by merging the depth and pose estimates from the different views. Figure 1.1 shows an example of depth map obtained from an urban scene.

SfM algorithms are extremely useful in many applications spanning from autonomous driving to augmented reality, where they are mainly used to recover the depth of objects and obstacles or as basis for other tasks such as scene understanding, localization and path planning. In robotics, depth estimation is essential for obstacle avoidance, grasping, and navigation. Compared to other technologies, depth estimation based on imaging has several advantages. It can be applied to a wide range of scenarios, including indoor and outdoor environments, and works with passive sensors such as cameras, which are widely available and inexpensive. Notably, depth can be estimated at a fine scale, down to the level of individual pixels or sub-pixel accuracy, and depth information can be integrated with other computer vision tasks such as object detection, segmentation, and tracking. On the other hand, depth estimation algorithms are typically sensitive to lighting conditions, texture, and occlusions, which can affect the accuracy of the depth maps.

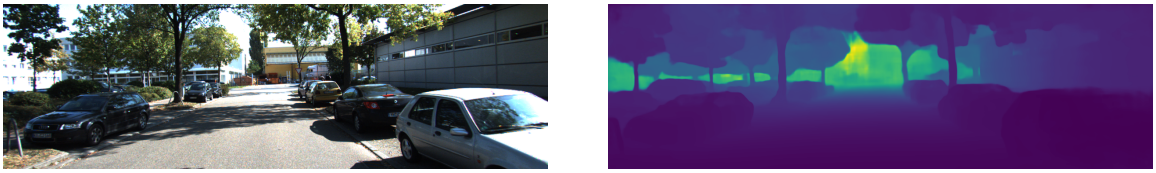


Figure 1.1: An urban scene and the corresponding estimated depth map.

Previously, SfM pipelines only relied on handcrafted tools such as point and feature match search over different images. With the advent of deep learning, neural networks have been employed throughout the pipeline. Both supervised and unsupervised learning approaches have been devised to tackle the depth estimation task. In the present work, we focus exclusively on the supervised learning approaches. Here, the neural networks are trained on large-scale datasets of RGB-D images, where the ground-truth depth maps are obtained from depth sensors such as LiDAR, structured light, or time-of-flight cameras.

An important parameter that determines the quality of the 3D reconstruction is the number n of images (or *views*) used to produce a single depth map. The objective is to produce a depth map for a specific *reference* image using information from the neighbouring *target* images. Hence, all pipelines leverage at least $n = 2$ images, where the base case consists of using a single target image together with the reference image. Generally, a higher number of images n provides better results as the effect of occlusions and noisy estimates tends to decrease. The typical SfM pipeline proceeds in two main stages: (i) recover camera pose (ii) use the estimated camera pose to compute depth maps. In this work, we focus on improving the camera pose estimates. Indeed, the majority of deep learning SfM pipelines directly regress the camera pose, providing the images as input to a neural network. Such pipelines have two main limitations: (i) they do not exploit the relationships and constraints of 3D geometry, over-relying on neural networks and (ii) they estimate the real scale of the scene, which is an ill-posed problem as it cannot be recovered from any number of images, unless specific information is provided. Overall, this leads to unreliable and biased camera pose estimates, which can limit the performance of the pipeline. Some pipelines try to alleviate this by refining the camera poses through additional computation in the form of repeated iterations of pose estimation, which is very expensive. A recent proposal tries to fully leverage the 3D constraints and couple it with the use of neural networks. However, the proposed pipeline is limited to the case $n = 2$ views. The present work aims to expand this approach to the general n-view case.

1.1. Contributions

In the present work we make the following contributions:

- (i) we propose an SfM pipeline which uses the Trifocal tensor to expand the camera pose estimation to the general n-view case. The pipeline couples the use of neural networks with the 3D geometrical constraints.
- (ii) we release a Python library [10] containing several algorithms to estimate the Trifocal tensor, a mathematical object which we use to recover camera pose estimates

over 3 views.

We perform different experiments on widely used datasets such as ETH3D [13] and KITTI Depth [2], comparing the proposed solution with existing work on the depth estimation task. The experiments show that our method improves over state-of-the-art pipelines on the KITTI Depth dataset by around 18% to 47% depending on the metric, while it displays promising results on ETH3D.

1.2. Structure

The structure of our work is organized as follows. In Chapter 2 we provide the fundamental theory from the domains of computer vision and projective geometry which we use throughout the rest of our work. Expert readers in such domains can skip the chapter. In Chapter 3 we formalize the problem statement, while Chapter 4 provides an overview of recent work produced to tackle the SfM problem, with particular focus on algorithms that leverage deep learning and neural networks. Chapter 5 details our proposed solution which we test extensively in chapter 6. Finally, Chapter 7 summarises our findings and achievements, proposing future improvements.

2 | Domain Background

In the following, we illustrate the fundamental elements of the classic Computer Vision pipeline used to solve the SfM problem. In particular, we start from illustrating the basic building blocks of projective geometry, such as the Camera Matrix, triangulation routines and useful algorithms. Following, we explore the epipolar geometry, that is the geometry induced by 2-views. This is very useful to understand concepts in a simpler setting with respect to the 3-views setting. The theoretical results obtained in this section are then extended to account for 3-views, which are at the core of our method. We present both the theoretical results both the computational algorithms used to estimate the mathematical objects we are dealing with. Most of the fundamental techniques mentioned are embedded in recent neural network based techniques. For example, we will build our algorithm around the estimation of the Trifocal tensor, which is explored in section 2.4.

2.1. Camera matrix

The camera matrix P is a 3x4 matrix that maps the 3D world points to their corresponding 2D image points. That is, given a 3D point in space visible to a camera, it is possible to find the exact 2D coordinates of the projection of the 3D point on the image plane, which is the plane on which the 3D world is projected to obtain an image. Both the 3D point and 2D point are expressed in homogeneous coordinates that can be subsequently transformed in Euclidean coordinates. The 3D point is commonly referred to as $\mathbf{X}$, while it's projection is $\mathbf{x}$. The camera matrix thus encodes the projective relationship: $P\mathbf{X} = \mathbf{x}$. The camera matrix can be decomposed in two matrices: $P = K[R|\mathbf{t}]$. The intrinsic matrix K contains the internal parameters of the camera, such as the focal length, the principal point, and the skew coefficient. The extrinsic matrix $[R|\mathbf{t}]$ contains the external parameters of the camera, such as the rotation and translation of the camera with respect to the world coordinate system. The camera matrix decomposition is very useful as it illustrates the conceptual difference between intrinsics and extrinsics. Indeed, the intrinsic parameters of the camera matrix are the properties that are specific to the camera and remain constant regardless of the scene being captured. Such parameters determine for example the size of

the image that the camera can capture. On the other hand, the extrinsic parameters of the camera matrix are the properties that describe the position and orientation of the camera with respect to the world coordinate system. These parameters include the rotation and translation of the camera. Rotation and translation can be expressed respectively through a 3×3 rotation matrix R and a 3×1 translation vector $\mathbf{t}$. Together, they represent the camera pose with a total of 6 degrees of freedom. In particular, it is useful to note that the extrinsic parameters describe the transformation that maps world coordinates to camera coordinates, rather than the opposite. This means that the translation vector represents the world center in camera coordinates, rather than the position of the camera in world coordinates, while the columns of R represent the directions of the world-axes in camera coordinates.

Remark 2.1.1 (Pose chaining)**.** Pose estimation is a central part of the SfM algorithms. As we will deal with multiple views, a common operation will be to chain relative poses. Let $\mathrm{C}_a, \mathrm{C}_b, \mathrm{C}_c$ be 3 cameras. Let $[\mathrm{R}_{ab}|\mathbf{t}_{ab}]$ and $[\mathrm{R}_{bc}|\mathbf{t}_{bc}]$ be the extrinsic matrices containing the pose of C_b with respect to C_a and of C_c with respect to C_b. This can be viewed as the extrinsic camera matrix of C_a in the world frame of C_b and C_b in the world frame of C_c respectively. We are interested in $[\mathrm{R}_{ac}|\mathbf{t}_{ac}]$, which is the pose of C_c with respect to C_a. Then, it can be derived that

$$\mathrm{R}_{ac} = \mathrm{R}_{bc}\mathrm{R}_{ab}$$

$$\mathbf{t}_{ac} = \mathbf{t}_{ab} + \mathrm{R}_{ab}\mathbf{t}_{bc}$$

Sometimes, it could also be useful to invert a relative pose $[\mathrm{R}_{ab}|\mathbf{t}_{ab}]$ from $[\mathrm{R}_{ba}|\mathbf{t}_{ba}]$. This can be accomplished as:

$$\mathrm{R}_{ab} = \mathrm{R}_{ba}^{\top}$$

$$\mathbf{t}_{ab} = -\mathrm{R}_{ba}^{\top}\mathbf{t}_{ba}$$

2.2. Triangulation

A frequent operation in 3D reconstruction algorithms is to triangulate the 3D position of a point given it's 2D position in M cameras. The problem for $M = 2$ can be formulated as:

$$\text{Find } \mathbf{X} \text{ such that } \mathbf{x} = \mathrm{P}\mathbf{X} \text{ and } \mathbf{x}' = \mathrm{P}'\mathbf{X}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are two matching points while P and P′ are two camera matrices.
Of course, if both the camera matrices and the 2D points are measured exactly, it is sufficient to intersect the back-projected rays. In the case in which the point correspondences

are not exact however, the rays will not intersect. This can be equivalently formulated as noting that the corresponding points do not satisfy the Fundamental matrix constraint. In the following, we detail a simple linear estimate of the 3D point.
From $\mathbf{x} = \mathrm{P}\mathbf{X}$ we can equivalently obtain $\mathbf{x} \times (\mathrm{P}\mathbf{X}) = 0$. By expanding the cross product we obtain 3 homogeneous equations, of which only 2 are linearly independent. Such equations can be arranged in the system:

$$\mathrm{A}\mathbf{X} = 0$$

where A is of shape 2×4 as 2 equations are used and the 3D point is a homogeneous quantity. Since the system is homogeneous, one possible solution is to factorise A with SVD as $\mathrm{A} = \mathrm{UDV}^\top$. Then, the last column of V is the eigenvector associated to the smallest eigenvalue of A, which must be 0. Indeed, A must have rank 3 as $\mathbf{X}$ is a homogeneous quantity. Such eigenvector is a solution of the system and provides the homogeneous coordinates of the 3D point $\mathbf{X}$.

2.3. Epipolar Geometry

Epipolar geometry is the term used to describe the relationships between 3D points in space and 2D points in the images. The epipolar geometry of a scene can be characterized through point matches, also called point correspondences. Suppose there exists a 3D point $\mathbf{X}$ imaged in two different views. Given that $\mathbf{X}$ is pictured in point $\mathbf{x}$ in the first view, where is the corresponding point $\mathbf{x}'$ located in the second view? What is the relationship between these two points? It turns out that such geometry can be fully encoded and described by a matrix F called the Fundamental matrix. Before analysing its properties, we lay out some important terms and definitions.

Definition 2.3.1. Let C and C′ be two camera centres, we define:

- **Camera Plane**: the plane on which the 3D points are projected to obtain an image
- **Baseline**: the line connecting camera centres C and C′
- **Epipole**: the intersection between the baseline and one of the camera planes. It represents the projection of the camera center of one image in the other image
- **Epipolar plane**: any plane which contains the baseline
- **Epipolar line**: the intersection between an epipolar plane and a camera plane.

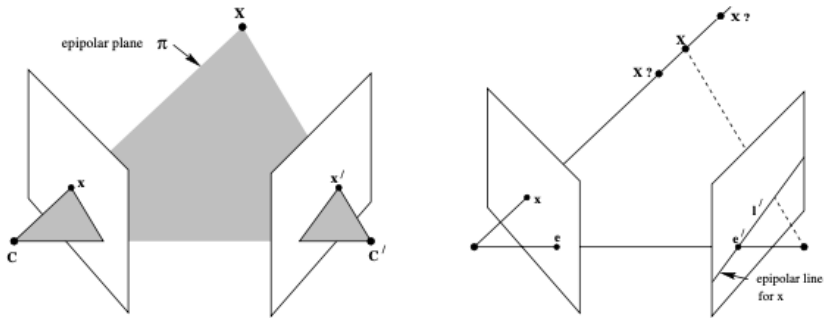


Figure 2.1: The epipolar plane and induced epipolar lines. Adapted from [3]

2.3.1. Fundamental matrix

As seen in the previous section, given a point $\mathbf{x}$ in the first view, we are interested in finding the coordinates of the corresponding point $\mathbf{x}'$ in the second view. It turns out that such point in the first view is located on an epipolar line of the other view. This is a strong constraint which can be exploited when searching for correspondences. The Fundamental matrix is a convenient way to encode such epipolar geometry.

Definition 2.3.2 (Fundamental matrix)**.** Given 2 images with non coincident camera centres, there exists a unique 3x3 rank 2 homogeneous matrix F such that:

$$\mathbf{x}'^{\top} \mathrm{F} \mathbf{x} = 0 \tag{2.1}$$

for any corresponding points $\mathbf{x}$ and $\mathbf{x}'$.

There are a few important properties that characterize all fundamental matrices.

Property 2.3.1. Let F be a Fundamental matrix. Let $\mathbf{x}$ be a point in the first view and $\mathbf{x}'$ a point in the second view. Then,

- F displays seven degrees of freedom
- Given point $\mathbf{x}$, the corresponding epipolar line is $\mathbf{l}' = \mathrm{F}\mathbf{x}$
- Given $\mathbf{x}'$, the corresponding epipolar line is $\mathbf{l} = \mathrm{F}^{\top}\mathbf{x}'$
- $\mathrm{F}\mathbf{e} = \mathbf{0}$ and $\mathrm{F}^{\top}\mathbf{e}' = \mathbf{0}$ where $\mathbf{e}$ and $\mathbf{e}'$ are the 2 epipoles
- For two general camera matrices P and P′, $\mathrm{F} = [\mathbf{e}']_{\times}\mathrm{P}'\mathrm{P}^{+}$, where P^{+} is the pseudo-inverse of P, and $\mathbf{e}' = \mathrm{P}'\mathrm{C}$, where C is the camera centre of the first view
- For two canonical cameras $\mathrm{P} = [\mathrm{I} \mid \mathbf{0}]$ and $\mathrm{P}' = [\mathrm{M} \mid \mathbf{m}]$, $\mathrm{F} = [\mathbf{e}']_{\times}\mathrm{M} = \mathrm{M}^{-\mathrm{T}}[\mathbf{e}]_{\times}$, where $\mathbf{e}' = \mathbf{m}$ and $\mathbf{e} = \mathrm{M}^{-1}\mathbf{m}$

Note that a homogeneous 3x3 matrix has 8 degrees of freedom, as one is lost due to the homogeneous coordinates. On top of this, the remaining degree of freedom is reduced since the Fundamental matrix also satisfies the following constraint:

$$det(\mathrm{F}) = 0 \tag{2.2}$$

Such constraint will be enforced to obtain a valid estimate of the Fundamental matrix in the following sections.

An interesting remark is that F depends on projective properties of the camera matrices P and P′. In contrast, the Fundamental matrix is unchanged when changing the world frames. More in general, F is invariant with respect to projective transformations. Let $\mathbf{H}$ be a homography. Computing F between $(\mathrm{P}, \mathrm{P}')$ and $(\mathrm{P}\mathbf{H}, \mathrm{P}'\mathbf{H})$ yields the same result. For this reason, given a Fundamental matrix from which we want to extract the camera matrices, it makes sense to assume the first view to simply be: $[\mathrm{I} \mid \mathbf{0}]$. Indeed, we have seen that given two canonical camera matrices, we can recover a Fundamental matrix. At the same time however, we can recover a couple of canonical camera matrices from a Fundamental matrix.

Property 2.3.2. Given a Fundamental matrix F, a couple of corresponding camera matrices can be set as:

$$\mathrm{P} = [\mathrm{I} \mid \mathbf{0}] \text{ and } \mathrm{P}' = [[\mathbf{e}']_\times \mathrm{F} \mid \mathbf{e}'],$$

where $\mathbf{e}'$ is such that $\mathrm{F}^\top \mathbf{e}' = \mathbf{0}$.

2.3.2. Essential matrix

There is another important matrix capable of encoding scene geometry. In particular, in the case of known intrinsic camera parameters, one can define the Essential matrix $\mathbf{e}$.

Definition 2.3.3 (Essential matrix)**.** The point-point correspondence is very similar to the definition 2.3.2 of F:

$$\hat{\mathbf{x}}'^\top \mathrm{E} \hat{\mathbf{x}} = 0 \tag{2.3}$$

where $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ are two corresponding points in *normalized coordinates*.

It is possible to obtain the normalized coordinates from a point in homogeneous coordinates as:

$$\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x} \tag{2.4}$$

That is, the Essential matrix describes the geometry induced by 2 *normalized* camera matrices $\mathrm{P} = [\mathrm{I}|\mathbf{0}]$ and $\mathrm{P}' = [\mathrm{R}|\mathbf{t}]$. Following the properties of the Fundamental matrix expressed in Property 2.3.1, E can be expressed as:

$$\mathrm{E} = [\mathbf{t}']_{\times}\,\mathrm{R} \tag{2.5}$$

Remark 2.3.1. An interesting remark that can be deduced from this equation is that E has 5 degrees of freedom. Indeed, the rotation matrix R and the translation $\mathbf{t}$ have both 3 degrees of freedom. There is however a scale ambiguity as the Essential matrix is an homogeneous quantity exactly as the Fundamental matrix. Hence, the overall degrees of freedom is reduced by 1.

From the previous point-point correspondence equations, one can deduce that the relationship between E and F is:

$$\mathrm{E} = \mathrm{K}'^{\top}\mathrm{F}\mathrm{K}$$

Remark 2.3.2. It is important to notice that given the Essential matrix E, it is not possible to recover the scale of translation $\mathbf{t}$, but rather it can be estimated only up to scale. There are two different ways to think about this fact. The first is simply to notice that the Essential matrix is a homogeneous quantity, hence up to scale. For this reason, extracting the value of $\mathbf{t}$ from equation 2.5 leads to an up to scale value. Another way to view this fact is instead to define the Essential matrix, including scale, with equation 2.5 and determining E up to scale from equation 2.3. Indeed, this last equation does not impose any scale restriction on E. In general, the translation vector is simply normalized to have $\|\mathbf{t}\| = 1$. Of course, the same holds for the Fundamental matrix too.

2.3.3. Linear Estimation

In the following sections we detail the main methods and algorithms used to estimate the Fundamental matrix. As will be seen further on, they are extremely useful and can be adapted to estimate other objects too, such as the Trifocal tensor. While we focus on the Fundamental matrix, the same algorithms also apply to the Essential matrix. The Linear estimation methods arises from the definition of the Fundamental matrix, in particular from equation 2.1. Given the 2D point correspondence $\mathbf{x} \leftrightarrow \mathbf{x}'$ expressed in homogeneous coordinates as $(x, y, 1)$ and $(x', y', 1)$, the equation $\mathbf{x}'^{\top}\mathrm{F}\mathbf{x} = 0$ can be expanded as:

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0$$

By collecting in a 9 dimensional vector $\mathbf{f}$ the entries of F and considering the general case of n points, we obtain the following linear system:

$$\mathrm{A}\mathbf{f} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{f} = \mathbf{0}$$

As the linear system is homogeneous, the solution $\mathbf{f}$ can be found only up to scale. In particular, if the rank of A is exactly 8, the solution is unique (up to scale). However, if there is noise in the measurements, A could end up having rank 9 and being inconsistent. In that case, one must resort to the least squares estimation and SVD decomposition of A.

Remark 2.3.3. As seen in equation 2.2, the Fundamental matrix must have a null determinant, which is equivalent to not having a full rank. The previous estimates obtained from $\mathbf{f}$ however do not impose this constraint in any way, meaning that the estimated Fundamental matrix in general will not be valid. This would be an issue as such a matrix would have misbehaving epipolar lines, that is epipolar lines in the same camera plane which do not intersect in the same epipole.

To solve this issue and obtain a valid Fundamental matrix, there are 2 main methods.
The first option is to estimate the Fundamental matrix without imposing any constraint on the determinant, for example with the previous linear estimates. Then, compute a new $\mathrm{F}' = \mathrm{U}diag(r, s, 0)\mathrm{V}^\top$ obtained by SVD decomposition of the original Fundamental matrix as $\mathrm{F} = \mathrm{U}diag(r, s, t)\mathrm{V}^\top$. The new matrix F' is the nearest singular matrix in Frobenius norm to F.
Alternatively, one can directly parametrize F with a 3x3 singular matrix. For example, one can choose

$$\mathrm{F} = [\mathbf{t}]_\times \mathrm{M} \tag{2.6}$$

which is a singular matrix since $[\mathbf{t}]_\times$ is singular. This parametrization requires a total of 12 parameters, 9 coming from M and 3 from $\mathbf{t}$. While this provides an over-parametrization since F only has 7 degrees of freedom, in general it is not a problem.

Remark 2.3.4. To improve the numerical performance of the linear algorithm, it is

generally considered a good practice to normalize the data, in this case the point correspondences. Here, we refer to an actual translation and scaling of the coordinates, rather than the normalization concept refered to when talking about the Essential matrix and the intrinsic parameters. Generally, the normaliaztion is carried out in order to obtain the centre of the points in the origin and a root mean square distance of $\sqrt{2}$. Notice that since the 2D point correspondences are homogeneous quantities, it is possible to express both the translation and the scaling with a single 3x3 matrix T. Hence, the point correspondences used as input to the linear algorithm will be: $\hat{\mathbf{x}}_i = \mathrm{T}\mathbf{x}_i$ and $\hat{\mathbf{x}}'_i = \mathrm{T}\mathbf{x}'_i$.

2.3.4. Gold Standard

The Gold Standard algorithm is one of the best options to estimate the Fundamental matrix. In the case of Gaussian noise perturbation of the point correspondences, this method returns the Maximum Likelihood Estimation of the Fundamental matrix. The main difference with respect to the previous methods is in both the minimization target and the quantities parametrized. Indeed, in this case there will be 12 parameters for the direct singular parametrization of F as shown in equation 2.6 and also the n 3D points will be parametrized, yielding other $3n$ parameters as the last homogeneous coordinate is set to 1. The distance to be minimized in this case coincides with the reprojection error, which can be defined as:

$$\mathcal{D} = \sum_i d\left(\mathbf{x}_i, \hat{\mathbf{x}}_i\right)^2 + d\left(\mathbf{x}'_i, \hat{\mathbf{x}}'_i\right)^2 \tag{2.7}$$

with $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ the measured correspondences, and $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}'_i$ are estimated "true" correspondences that satisfy $\hat{\mathbf{x}}_i'^{\top}\mathrm{F}\hat{\mathbf{x}}_i = 0$ exactly for some rank-2 matrix F, the estimated Fundamental matrix.

Algorithm 2.1 Gold Standard estimation of F

N: total number of iterations
n: total number of point correspondances available
Collect $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ for $i = 1, ..., n$, with $n \geq 8$
Compute an initial estimate $\hat{\mathrm{F}}$ with the normalized linear method from section 2.3.3
if $det(\hat{\mathrm{F}}) \neq 0$ **then**
    Correct $\hat{\mathrm{F}}$ by SVD decomposition
end if
Extract the epipole $\mathbf{e}'$ from $\hat{\mathrm{F}}$, using $\mathrm{F}'^{\top}\mathbf{e}' = \mathbf{0}$.
Recover the camera matrices associated to $\hat{\mathrm{F}}$ through: $\mathrm{P} = [\mathrm{I}|\mathbf{0}]$ and $\mathrm{P}' = [[\mathbf{e}']_{\times}\hat{\mathrm{F}} \mid \mathbf{e}']$

Triangulate the 2D point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ using P, P' and obtain an initial estimate of the 3D points $\hat{\mathbf{X}}_i$
for i in $1 : N$ **do**
    Minimize $\sum_i d\left(\mathbf{x}_i, \mathrm{P}\hat{\mathbf{X}}_i\right)^2 + d\left(\mathbf{x}'_i, \mathrm{P}'\hat{\mathbf{X}}'_i\right)^2$ over F and $\{\hat{\mathbf{X}}_i\}_{i=1}^n$ using Levenberg-Marquardt
    Update the camera matrix P' from the new values of $\hat{\mathrm{F}}$
end for

Notice that by parametrizing the 3D points $\{\hat{\mathbf{X}}_i\}_{i=1}^n$ and projecting them to $\{\hat{\mathbf{x}}_i\}_{i=1}^n$ with camera matrices obtained from F, we are sure that the condition $\hat{\mathbf{x}'}^{\top}\mathrm{F}\hat{\mathbf{x}} = 0$ is always valid.

2.4. Trifocal tensor

In the previous sections we introduced the main tools and concepts to deal with 2-view geometry. In the following section, we focus on the *Trifocal Tensor* T, which can be thought of as a generalization to the Fundamental matrix. As such, it expresses the relationship between lines and points in 3 different views.

2.4.1. Definition and properties

We will mainly characterize the Trifocal tensor based on the point-point-point relationships as they turn out to be the most useful in our application. T is a collection of three 3×3 matrices $\{\mathrm{T}_i\}_{i=1}^3$ accounting for a total of 27 elements. The $\{\mathrm{T}_i\}_{i=1}^3$ are such that $\mathrm{T} = [\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3]$ Excluding the overall scale, or considering the tensor in homogeneous coordinates, we actually have 26 independent parameters. Still, it turns out that T displays

only 18 degrees of freedom. As a result, there must be $26 - 18 = 8$ algebraic constraints that the tensor fulfills.

Definition 2.4.1. Let us set $\mathrm{P}_1 = [\mathrm{I}|\mathbf{0}]$, $\mathrm{P}_2 = [\mathrm{A}|\mathbf{a_4}]$ and $\mathrm{P}_3 = [\mathrm{B}|\mathbf{b_4}]$ to be three canonical camera matrices. Then:

$$\mathrm{T}_i = \mathbf{a_i}\mathbf{b}_4^\top - \mathbf{b}_4\mathbf{a}_i^\top \tag{2.8}$$

where $\mathbf{a}_i$ and $\mathbf{b}_i$ are the columns of A and B.

Equation 2.8 is very important as it characterizes the tensor based on 3 canonical camera matrices. This relationship is used in several algorithms to estimate T.
We now introduce the correspondence equations over three views. They can be viewed as the extension of the equations 2.3 for the Fundamental matrix.

Property 2.4.1. Let $\mathbf{x}, \mathbf{x}', \mathbf{x}''$ be 3 point matching points in the first, second and third view, while $\mathbf{l}, \mathbf{l}', \mathbf{l}''$ are 3 corresponding lines in the same views. The correspondences induced by the Trifocal tensor can be summarised as:

- Line-line-line correspondence

$$\mathbf{l}'^{\mathrm{T}} \left[\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3\right] \mathbf{l}'' = \mathbf{l}^{\mathrm{T}} \text{ for a correspondence } \mathbf{l} \leftrightarrow \mathbf{l}' \leftrightarrow \mathbf{l}''$$

- Point-line-line correspondence

$$\mathbf{l}'^{\top} \left(\sum_i \mathbf{x}_i\ \mathrm{T}_i \right) \mathbf{l}'' = 0 \text{ for a correspondence } \mathbf{x} \leftrightarrow \mathbf{l}' \leftrightarrow \mathbf{l}''$$

- Point-line-point correspondence

$$\mathbf{l}'^{\top} \left(\sum_i \mathbf{x}_i\ \mathrm{T}_i \right) [\mathbf{x}'']_\times = \mathbf{0}^\top \text{ for a correspondence } \mathbf{x} \leftrightarrow \mathbf{l}' \leftrightarrow \mathbf{x}''$$

- Point-point-line correspondence

$$[\mathbf{x}']_\times \left(\sum_i \mathbf{x}_i\ \mathrm{T}_i \right) \mathbf{l}'' = \mathbf{0} \text{ for a correspondence } \mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{l}''$$

- Point-point-point correspondence

$$[\mathbf{x}']_\times \left(\sum_i \mathbf{x}_i \ \mathrm{T}_i \right) [\mathbf{x}'']_\times = \mathbf{0}_{3\times 3} \tag{2.9}$$

where $\mathbf{x}_i$ represents the i^{th} coordinate of $\mathbf{x}$.

Given the point-line-line correspondence, we can now derive an important property of epipoles.

Property 2.4.2. Let $\mathbf{X}$ be a point in 3D space and $\mathbf{l}'$ its epipolar line in the 2^{nd} image. Then:

$$\mathbf{l}'^\top \left(\sum_i x_i \ \mathrm{T}_i \right) = 0$$

meaning that the left null vector of $\sum_i x_i \ \mathrm{T}_i$ is the epipolar line in the 2^{nd} image.
The same can be obtained for the epipole in the 3^{rd} image:

$$\left(\sum_i x_i \ \mathrm{T}_i \right) \mathbf{l}'' = 0$$

That is, the epiolar line $\mathbf{l}''$ is the right null vector of T.

As in the case of the Fundamental matrix, T also encodes information on the homographies between different views. For example, it is possible to determine the homography H between the 1^{st} and the 3^{rd} images by using a line in the 2^{nd} image. Let $\mathbf{l}'$ be a line in the 2^{nd} image. Then:

$$\mathrm{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3], \text{ where } \mathbf{h_i} = \mathrm{T}_i^\top \mathbf{l}'$$

Remark 2.4.1. Some useful properties of the T include:

- T_i has rank 2 $\forall \ i = 1, ..., 3$
- $\mathbf{l}''_i = e'' \times \mathbf{b}_i$ is the right null-vector of T_i
- Epipolar lines $\mathbf{l}''_i$ for $i = 1, 2, 3$ intersect in the epipole e''
- $\mathbf{l}'_i = e' \times \mathbf{a}_i$ is the left null-vector of T_i

- Epipolar lines $\mathbf{l}'_i$ for $i = 1, 2, 3$ intersect in the epipole e'

Property 2.4.3. A useful property of the Trifocal tensor is that we can recover the fundamental matrices between different views by leveraging previous relationships. The epipolar line corresponding to point $\mathbf{x}$ can be computed by projecting $\mathbf{x}$ to the other view and linked the projected point to the epipole. Let $\mathbf{x}$ be in the first view. We showed that we can recover the homography between view 1 and 2 by computing $[\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3]\,\mathbf{l}''$. Applied to $\mathbf{x}$, it transfers the point from view 1 to view 2. So, $\mathbf{l}' = [\mathbf{e}']_\times\,([\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3]\,\mathbf{l}'')\,\mathbf{x}$. By choosing $\mathbf{l}'' = \mathbf{e}''$ we obtain:

$$\mathrm{F}_{21} = [\mathbf{e}']_\times\,[\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3]\,\mathbf{e}''$$

Similarly,

$$\mathrm{F}_{31} = [\mathbf{e}'']_\times\,[\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3]\,\mathbf{e}'$$

Property 2.4.4. To recover camera matrices from T, we can resort to the previous computations of the fundamental matrices. As the geometry is defined up to a projective ambiguity, we can define $\mathrm{P}_1 = [\mathrm{I}|\mathbf{0}]$. Giver our knowledge of F_{21}, we can derive that:

$$\mathrm{P}' = [[\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3]\,\mathbf{e}''|\mathbf{e}']$$

which implies that the couple $\{\mathrm{P}, \mathrm{P}'\}$ have Fundamental matrix F_{21}. While one could expect to define:

$$\mathrm{P}'' = [[\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3]\,\mathbf{e}'|\mathbf{e}'']$$

it turns out that the triplet $\{\mathrm{P}, \mathrm{P}', \mathrm{P}''\}$ would be inconsistent. Rather, the correct definition of the third camera matrix is:

$$\mathrm{P}'' = [\left(\mathbf{e}''\mathbf{e}''^\top - \mathrm{I}\right)[\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3]\,\mathbf{e}'|\mathbf{e}'']$$

Given that it is possible to compute T from the camera matrices, and also to compute the camera matrices given T, the Trifocal tensor fully captures the 3D geometry in the 3-view setting. As usual, this holds up to a projective equivalence.

2.4.2. Estimation algorithms

In the following sections we explore the several algorithms available to compute a numerical estimate of T. One key ingredient in the numerical estimate is to remember that the Trifocal tensor must be a valid one. As such, it must satisfy the condition expressed in Equation 2.8. If there exist 3 camera matrices associated to T that satisfy this expression, the Trifocal tensor is said to be "geometrically valid". Failing to satisfy these constraints implies that the induced geometry might not be consistent. For example, the final position of a point transferred from one view to another will depend on which relationship is used, rather than being unique.
Section 2.4.3 details the linear estimation of T. While this can be a helpful initialization for more advanced numerical schemes, it doesn't always provide a good estimate if used by itself. For this reason, section 2.4.4 introduces the Gold Standard algorithm based on minimization of reprojection error. Section 2.4.5 provides an overview of the relationship between T and F. All the algorithms can be implemented in couple with the RANSAC algorithm described in section 2.6 to obtain more robust estimates.

2.4.3. Linear estimation

The linear estimate of T is obtained by setting up a linear system and trying to satisfy the point-point-point relationship. For each point $\mathbf{X}$, the relationship 2.9 implies 9 constraints, that is 9 linear equations. Actually, of those 9, only 4 are linearly independent. Given that T has 26 degrees of freedom, at least 7 points are needed to estimate it fully. As for the Fundamental matrix, normalizing the input data is of great help to obtain a better conditioned numerical problem. By expanding 2.9 and rearranging the terms, one can obtain a linear system of the shape $A\mathbf{x} = \mathbf{0}$. As soon as there are more than 7 points however, the system can only be solved by performing a least squares approximation due to the error on the measurements of corresponding points. As such, the linear estimation method will, in general, yield a non valid tensor.

2.4.4. Gold Standard

The Gold Standard algorithm proceeds by minimizing the reprojection error. To make sure that the computed Trifocal tensor is valid, the algorithm directly parametrizes the 3 projection matrices. By choosing $P_1 = [I|\mathbf{0}]$, one can simply parametrize P_2 and P_3. As they are both 3×4 matrices, the total parameters are 24. While this means that the Trifocal tensor is overparametrized, indeed only 18 parameters are needed, this is not a problem in the optimization procedure. In the 3-view setting, the reprojection error can

be expressed as:

$$\mathcal{D} = \sum_i d\left(\mathbf{x}_i, \hat{\mathbf{x}}_i\right)^2 + d\left(\mathbf{x}'_i, \hat{\mathbf{x}}'_i\right)^2 + d\left(\mathbf{x}''_i, \hat{\mathbf{x}}''_i\right)^2$$

where $\mathbf{x}_i$,$\mathbf{x}'_i, \mathbf{x}''_i$ are matched points and $\hat{\mathbf{x}}_i$,$\hat{\mathbf{x}}'_i, \hat{\mathbf{x}}''_i$ the estimated points in the 3 views. In the Gold Standard algorithm, the optimization is carried out not only with respect to the camera matrices, but also with respect to a set of 3D points. Once the 3D points are projected to the 2D points in the 3-views, the reprojection error is minimized iteratively. As such, one can compute a set of point correspondences $\{\hat{\mathbf{x}}_i \leftrightarrow \hat{\mathbf{x}}'_i \leftrightarrow \hat{\mathbf{x}}''_i\}$, which exactly satisfy the trilinear relations of the estimated tensor in 2.9. Overall, the minimization is carried out over a total of $3n + 24$ variables: $3n$ for the n 3D points $\widehat{\mathbf{X}}_i$, and 24 for the elements of the camera matrices $\mathrm{P}', \mathrm{P}''$.

Algorithm 2.2 Gold Standard estimation of T

1: N: total number of iterations
2: n: total number of point correspondances available
3: Collect $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i \leftrightarrow \mathbf{x}''_i$ for $i = 1, ..., n$, with $n \geq 7$
4: Compute a valid initialization of $\hat{\mathrm{T}}$ through a linear estimate
5: Compute the camera matrices P', P'' from $\hat{\mathrm{T}}$ using 2.8.
6: Triangulate $\{\hat{\mathbf{X}}\}_{i=1}^n$ from the correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i \leftrightarrow \mathbf{x}''_i$ using camera matrices $\mathrm{P} = [\mathrm{I} \mid \mathbf{0}], \mathrm{P}', \mathrm{P}''$
7: **for** i in $1 : N$ **do**
8: Minimize $\sum_i d\left(\mathbf{x}_i, \mathrm{P}\hat{\mathbf{X}}_i\right)^2 + d\left(\mathbf{x}'_i, \mathrm{P}'\hat{\mathbf{X}}_i\right)^2 + d\left(\mathbf{x}''_i, \mathrm{P}''\widehat{\mathbf{X}}_i\right)^2$ over P', P'' and $\{\hat{\mathbf{X}}_i\}_{i=1}^n$ using Levenberg-Marquardt
9: **end for**

2.4.5. Trifocal tensor from Fundamental matrix

Rather than directly estimating the Trifocal tensor from the trilinear constraints, one could prefer to estimate it from Fundamental matrices. That is, given F_{12} and F_{13}, we are interested in recovering a valid Trifocal tensor. Of course, one can extract the two couples $\mathrm{R}_{12}, \mathbf{t}_{12}$ and $\mathrm{R}_{13}, \mathbf{t}_{13}$, however both the translation vectors have unknown scale, hence it is necessary to find a relative scale between the two. To start, one is free to set $||\mathbf{t}_{12}|| = 1$, thanks to the 3D reconstruction being up to a projective transformation. At this point, one has to determine the scale λ of $\mathbf{t}_{13}$, relative to $\mathbf{t}_{12}$. Fortunately, [7] shows that it is possible to recover this relative scale by solving a minimization problem which has a closed form solution. To do this, one has to triangulate the 2D points $\mathbf{x}_i$ and $\mathbf{x}'_i$ in

order to recover the 3D points $\{\mathbf{X}_i\}_{n=1}^{N}$. Then, minimization of algebraic error is carried out:

$$\arg\min_{\lambda\in\mathrm{R}} \sum_{n=1}^{N} \left\| \mathbf{x}_{3,n} \times \left(\mathrm{K}_3 \left(\mathrm{R}_{31}\mathbf{X}_n + \lambda \frac{\mathbf{t}_{31}}{\|\mathbf{t}_{31}\|} \right) \right) \right\|^2$$

In case one has a very accurate algorithm to compute the Fundamental matrices, this is a great way to leverage it for the 3-view case.

2.5. Sampson Error

Most of the estimation algorithms we introduce require to minimize some error. The common choices are to minimize either the reprojection error or the algebraic error. The reprojection error generally requires a more complex minimization process. For example, the Gold Standard estimates of the Fundamental matrix and Trifocal tensor, which minimize the reprojection error, not only require to parametrize F or T, but also the 3D points $\mathbf{X}_i$. This is in contrast with the algebraic error, which has a simpler formulation. The Sampson error is an attempt to strike a balance between the two aforementioned approaches. Indeed, it attempts to linearize the reprojection error in the neighbourhood of the point. The Fundamental matrix and Trifocal tensor implicitly define a variety through a linear system of shape $\mathrm{A}\mathbf{x} = \mathbf{0}$, where A depends on the point correspondences. Hence, any point matches which are consistent with F or T satisfy the linear system. In the case of the Fundamental matrix, A depends on the coordinates of the matching points, that can be seen as a vector $\mathrm{X} = (x, y, x', y')$. Since $\mathrm{A}\mathbf{x}$ is a 2D vector which represents the variety, we refer to it as $\mathcal{C}(\mathbf{X})$. One can think of $\mathcal{C}(\mathbf{X})$ as a cost function. Indeed, the matching points that do not satisfy the epipolar constraints will not lie on the variety, while points that are consistent with F will lie on it. By performing a first order Taylor expansion to approximate such cost, on obtains: cost function may be approximated by a Taylor expansion:

$$\mathcal{C}\left(\mathbf{X} + \boldsymbol{\delta}_{\mathbf{X}}\right) = \mathcal{C}(\mathbf{X}) + \frac{\partial \mathcal{C}}{\partial \mathbf{X}} \boldsymbol{\delta}_{\mathbf{X}}.$$

The distance from a point $\mathbf{X}$ to the variety is the minimum distance between $\mathbf{X}$ and a point on the variety $\widehat{\mathbf{X}}$. Here, we assume that X is sufficiently near the variety, so that one can choose $\delta_{\mathbf{X}} = \widehat{\mathbf{X}} - \mathbf{X}$. Since $\widehat{\mathbf{X}}$ lies on the variety, we know that $\mathcal{C}(\widehat{\mathbf{X}}) = \mathbf{0}$. By substituting this inside the equation, we obtain

$$\mathcal{C}(\mathbf{X}) + \frac{\partial \mathcal{C}}{\partial \mathbf{X}} \boldsymbol{\delta}_{\mathbf{X}} = \mathbf{0}$$

By noticing that the partial derivative is a Jacobian matrix J, and letting ϵ be the cost

$\mathcal{C}(\mathbf{X})$, we can simplify the expression to:

$$\mathrm{J}\boldsymbol{\delta}_{\mathbf{X}} = -\boldsymbol{\epsilon}$$

By ssing lagrangian multipliers to minimize the norm of the error $\boldsymbol{\delta}_{\mathbf{X}}$, one can finally obtain the expression for the Sampson error:

$$\|\boldsymbol{\delta}_{\mathbf{X}}\|^2 = \boldsymbol{\delta}_{\mathbf{X}}^{\top}\boldsymbol{\delta}_{\mathbf{X}} = \boldsymbol{\epsilon}^{\top}\left(\mathrm{J}\mathrm{J}^{\top}\right)^{-1}\boldsymbol{\epsilon} \tag{2.10}$$

When estimating the Trifocal tensor or the Fundamental matrix, a very useful remark to simplify the computation of the Jacobian is that $\mathcal{C}$ is multilinear with respect to the coordinates of the point matches. One can check this by recalling the definition of the cost $\mathcal{C}$, which in such cases follows the point-point or point-point-point correspondences. Then, the following equation:

$$\frac{\partial \mathcal{C}}{\partial \mathbf{X}_i}(\mathbf{X}) = \mathcal{C}(\mathbf{X} + \mathbf{u}_i) - \mathcal{C}(\mathbf{X})$$

holds exactly, rather than being an approximation. Here, $\mathbf{u}_i$ is a unit vector with value 1 in index i. This equation can hence be leveraged to compute the Jacobian without the need to compute explicitly any partial derivative.

2.6. RANSAC

In this section we describe the RANSAC algorithm [1], which is used throughout our work to perform robust estimates of 3D reconstruction models, such as the Essential matrix, the Fundamental matrix the and the Trifocal tensor. RANSAC, which stands for Random Sample Consensus, is an iterative algorithm used to estimate the parameters of a model from a dataset which could contain outliers. Given that the procedure is based on repeated sub-samples from the initial dataset, the algorithm is stochastic and produces a good result only with a certain probability. This probability however increases as more samples are performed.

The algorithm assumes that the dataset contains a certain amount of outliers: such points should be disregarded when estimating the model's parameters. Indeed, begin outliers, they cannot fit the true underlying model we are looking to estimate in the first place. The algorithm also assumes that the inliers are a set of points which can be explained by some choice of the model parameters, at least up to small noise perturbations. In the

domain of 3D reconstruction, for example when estimating the Fundamental matrix and Trifocal tensor, a common source of outliers are erroneous measurements or erroneous point matches from the optical flow. When dynamic objects are present in the scene, the point matches obtained for such objects can be considered as outliers with respect to the set of point matches obtained from the images. Indeed, their position cannot be explained by the epipolar constraints which are obtained from the static scene.

A common example to showcase RANSAC is fitting a line to a set of points. The points are a mixture of both inliers and outliers. The outliers do not fit the model, while the inliers can be obtained by perturbing some line with a small amount of noise. A simple least squares regression on the whole dataset will poorly fit all data points, as the error accounts for both the distance to the line of inliers and outliers. To overcome this, RANSAC tries to exclude the outliers from the estimation, leading to more robust estimates. The main idea is to take a small random sub-sample of the dataset, fit the model to such data and then expand the set of points which fit such model. If the initial sub-sample mainly consists of inliers, inliers which were not included in the sub-sample will fit well the estimated model. Once such points are found, one can fit again the model on this bigger subset of data and obtain a better estimate. Repeating this simple operation for many iterations can lead to excellent results.
When using RANSAC in the domain of Trifocal tensor or Fundamental matrix estimation, one needs a good measure to estimate the error between the fitted model and the other points in the dataset. This can be achieved by using the Sampson error described in section 2.5. Algorithm 2.3 provides an overview of the main steps involved in RANSAC.

Algorithm 2.3 RANSAC Algorithm

n – Sample size
N– Maximum iterations
t – Threshold to accept data point as inlier with respect to a model
d – Minimum number of points to believe that model is fitting well the data
for i in $1 : N$ **do**
  DataSample = Sample n data points from *Data*
  ModelSample = Fit model to *DataSample*
  ConsensusPoints = *DataSample*
  for every *DataPoint* **in** *Data* **do**
    if *DataPoint* **not in** *DataSample* **and** error (*DataPoint* , *ModelSample*) $< t$ **then**
      add *DataPoint* to *ConsensusPoints*
    end if
  end for
  if size(*ConsensusPoints*) $> d$ **then**
    ImprovedModel = Fit model to *ConsensusPoints*
    Err = error(*ConsensusPoints*,*ImprovedModel*)
    if *Err* $<$ *BestErr* **then**
      BestErr = *Err*
      BestModel = *ImprovedModel*
    end if
  end if
end for
Return *BestModel*, *BestErr*, *ConsensusPoints*

Of course, RANSAC has some limitations. First of all, there is no upper bound on its convergence to the optimal model apart from the exhaustion of all possible sub-samples. If the iterations are interrupted too early, the model obtained might not fit data well at all, hence it requires some external knowledge of the problem to set correct thresholds. Moreover, the proportion of contamination of outliers in the data can challenge the algorithm. In most of our use-cases, it is sufficient to place more effort on pre-filtering the data in order to reduce as possible such contamination proportion.

3 | Problem Formulation

We provide an overview of the problem formulation and the notation used in upcoming sections. Our main goal is to recover the 3D reconstruction of a scene starting from n 2D images of the scene taken from different viewpoints.

Inputs Our inputs consist in:

- n 2D images in RGB format $\{\mathrm{I}_i\}_{i=1}^n$ such that $\mathrm{I}_i \in \mathbb{R}^{\mathrm{H}\times\mathrm{W}\times 3}\ \forall i = 1, ..., n$
- The intrinsic parameters $\{\mathrm{K}_i\}_{i=1}^n$ of the cameras from which $\{\mathrm{I}_i\}_{i=1}^n$ are taken, where $\mathrm{K}_i \in \mathbb{R}^{3\times 3}\ \forall i = 1, ..., n$

Output The output is a 3D reconstruction of the scene. While there are different formats to characterize it, in this work we focus on recovering camera pose and depth maps, which are sufficient to compute the 3D reconstruction. Specifically, the outputs are:

- a pair $(\mathrm{R}_i, \mathbf{t}_i)\ \forall\ i = 1, ..., n$ with $\mathrm{R}_i \in SO(3)$ representing the camera rotation and $\mathbf{t}_i \in \mathbb{R}^3$ being the translation vector of the camera centre.
- a depth map $\mathrm{D}_i \in [\mathrm{d}_{min}, \mathrm{d}_{max}]^{\mathrm{H}\times\mathrm{W}}$, where $\mathrm{d}_{min}, \mathrm{d}_{max}$ are the minimum and maximum depth values.

Objective Given the n images as input, we aim to reconstruct the scene by predicting camera pose $[\mathrm{R}_i|\mathbf{t}_i]$ and depth maps D_i for every image I_i provided.

Assumptions The main assumptions of the method are common to most of the SfM work. Such assumptions can be expressed in several key points:

1. Static and rigid scene: the scene being reconstructed is static and rigid, i.e. it contains no moving objects and the relative position between objects is fixed
2. Constant lighting conditions: the lighting conditions in the scene should be consistent across all images
3. Sufficient overlap between images: some portion of the scene has to be visible in different images in order to match points

4. Sufficient texture in the scene: images with extended texture-less regions provide less diversity in the extracted features and make point matching more noisy

Challenges We now list some of the main challenges that must be accounted for in the SfM task.

1. Inconsistent assumptions: it is common that the assumptions from the previous sections are rarely met. For example, we carry out some experiments on urban scenes from several different datasets. As such, it is frequent to encounter moving objects such as cars, pedestrians and buses. The movement of pedestrians in particular is also non-rigid, which makes it even more difficult to correctly estimate their depth.

2. Computational load: while there are several algorithms based on neural networks that can handle the general n-view case, increasing the number of images used in the reconstruction requires more memory. In particular, due to such computational limitations, it is common to train the neural networks in the 2-view setting and use more views only at inference time. This makes it easier to train the model with bigger batch sizes while running on the same GPU.

Figure 3.1 and 3.2 show examples of challenging scenes with dynamic objects and low overlap.

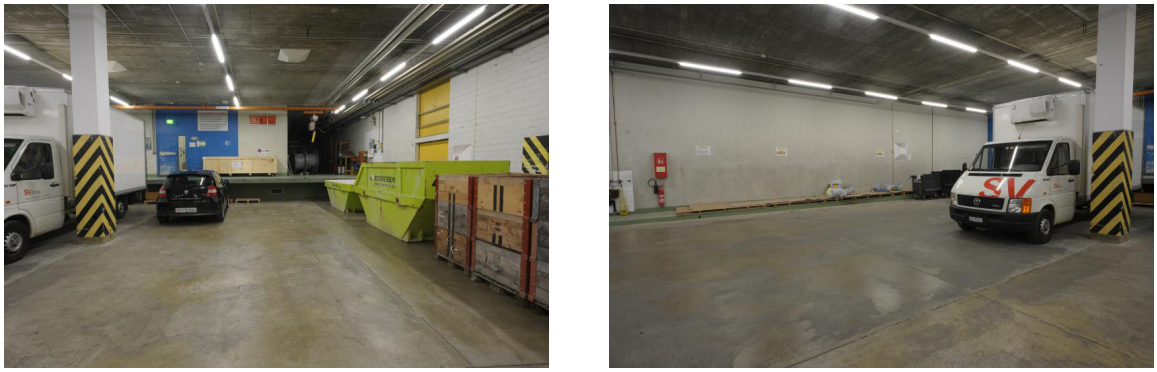


Figure 3.1: An indoor scene with low overlap. The two frames only share the column and a small part of the truck.

Figure 3.2: Two different urban scenes with dynamic objects. Cars and pedestrians are moving, the latter with non-rigid motion. Dynamic objects cause outliers both during camera pose and depth map estimation.

4 | Related Work

In recent years there has been a rapid increase in the number of techniques using neural networks to solve the 3D reconstruction problem [19]. Their performance has improved upon traditional techniques on several available benchmarks. Generally, neural network based approaches to SfM involve learning the relationships between the 2D images and the 3D structure of the scene by leveraging deep neural architectures. This involves training the neural network on a dataset of images and corresponding measured depth maps. One advantage of using neural networks for 3D reconstruction is that they can learn to handle a wide range of variations in the input data, such as changes in lighting, appearance, viewpoint and dynamic motion. Moreover, by leveraging the strong priors learnt during training, neural networks can better handle situations of uncertainty such as occlusions or modest camera translation, regularizing the problem in such degenerate configurations. This makes them well-suited for handling real-world scenes, which frequently display such features. Overall, the combination of traditional techniques and deep learning approaches can offer a powerful way to solve the SfM problem and enable the creation of high-quality 3D models from 2D images.

4.1. Taxonomy of methods

In the coming sections we illustrate how deep learning approaches to 3D reconstruction implement and expand on traditional geometric techniques. One of the primary distinctions among the available methods lies in whether they adopt a supervised or unsupervised learning approach. While the unsupervised methods have delivered impressive results, this work exclusively focuses on the supervised approaches. Figure 4.1 provides a taxonomy of recent literature.

	Limited use of geometric knowledge	Efficient use of geometric knowledge
2-view	DeMoN [14]	2-ViewRevisited [16]
n-view	MVSNet [18] DPSNet [6] DeepSfM [17]	?

Figure 4.1: Taxonomy of pipelines. Most of the pipelines proposed in literature make limited or no use of the epipolar constraints during the camera pose estimation step. 2-ViewRevisited [16] couples neural networks with the epipolar constraints, but is limited to the 2-view case. No pipelines with the same properties have been proposed for the general n-view setting.

In section 4.2, we discuss how [14] employs neural networks to estimate camera poses and depth maps in the 2-view case. We argue that the pipeline neglects the well known 3D geometric constraints and solves an ill-posed problem. In section 4.3, we demonstrate how recent pipelines have expanded [14] to the n-view case by introducing two powerful tools, namely (i) cost volumes and (ii) the plane sweep algorithm. While such pipelines can deal with the n-view case, they rely on camera poses from [14] and still try to recover the real scale of the scene, which is ill-posed. Lastly, section 4.4 provides an overview of the pipeline introduced by [16], which proposes an alternative approach to the 2-view case. This pipeline tries to couple neural networks with the 3D constraints and does not try to recover the real scale of the scene. While this provides excellent results, the pipeline is limited to the 2-view case only.

4.2. 2-view deep pose estimation

In this section we focus on 2-view pipelines for SfM. The 2-view case can be seen as the base case of n-view SfM, and the pipelines which tackle the general n-view problem usually leverage algorithms and ideas originated in the 2-view setting. In particular, we focus on the work from [14], which is a neural network based pipeline to the SfM problem. Section 4.4 presents another 2-view method. We decide to leave if for the following sections as it uses some tools which are only presented in section 4.3.

4.2.1. DeMoN

DeMoN [14] is one of the first approaches to formulate SfM as a learning problem in the domain of neural networks. The main contributions of the approach are: (*i*) propose an end-to-end differentiable neural architecture to estimate camera pose and depth maps and (*ii*) propose a neural network component which can jointly optimize the camera pose and depth map estimates through iterative application.

This approach has three major limitations: (*i*) it can only handle the 2-view case, (*ii*) it has to regress the ground truth scale and (*iii*) it does not leverage 3D geometrical knowledge. Limitation (*iii*) is given by the use of a neural network in all stages of the estimation. As such, the authors do not inject any knowledge of the epipolar constraints in the scene, nor the relationship between optical flow and camera pose. Rather, this knowledge has to be learnt by the network, which could struggle to generalise. We now provide some details on the neural artchitecture.

The pipeline is based on 3 main modules: the Boostrap net, the Iterative net and the Refinement net.

- The **Boostrap net** consists of 2 consecutive encoder-decoder blocks. The encoder-decoder blocks are obtained by stacking several convolutional layers. The first block receives as input an image pair $(\mathrm{I}_1, \mathrm{I}_2)$ and outputs an estimate of the optical flow. The second block uses the optical flow to estimate a depth map as well as the camera motion and a scale parameter s. The scale parameter is used to match the estimated depth D with the absolute ground truth scale of the scene. Indeed, the translation vector $\mathbf{t}$ and the depth values are normalised so that the network predicts a unit norm translation vector. In particular, to fully parametrize R, $\mathbf{t}$ and s, 7 parameters are output from the last layer of a dense feed-forward network - commonly referred to as Pose Net.
- The **Iterative net** follows the same architecture of the Boostrap net but is applied multiple times to optimize the camera pose and depth estimates.
- The **Refinement net** upscales the depth maps to the higher resolution of 256×192 pixels. Indeed, all computations up to this stage are carried out at a lower resolution of 64×48 pixels in order to reduce computational burden.

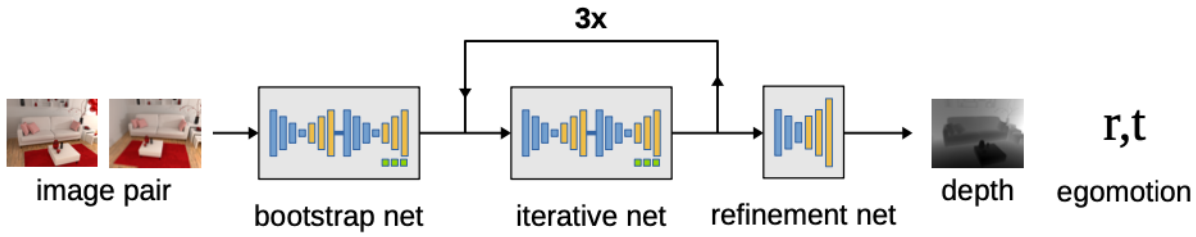


Figure 4.2: The architecture of DeMoN. Each network component contains convolutional encoder-decoder stacks. The camera pose is parametrized by the two vectors **r** and **t**. Adapted from [14].

Figure 4.2 shows an overview of the network architecture. In their work, the authors notice that forcing the network to produce an intermediate optical flow before the depth estimation greatly enhances the training procedure. This is achieved by adding a loss term which accounts for the ground truth supervision on the optical flow. Without this additional loss term, the network tends to rely on a single image to perform depth estimation, exactly as monocular estimators.

As for the optical flow, all the final and intermediate estimated objects require ground truth supervision. Since the supervision is a simple L_1 loss between the estimated value and ground truth value, the network is forced to learn the scale of the scene directly from the images through s. As previously remarked, this is an ill-posed problem which can hurt the overall performance of the network. Indeed, as it is not possible to recover the scale from the images, the network weights could end up overfitting the dataset scale, reducing the performance on unseen datasets.

The training loss is composed of a total of 7 different loss terms. This is due to the ground truth supervision of all intermediate estimates, such as the optical flow. Such losses are required to stabilise the training, but have to be correctly weighted, adding further complexity to the algorithm. The loss function for a single depth map is:

$$\mathcal{L}_{\text{depth}} = \sum_{i,j} |\xi_{gt}(i,j) - s\xi(i,j)|$$

where $\xi = 1/\text{D}$ is the estimated inverse depth. As most other methods, the depth estimates are carried out in the inverse depth space. This enables the depth estimation of points at infinity, for example the sky, which thus has inverse depth equal to 0.

4.3. Cost volumes and plane sweep

In this section, we detail two important tools to estimate camera pose and depth maps used in recent methods. The first is the plane sweep algorithm, which can be used to verify hypotheses or, stated differently, to search for an optimal value. The second is the cost volume, which is generated by the plane sweep algorithm. Indeed, the plane sweep algorithm sweeps through different values in a pre-defined range and produces a cost volume which encodes information on the goodness of a specific value. If the cost of a value is low, the value is a likely hypothesis, while if the cost is high, the value is unlikely to be a good solution of the search problem. While this definition is general and vague, we now contextualize it:

(*i*) subsection 4.3.1 discusses the general plane sweep algorithm used for depth map estimation

(*ii*) subsection 4.3.2 provides more details on the use of cost volumes for depth map estimation

(*iii*) subsection 4.3.3 discusses cost volumes applied to camera pose estimation

4.3.1. Plane sweep

The main idea behind plane sweep in the domain of depth map estimation is to find the most likely depth value for a specific pixel. Since the estimated camera pose is available, this can be carried out by comparing the position of the pixel in different views. As detailed in section 2.3.1, a point in the first view is projected along the corresponding epipolar line in the second view. The position on the epipolar line depends on the depth of the point. With this information, one can try to recover the most likely depth values. Such mechanism is now explained in detail.

Let I_1 and I_2 be two images and $\mathrm{R}, \mathbf{t}$ be the relative camera pose between them. The plane sweep algorithm can be carried out in four main steps:

1. choose a pixel $\mathbf{x}$ and sample a depth hypothesis d
2. project $\mathbf{x}$ from the 2D image I_1 to the 3D space point $\mathbf{X}$, assuming it has depth d
3. re-project $\mathbf{X}$ on I_2 at the point $\mathbf{x}'$
4. compute a cost metric as the incompatibility between the features of I_1 in $\mathbf{x}$ and the features of I_2 in $\mathbf{x}'$

Figure 4.3 provides a visualization of the projection and re-projection operations.

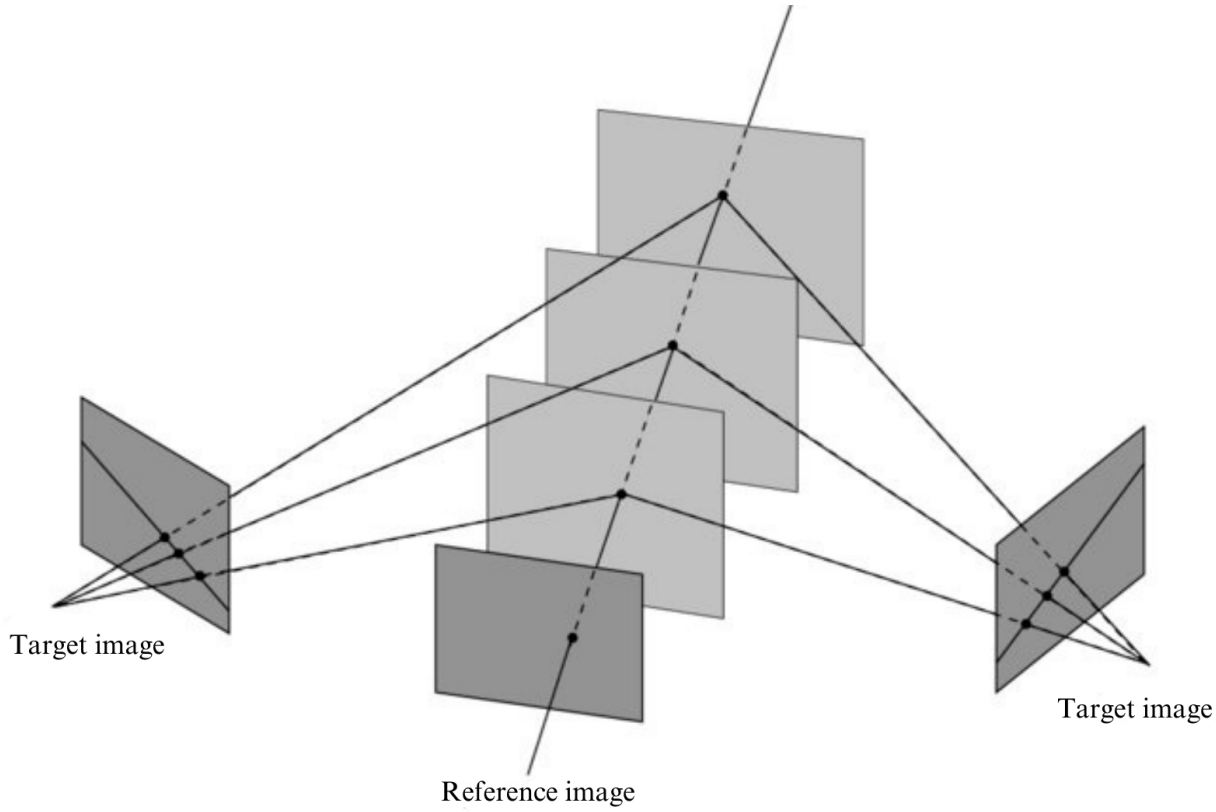


Figure 4.3: Points are projected from the reference image to fronto-parallel planes and re-projected to the target images. Adapted from [9].

By repeating this process for all the pixels in I_1 and for S depth samples, one can compute a cost volume which encodes how likely a specific pixel is to have a specific depth value. This process yields a cost volume for every couple of images. If the camera pose is known between more than two images, it is possible to repeat the process keeping I_1 fixed and aggregating the new cost volumes with the already computed ones. The projection and re-projection operations are encoded by the following equations:

$$\tilde{u} \sim \mathrm{K}\,[\mathrm{R} \mid \mathbf{t}] \begin{bmatrix} (\mathrm{K}^{-1}u)\,d \\ 1 \end{bmatrix} \tag{4.1}$$

$$\tilde{\mathrm{F}}(u) = \mathrm{F}\,(\tilde{u}) \tag{4.2}$$

where u and $\tilde{u}$ are the homogeneous coordinates of the pixel before and after the warping, F and $\tilde{\mathrm{F}}$ are the feature maps in the original view and the corresponding warped features, while d is one of the S depth hypotheses.

The same could be carried out considering the image RGB pixel values rather than a feature map F, but the result would be less robust. Indeed, feature maps can cope

better with different lighting and other perturbations as they are the output of learnt convolution filters. Moreover, as they encode semantic knowledge of the image, it makes sense to match the features rather than the direct RGB values.

Overall, the algorithm has many advantages:

- ***n*-view** It can handle the general case of n-view depth map estimation. As long as the camera poses are coherent and expressed in the same scale, it makes sense to aggregate several cost volumes to improve the depth map estimate of one specific image. The recent works that tackle the general n-view case all adopt this framework, as will be detailed in the coming sections.
- **Knowledge injection** Since the plane sweep explicitly makes use of the feature warping operation, the neural network is working on top of geometrical knowledge of the scene. This is extremely useful as the network does not need to learn to recover such relationships, which are instead provided directly. Without this knowledge, the network would have to use a portion of its parameters just to learn the feature warping operation, which would still be learnt in a statistical framework. Overall, this increases the ability of the network to generalise in unseen datasets and stabilizes the training.
- **Regularization** Once a cost volume is constructed, there are several operations which can be performed to further improve the quality of the final depth maps. Such operations are detailed in the next sections.
- **Robustness** Since the cost volume is constructed by comparing feature maps, rather than RGB values, the algorithm is more robust to changes in lighting and other perturbations. On top of this, the n-view setting allows more robustness in handling occlusions, as an object which is not visible from a camera could become visible from a different one. In general, aggregating more cost volumes tends to output better results.

The plane sweep algorithm still has some drawbacks:

- **Dynamic objects** In presence of dynamic objects, the plane sweep outputs incorrect cost volumes. This happens due to the feature warping equations. Indeed, equations 4.2 assume that the scene is rigid and warp all points in the same way. However, if a dynamic object is captured in the scene, its position differs based on the image. Hence, when a point $\mathbf{x}$ is re-projected to a different view, the corresponding pixel of the dynamic object has moved and the cost volume estimates a wrong depth.

- **Camera pose coherence** A key point is that the camera poses used in equation 4.2 must be coherent. That is, they must have a common scale. This scale can either be the real scale of the scene or any other scale which keeps the proportions of the real scene. Performing the plane sweep with non-coherent camera poses will mix incompatible cost volumes.

Algorithm 4.1 encapsulates all the steps that make up the plane sweep algorithm.

Algorithm 4.1 Plane Sweep

Reference: reference frame
Targets: target frames
Depths ← SampleDepths(S,d_{min},d_{max})
F_{ref} ← ExtractFeatures(*Reference*)
for *Target* **in** *Targets* **do**
  F_{tar} ← ExtractFeatures(*Target*)
  [R|**t**] ← ComputePose(*Reference*, *Target*)
  for d **in** *Depths* **do**
    $\tilde{F}_{tar}$ ← WarpFeatures(F_{tar},[R|**t**],d)
    $V_{couple}(d, Target)$ ← ComputeCostVolume(F_{ref},$\tilde{F}_{tar}$)
  end for
  V ← AggregateCostVolumes(V_{couple})
end for

4.3.2. Depth cost volume

In this section we provide further details about the main operations performed on cost volumes. Indeed, while the plane sweep algorithm is performed in the same way across most works in the literature, cost volumes are treated differently. First, we list the main steps a cost volume usually goes through:

- **Construction**: The first step is the construction of the cost volumes. This step is conducted through the plane sweep algorithm. The construction step outputs a 4D volume $V \in \mathbb{R}^{H \times W \times L \times S}$, where S is the number of depth hypotheses and L is the number of remaining feature channels and depends on the aggregation strategy. Aggregation is the operation which collects the reference and target feature maps. In presence of multiple target images, each pair of reference and target images produce a cost volume V_i which are then condensed in a final cost volume V. V encapsulates the depth information from all the available views.

- **Regularization**: Regularization is the process of taking an aggregated 4D cost volume and transforming it in a 3D probability volume. The regularized cost vol-

ume V is such that $V \in [0,1]^{H \times W \times S}$. For each point $\mathbf{x}$ in the image, $V(\mathbf{x})$ is the distribution of probability over the S depth samples of $\mathbf{x}$ having a specific depth.

We now compare the main strategies adopted by relevant works in the literature.

MVSNet

One of the first methods to employ cost volumes for 3D reconstruction is MVSNet [18]. The cost volume construction step is very similar to the general plane sweep algorithm illustrated in previous sections, hence we do not discuss it further. An overview of the achitecture is available in figure 4.4.

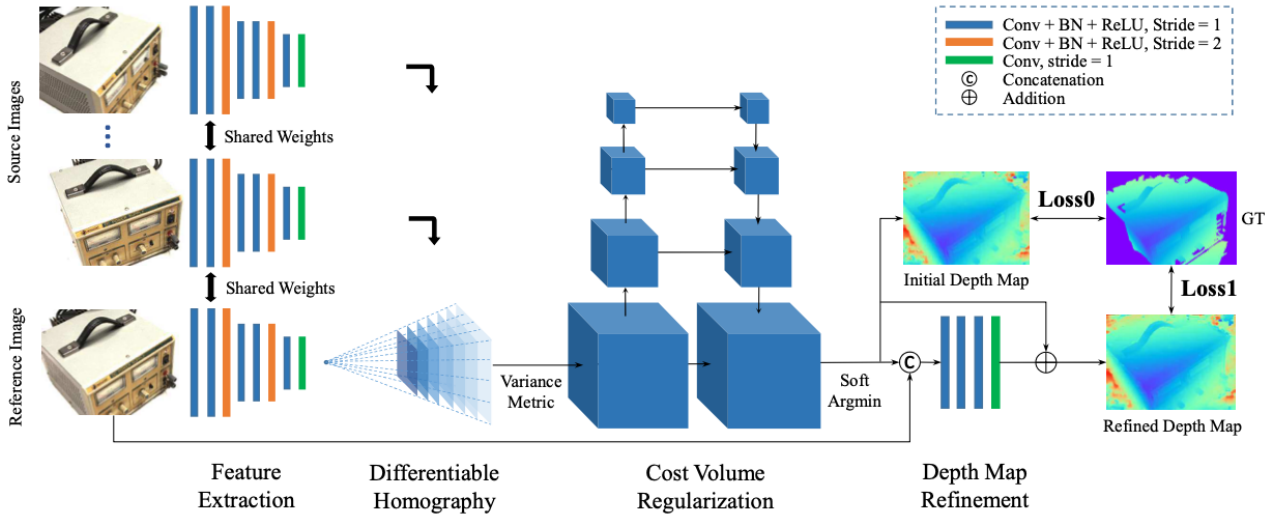


Figure 4.4: The architecture of MVSNet. The input cost volume before regularization is 4D, which is then reduced to 3D after regularization. Adapted from [18].

The main peculiarities of the work are:

- An aggregation strategy based on a variance metric
- A multi-scale regularization network
- A regression objective for the depth estimation

We now discuss the details of every choice

- **Aggregation** Cost volume aggregation is carried out by using a variance metric $\mathcal{M}$ that maps multiple feature maps into one cost volume:

$$\mathcal{M} : \underbrace{\mathbb{R}^{H \times W \times Ch \times S} \times \cdots \times \mathbb{R}^{H \times W \times Ch \times S}}_{n} \rightarrow \mathbb{R}^{H \times W \times Ch \times S}$$

such that the aggregated 4D cost volume V is:

$$\mathrm{V} = \mathcal{M}\left(\mathrm{F}_1, \cdots, \mathrm{F}_n\right) = \frac{\sum_{i=1}^{n}\left(\mathrm{F}_i - \overline{\mathrm{F}_i}\right)^2}{n} \tag{4.3}$$

where F_i are the feature volumes warped from source to target view and $\overline{\mathrm{F}_i}$ is the average feature volume. All the operations carried out are point-wise. By using this cost metric, the authors measure the multi-view feature difference. Previous methods such as [4] instead leveraged the mean operation to aggregate feature volumes, loosing such information.

- **Regularization** The regularization is carried out by using multi-scale 3D convolutions on the 4D cost volume, regressing an initial depth map which is then refined. The regularization network is a 3D extension of UNet [11], with an encoder-decoder structure based on convolutions. Neighboring information is aggregated from a large receptive field without excessive memory and computation cost. Another strategy to reduce the computational burden is to reduce the input feature map dimension from 32-channel to 8-channel after the first 3D convolutional layer. The last convolutional layer outputs a 1-channel volume so that the cost volume is reduced from 4D to 3D. Finally, softmax is applied along the depth direction to obtain a probability distribution $\mathrm{V}(\mathbf{x}, d)$ over depth hypotheses for every pixel $\mathbf{x}$.

- **Depth regression** Rather than performing classification on the depth hypotheses using the computed probability distribution, the authors proceed to compute a weighted regression. The weights are extracted from the probability volume. Indeed, given the probability distribution over depth hypotheses, the best estimate would correspond to the *argmax* operation. However, this would break the differentiability of the network. Given this observation, the authors prefer instead to compute the expected value over such distribution:

$$\mathrm{D}(\mathbf{x}) = \sum_{d=d_{min}}^{d_{max}} d\ \mathrm{V}(\mathbf{x}, d) \tag{4.4}$$

where depth d is weighted by the corresponding value of the probability volume $\mathrm{V}(\mathbf{x}, d)$. This estimate is pixel-wise, meaning it is repeated for each pixel, and has the advantage of being a continuous value rather than being a sample from the discrete uniform distribution of depth hypotheses.

The authors also show that the output probability distribution is a useful tool to gauge the quality and reliability of the estimates. Indeed, a single-modal distribution is generally

associated to correct pixel matches, while multi-modal distributions can be used to flag falsely matched pixels. The last step of the pipeline is to refine depth estimates to improve estimation, specifically in boundary regions. A residual network takes care of transforming the initial depth estimate $\mathrm{D}_0(\mathbf{x})$ into a refined depth $\mathrm{D}_r(\mathbf{x})$.

The training loss for a single depth map is defined as:

$$\mathcal{L} = \sum_{\mathbf{x} \in \mathbf{x}_{\text{val}}} \underbrace{\left\| \mathrm{D}_{gt}(\mathbf{x}) - \mathrm{D}_0(\mathbf{x}) \right\|_1}_{\text{Loss 0}} + \underbrace{\left\| \mathrm{D}_{gt}(\mathbf{x}) - \mathrm{D}_r(\mathbf{x}) \right\|_1}_{\text{Loss 1}} \tag{4.5}$$

where $\mathbf{x}_{\text{val}}$ are the valid pixels (ground truth depth maps are not always complete) and $\mathrm{D}_{gt}(\mathbf{x})$ the ground truth depth in pixel $\mathbf{x}$. By comparing this loss with the one used in *DeMoN* [14] it is possible to notice another advantage provided by the cost volume architecture. Indeed, this network requires far less supervision signals, while *DeMoN* required not only the final depth maps but also intermediate supervision. This approach is more aligned with the end-to-end training framework which has commonly showed excellent performance in the majority of deep learning tasks.

While the results improve upon previous methods, some points must be noted.

- The network uses ground truth camera poses as input, hence only performs the depth estimation task. Having a correct camera pose helps to obtain quality depth maps both in training and inference.

During training, the authors choose to employ $n = 3$ views, while other works only use $n = 2$ during training.

DPS

Deep Plane Sweep Stereo Net, or *DPSNet* [6], is a concurrent effort to MVSNet [18]. While the approaches are very similar, [6] proposes:

- A different aggregation strategy based on feature concatenation, rather than computing a cost metric directly on the feature volumes. The resulting volume V is such that $\mathrm{V} \in \mathbb{R}^{\mathrm{H} \times \mathrm{W} \times 2\mathrm{Ch} \times \mathrm{S}}$, with Ch being the dimension of feature channels extracted for each image. In presence of multiple target images, the cost volumes can be further aggregated with a simple average operation.
- A cost volume refinement strategy which employs a so called *Context Network* that tries to preserve the edges and contours of the image. By feeding both the cost volume and the original feature maps in a 2D convolutioanl network, authors can

partially prevent the depth maps from loosing sharpness near edges. Such operation is carried out using a residual connection and is performed on every slice of the cost volume, where the slicing is performed along dimension of the S depth planes.

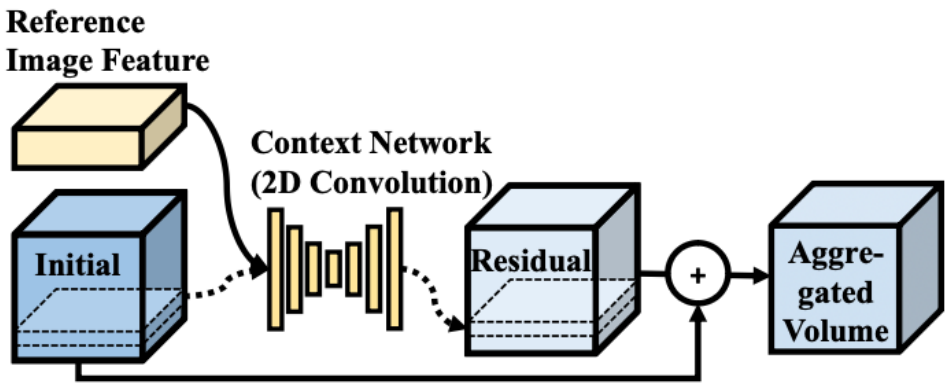


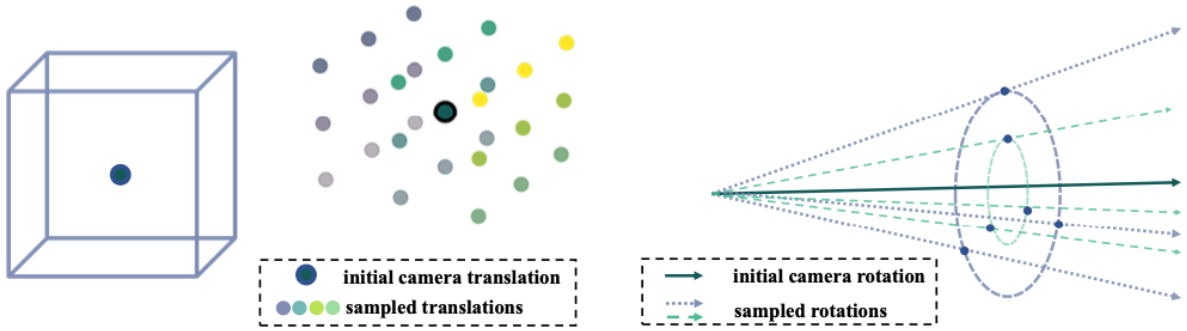
Figure 4.5: An overview of the Context network. Adapted from [6].

During training, the authors only use $n = 2$ views to reduce the computational load, but the inference can be carried out with any n. Also in this case, the authors leverage the ground truth pose to skip the camera pose estimation step.

4.3.3. Pose cost volume

A very interesting proposal by [17] is to expand the cost volume and plane sweep tools to the camera pose estimation. This can further inject physical world knowledge in the neural network, with all the benefits previously discussed. The main contributions of this work are:

- Introduce a cost volume for camera pose estimation
- Connect the pose cost volume and depth cost volume to emulate the traditional Bundle Adjustment algorithm, iteratively and jointly optimizing camera pose and depth map



(a) Camera translation sampling (b) Camera rotation sampling

Figure 4.6: Pose samples centered around the initial rotation matrix and translation vector. Adapted from [17].

We now discuss each of the contributions separately:

- **Pose cost volume** As the plane sweep sampled depth hypotheses to generate the depth cost volumes, pose cost volumes are built by sampling camera positions centered on an initial camera pose estimate. Let $[\mathrm{R}|\mathbf{t}]$ be an initial camera pose. To obtain new rotation matrices, the authors sample a rotation matrix R_{hyp} uniformly in the Euler angle space in a predefined range, then use it to perturb the initial matrix through $\mathrm{R}^* = \mathrm{R}_{hyp}\mathrm{R}$. To obtain new translation vectors, the authors sample a translation vector $\mathbf{t}_{hyp}$ uniformly around $(0, 0, 0)$ and use it to perturb the initial translation through $\mathbf{t}^* = \mathbf{t}_{hyp}+\mathbf{t}$. Following this procedure, a group of P new camera poses $[\mathrm{R}_p^*|\mathbf{t}_p^*]_{p=1}^{\mathrm{P}}$ around the input camera pose are obtained and can be used for the construction of camera pose cost volume. As an initial estimate is necessary, the authors use poses estimated from [14] to initialize the algorithm. The pose cost volume can be generated by warping the features following a similiar homography to that of 4.4:

$$\tilde{u} \sim \mathrm{K}\left[\mathrm{R}_p^* \mid \mathbf{t}_p^*\right] \begin{bmatrix} (\mathrm{K}^{-1}u)\,\mathrm{D} \\ 1 \end{bmatrix} \tag{4.6}$$

  where D is the initial estimate of the depth map. Figure 4.6 provides a visualization of the camera pose hypotheses sampling.

- **Iterative optimization** Since the cost volume construction requires initial estimates of camera pose and depth maps, the network can be applied multiple times in iterative fashion. By using previous outputs as new inputs, the network can converge to good estimates starting from relatively inaccurate inputs. Moreover, since the pose cost volume is estimated using the depth maps, this connects the optimization of the pose with that of the depth. Hence, running the network iteratively

allows for a joint optimization of the two estimates, rather than a disjoint process.

The training of the network is performed in the 2-view setting only, as [6]. In this case, the authors set as loss function:

$$\mathcal{L} = \lambda_r \mathcal{L}_{rotation} + \lambda_t \mathcal{L}_{translation} + \lambda_d \mathcal{L}_{depth}$$

with

$$\mathcal{L}_{depth} = \sum_i H\left(\mathrm{D}_{0,i}, \mathrm{D}_{gt,i}\right) + H\left(\mathrm{D}_i, \mathrm{D}_{gt,i}\right)$$

where $\mathrm{D}_{gt,i}$ is the i^{th} ground truth depth, $\mathrm{D}_{0,i}$ the non refined estimated depth, D_i the refined estimated depth and H being the Huber loss function. The rotation and translation losses are instead obtained with the usual $\mathcal{L}_1$ distance between the ground truth and estimated values. All values for λ are determined experimentally.

Overall, the proposed framework is very interesting as it further incorporates domain knowledge in the network, obtaining better generalisation. However, the computational burden of building two cost volumes through plane sweep are very high. Moreover, several iterations are needed to reach convergence. This can be traced back to the quality of the initial camera pose and depth estimates, which are taken from [14]. We previously argued that such method has some limitations, for example the scale bias and the limited geometrical knowledge. As a consequence, integrating such estimates in the pipeline requires a big computational effort to get better measurements. The authors show than on average, convergence can be reached in about 4 to 6 iterations. We thus strongly believe that incorporating a better camera pose estimation technique as initialization can at least reduce computational burden and potentially even improve results.

4.4. Revisiting 2-view deep pose estimation

We now discuss the work from [16], which proposes a novel mixed approach to estimate camera pose for the 2-view SfM problem. The main contribution of this work is to wisely balance the use of neural networks and traditional SfM pipeline tools. The proposed pipeline involves the following steps:

1. Estimate optical flow using a neural network.
2. Retrieve a normalized camera pose from the optical flow using the Essential matrix E.
3. Use plane sweep in a scale-invariant framework.

Figure 4.7 provides an overview of the pipeline.

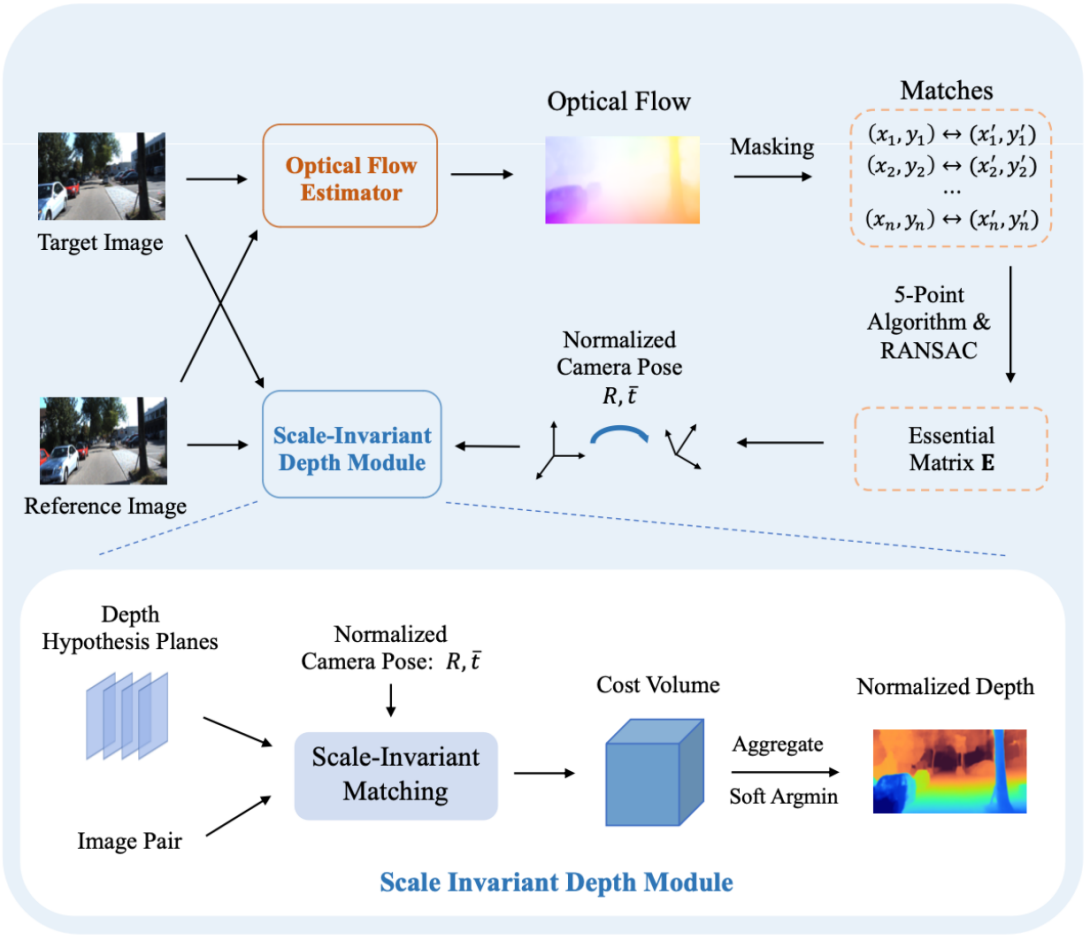


Figure 4.7: An overview of the pipeline. Optical flow is extracted and filtered using SIFT. Adapted from [16].

We discuss the main contributions of this work:

Optical flow The proposed pipeline leverages neural networks in a task that they excel at. Since neural networks have a high level of semantic understanding of images, the neural network optical flow estimators prove to be very effective. To this end, the authors use DICL [15], which is a state-of-the-art optical flow estimator. The authors note that SIFT [8] keypoint locations can be used to produce a mask and discard the noisy point matches from DICL. If a better optical flow estimator is introduced in the future, it can be easily swapped out and substituted, with no impact on the pipeline and without needing to retrain the networks.

Essential matrix Once the optical flow is recovered, the authors use the equations that describe the epipolar geometry to extract the camera pose. The tools described

in 2.3.1 are used to estimate E and recover the camera pose from it. Since the translation vector can only be recovered in direction and not in scale, the authors set $\|\mathbf{t}\| = 1$. This approach is expected to perform better as it directly uses the epipolar geometry knowledge, rather than trying to learn it through a neural network. This is in contrast to [14], which uses a neural network to regress the camera pose from the optical flow. Learning a relationship through neural network training when it is already available in a simple equation is unnecessary and can potentially degrade performance.

Scale invariant module By using a normalized camera pose as input to the plane sweep, the output depth map D is also normalised. The authors then rescale the depth map to match the ground truth, thereby obtaining the final depth map output. Setting $\alpha = \|\mathbf{t}\|$, αD matches the scale of the ground truth. By using a normalised camera pose in the plane sweep algorithm, the distribution of depth hypotheses is not distorted by the scale of $\mathbf{t}$, which instead is fixed to 1. The authors refer to this as scale invariant depth module.

The proposed algorithm tackles the weakness of previous deep pose estimation algorithms. In this case, the authors focus on solving a well-posed problem, rather than focusing on predicting the ground truth scale. There remain two limitations of the method:

- By computing E from the optical flow, the pipeline ceases to be differentiable end-to-end. Instead, the optical flow estimator has to be trained separately from the cost volume networks.
- The method is inherently two-view. As the pose extracted from E has unknown scale and is only pairwise, there is no way to even recover the relative position between more than two cameras. This is different from [14], which instead recovers the absolute scale of $\mathbf{t}$.

5 | Proposed Solution

In this chapter we provide an overview of the pipeline we propose to extend [16] to the general n-view case. Section 5.1 discusses the motivations behind our proposal. Section 5.2 illustrates our proposed pipeline, detailing the main steps and algorithmic choices. Section 5.3 provides specific details on our implementation. Section 5.4 highlights the main advantages and limitations of our pipeline.

5.1. Motivation

In this section we provide the motivations underlying our proposed pipeline.
Our objective is to expand the pipeline proposed in [16] to the n-view setting. The advantage of using multiple target images $\{\mathrm{I}_{tar_i}\}_{i=1}^{n-1}$ to estimate the depth map D of the reference image I_{ref} is that the noise can be decreased and occlusions can be better handled. Indeed, by gathering depth information from multiple target images, bad estimates coming from a single target image due to variable lighting conditions and object occlusion are less relevant.

Since [16] uses the Fundamental matrix F, the authors are unable to leverage more than 2-views. Indeed, all scale information between different camera pose estimates $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^{n}$ is lost as each translation vector $\mathbf{t}_i$ is such that $\|\mathbf{t}_i\| = 1$. This happens because the Fundamental matrix F is limited to 2-views and cannot provide information on the relative scale between different camera poses. Such poses cannot be used in the plane sweep algorithm with $n > 2$ as they distort the scene geometry. By estimating the camera poses through the Trifocal tensor T we can instead collect relative scale information between 3 images and recover all the camera poses $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^{n}$ in a common scale. This preserves the original proportions and geometry of the scene, hence it makes sense to use the plane sweep also with $n > 3$.

On top of this, we maintain the coupling of neural networks and epipolar constraints. This provides domain knowledge to the neural network and leads to better camera pose estimates. We achieve this by using neural networks to estimate the optical flows, which

provide the point matches used to estimate the Trifocal tensors T_i. Finally, the camera poses $[\mathrm{R}_i|\mathbf{t}_i]$ are recovered from T_i.

This is advantageous with respect to [17] as we do not require multiple expensive iterations to refine an initial noisy camera pose estimate.

5.2. Proposed pipeline

We now provide an overview of our pipeline, while the next sections dive deeper on our specific proposals. The pipeline is composed of three main steps:

1. **3-view point matching** We start off by selecting $n-2$ triplets which are used to estimate $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^n$. For every triplet $(\mathrm{I}_a, \mathrm{I}_b, \mathrm{I}_c)$, we recover 3-view point matches $\mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{x}''$ by estimating the optical flows O_{ab} and O_{bc} between the couples $(\mathrm{I}_a, \mathrm{I}_b)$ and $(\mathrm{I}_b, \mathrm{I}_c)$.

2. **Pose estimation** Using the 3-view point matches $\mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{x}''$ we estimate the Trifocal tensor T. The camera poses $[\mathrm{R}_{ba}|\mathbf{t}_{ba}]$ and $[\mathrm{R}_{bc}|\mathbf{t}_{bc}]$ are extracted from T. We then rescale and chain the camera pose estimates obtained in different triplets to recover the n-view normalised camera poses $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^n$ with a shared scale.

3. **Depth estimation** Finally, the plane sweep algorithm is carried out by using the normalised camera poses $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^n$. Even though the camera configuration is normalised, it preserves the original geometry of the scene and the plane sweep algorithms outputs a normalised depth map D for the reference frame I_{ref}.

Figure 5.1 provides a summary of the pipeline in the case $n = 4$.

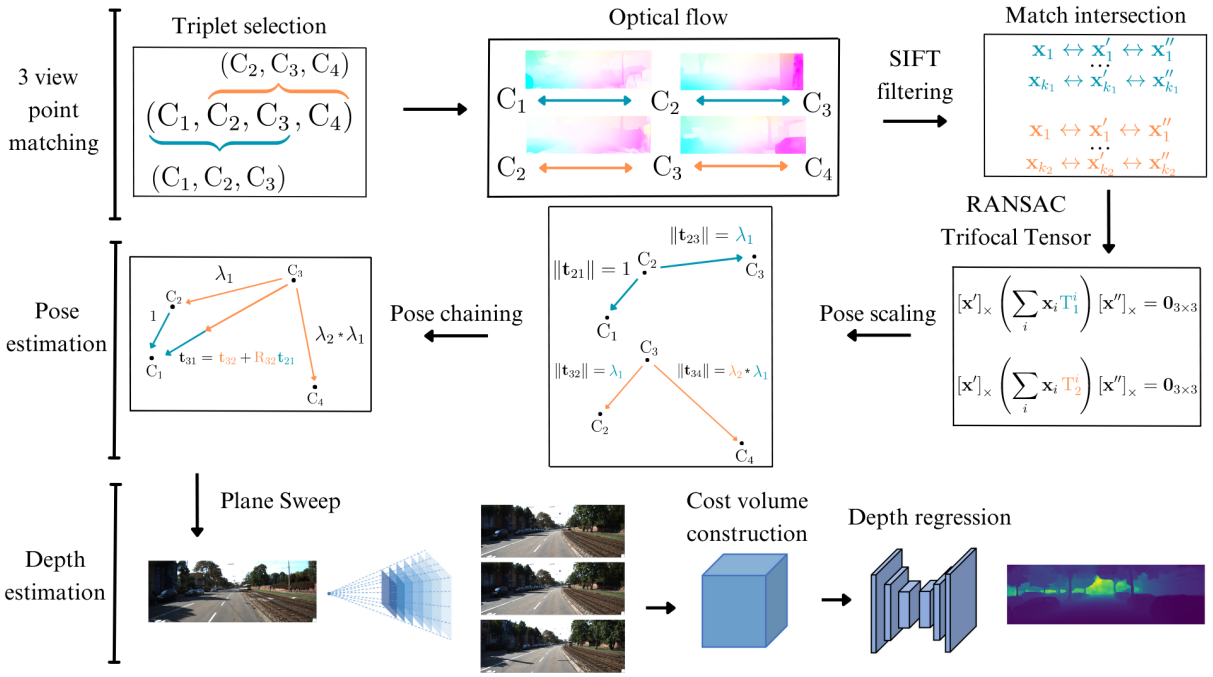


Figure 5.1: Our proposed pipeline when $n = 4$. The pipeline is a sequence of three main steps: 3-view point matching, pose estimation and depth estimation.

In the following sections we dive deep into the details of the steps.

5.2.1. 3-view point matching

The inputs to this step are a sequence of n ordered cameras $(\mathrm{C}_1, ..., \mathrm{C}_n)$ and the corresponding images $(\mathrm{I}_1, ..., \mathrm{I}_n)$. The outputs are a set of $n-2$ overlapping triplets $\{(\mathrm{C}_{a_i}, \mathrm{C}_{b_i}, \mathrm{C}_{c_i})\}_{i=1}^{n-2}$ and a set of 3-view point matches $\{\{\mathbf{x}_j \leftrightarrow \mathbf{x}'_j \leftrightarrow \mathbf{x}''_j\}_{j=1}^{k_i}\}_{i=1}^{n-2}$ for each triplet, where k_i is the number of matching points in the i-th triplet.

The triplet selection strategy is an important part of our pipeline. We propose two different strategies that output overlapping triplets. Given two triplets, we consider them as overlapping when they share 2 out of the 3 cameras. We look for overlapping triplets as it prevents the relative scale information from going lost. For example, let us consider two **non** overlapping triplets $(\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3)$ and $(\mathrm{C}_4, \mathrm{C}_5, \mathrm{C}_6)$. The Trifocal tensor will return four camera poses such that $\|\mathbf{t}_{21}\| = 1, \|\mathbf{t}_{54}\| = 1$, $\|\mathbf{t}_{23}\| = \lambda_1$ and $\|\mathbf{t}_{56}\| = \lambda_2$. The main problem is that $\lambda_1 > \lambda_2$ does not always imply that $\|\mathbf{t}_{gt,23}\| > \|\mathbf{t}_{gt,56}\|$, where $\mathbf{t}_{gt,23}$ and $\mathbf{t}_{gt,56}$ are the ground truth translation vectors expressed in absolute scale, for example meters. Figure 5.2 provides an example of this issue.

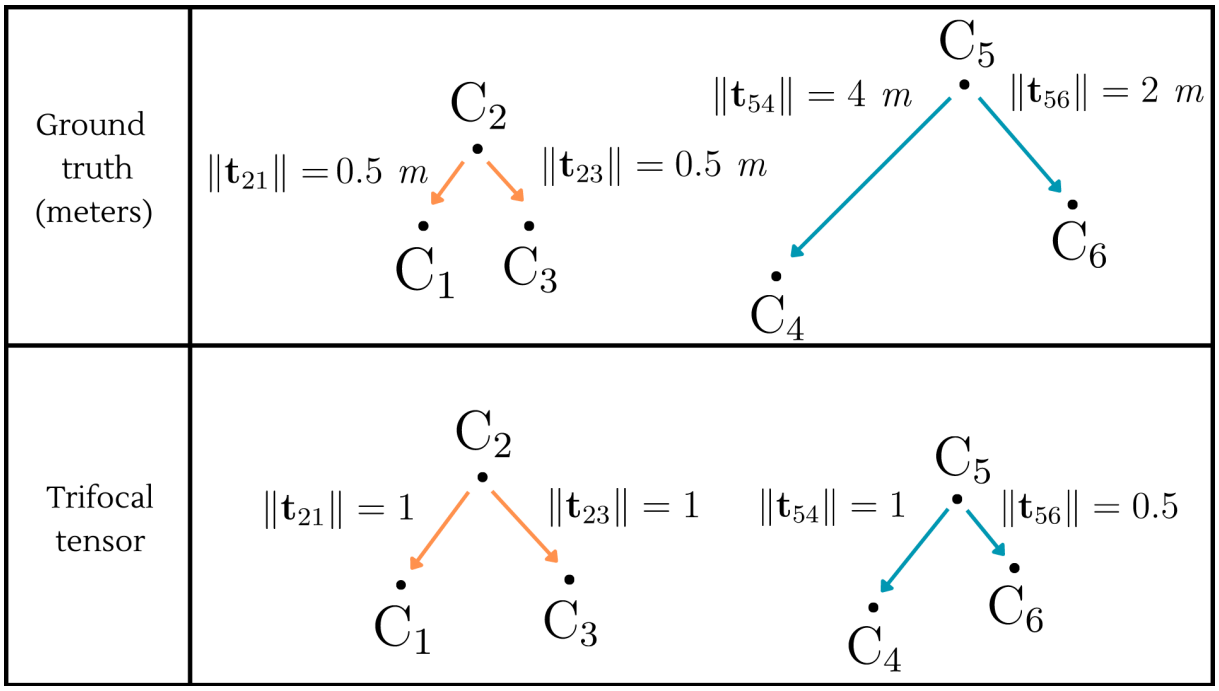


Figure 5.2: Ground truth measurements are in the same scale, i.e. meters. Trifocal tensor estimates instead are in different scales. Each translation is normalised with respect to the translation between the first two cameras of the triplet, namely $\mathbf{t}_{21}$ and $\mathbf{t}_{54}$. It does not make sense to compare their relative scales, for example $\lambda_1 > \lambda_2$ but $\|\mathbf{t}_{gt,23}\| < \|\mathbf{t}_{gt,56}\|$.

This happens because both λ_1 and λ_2 are expressed in different scales, rather than the same scale. By choosing two overlapping triplets instead we will estimate the same translation vector twice and can recover the proportions between such scales.
Once the overlapping triplets are available, we need to find point matches over 3 views. These will be used in the next step of the pipeline to estimate T. Since the optical flow can only provide point matches over 2 views, we propose to intersect the point matches between $(\mathrm{I}_{a_i}, \mathrm{I}_{b_i})$ and $(\mathrm{I}_{b_i}, \mathrm{I}_{c_i})$.

The 3-view point matching step of our pipeline can be subdivided in:

1.1 **Triplet selection** The triplets are selected following one of our proposed triplet selection strategies, namely (i) the *sliding window* strategy or (ii) the *pivot* strategy. Both strategies provide overlapping triplets. The sliding window strategy slides over the sequence of cameras $(\mathrm{C}_1, ..., \mathrm{C}_n)$ with a window size of 3 and a stride of 1. The obtained triplets thus are: $\{(\mathrm{C}_i, \mathrm{C}_{i+1}, \mathrm{C}_{i+2})\}_{i=1}^{n-2}$. The pivot strategy instead keeps a two fixed cameras and slides over the rest. For example, one can keep C_1 and C_2 as fixed cameras. This generates the triplets $\{(\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_i)\}_{i=3}^{n}$. While this strategy has the advantage of already rescaling all camera poses with respect to $\mathbf{t}_{21}$, it requires

to compute poses between cameras that have a bigger index distance. As the frames are ordered, computing the camera pose between C_2 and C_4 (index distance 2) in general will yield higher errors than when computing the relative pose between C_2 and C_1 (index distance 1). This holds as there is a wider baseline between the images, making the task of matching points more complex.

1.2 **Optical flows** Two optical flows O_{a_i,b_i} and $O_{b_ic_i}$ are computed between the neighbouring couples of cameras (I_{a_i}, I_{b_i}) and (I_{b_i}, I_{c_i}). This outputs two sets of matching points $\{\mathbf{x}_j \leftrightarrow \mathbf{x}'_j\}_{j=1}^{k'_{ab}}$ and $\{\mathbf{x}'_l \leftrightarrow \mathbf{x}''_l\}_{l=1}^{k'_{bc}}$. As in [16], we use DICL [15], a neural network trained to output optical flows.

1.3 **SIFT filtering** SIFT keypoint locations filter the noisy matches $\{\mathbf{x}_j \leftrightarrow \mathbf{x}'_j\}_{j=1}^{k'_{ab}}$ and $\{\mathbf{x}'_l \leftrightarrow \mathbf{x}''_l\}_{l=1}^{k'_{bc}}$, reducing the point matches to $\{\mathbf{x}_j \leftrightarrow \mathbf{x}'_j\}_{j=1}^{k_{ab}}$ and $\{\mathbf{x}'_l \leftrightarrow \mathbf{x}''_l\}_{l=1}^{k_{bc}}$, where $k_{ab} < k'_{ab}$ and $k_{bc} < k'_{bc}$.

1.4 **Match intersection** To recover the 3-view point matches, the two sets of matching points are intersected. The point match intersection yields the 3-view point matches $\{\{\mathbf{x}_j \leftrightarrow \mathbf{x}'_j \leftrightarrow \mathbf{x}''_j\}_{j=1}^{k_i}\}_{i=1}^{n-2}$, where the point match intersection is defined as:

$$\mathbf{x}_j \leftrightarrow \mathbf{x}'_j \wedge \mathbf{x}'_l \leftrightarrow \mathbf{x}''_l = \begin{cases} \mathbf{x}_j \leftrightarrow \mathbf{x}'_j \leftrightarrow \mathbf{x}''_j & \text{if } \mathrm{x}'_j = \mathbf{x}'_l \\ \emptyset & \text{otherwise} \end{cases}$$

5.2.2. Pose estimation

The inputs to this step are a set of $n-2$ overlapping triplets $\{(C_{a_i}, C_{b_i}, C_{c_i})\}_{i=1}^{n-2}$ and a set $\{\{\mathbf{x}_j \leftrightarrow \mathbf{x}'_j \leftrightarrow \mathbf{x}''_j\}_{j=1}^{k_i}\}_{i=1}^{n-2}$ of 3-view point matches for each triplet. The outputs are a sequence of relative camera poses $[R_i|\mathbf{t}_i]_{i=1}^{n}$ between the reference frame I_{ref} and the target frames $\{I_{tar_i}\}_{i=1}^{n-1}$.

Given the overlapping triplets $\{(C_{a_i}, C_{b_i}, C_{c_i})\}_{i=1}^{n-2}$, we can now estimate the Trifocal tensor T_i in every triplet. As mentioned in 5.2.1, the camera poses $[R_i|\mathbf{t}_i]$ extracted from T_i are expressed in different scale for every i-th triplet. To perform the n-view plane sweep algorithm however, the camera poses have to share a common scale. If this condition is not met, the 3D geometry of the scene is distorted. Our proposal is to rescale all camera poses $[R_i|\mathbf{t}_i]$ with respect to the scale of $\mathbf{t}_1$. This can be achieved thanks to our choice of overlapping triplets.

Once the camera poses are expressed in the same scale, it is possible to recover the camera pose between the reference frame I_{ref} and each target frame $\{I_{tar_i}\}_{i=1}^{n-1}$. We propose to achieve this by chaining the neighbouring camera poses.

The pose estimation step of our pipeline can be subdivided in:

2.1 **RANSAC Trifocal tensor** The Trifocal tensor is estimated in each triplet from the 3-view correspondences $\mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{x}''$. We propose to estimate T following the approach discussed in section 2.4.5. Given a triplet $(\mathrm{C}_a, \mathrm{C}_b, \mathrm{C}_c)$, we estimate two Fundamental matrices F_{ab} and F_{bc} with the RANSAC algorithm. The relative scale of $\mathbf{t}_{bc}$ is then computed through the closed form solution of

$$\arg\min_{\lambda \in \mathrm{R}} \sum_{k=1}^{N} \left\| \mathbf{x}''_k \times \left(\mathrm{K} \left(\mathrm{R}_{ba} \mathbf{X}_k + \lambda \frac{\mathbf{t}_{bc}}{\|\mathbf{t}_{bc}\|} \right) \right) \right\|^2$$

2.2 **Pose scaling** Given two overlapping triplets $(\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3)$ and $(\mathrm{C}_2, \mathrm{C}_3, \mathrm{C}_4)$ obtained with sliding window strategy, we rescale translation vectors from $(\mathrm{C}_2, \mathrm{C}_3, \mathrm{C}_4)$ to match the scale of $(\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3)$. Let $\mathbf{t}_{23}^{1,2,3}$ be the translation between C_2 and C_3 computed in triplet $(\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3)$ and $\mathbf{t}_{32}^{2,3,4}$ the translation between C_3 and C_2 computed in triplet $(\mathrm{C}_2, \mathrm{C}_3, \mathrm{C}_4)$. Since we have estimated the translation vector $\mathbf{t}_{23}$ in two different triplets, we can match the scale of $\mathbf{t}_{32}^{2,3,4}$ to that of $\mathbf{t}_{23}^{1,2,3}$. Indeed,

$$\lambda_1 \mathbf{t}_{32}^{2,3,4} \approx \mathbf{t}_{32}^{1,2,3}$$

since

$$\|\mathbf{t}_{23}^{1,2,3}\| = \lambda_1$$
$$\|\mathbf{t}_{32}^{2,3,4}\| = 1$$

Therefore, we can conclude that $\lambda_1 \mathbf{t}_{32}^{2,3,4}$ and $\lambda_1 \mathbf{t}_{34}^{2,3,4}$ match the scale of the previous triplet. Figure 5.3 shows the pose scaling between two triplets.

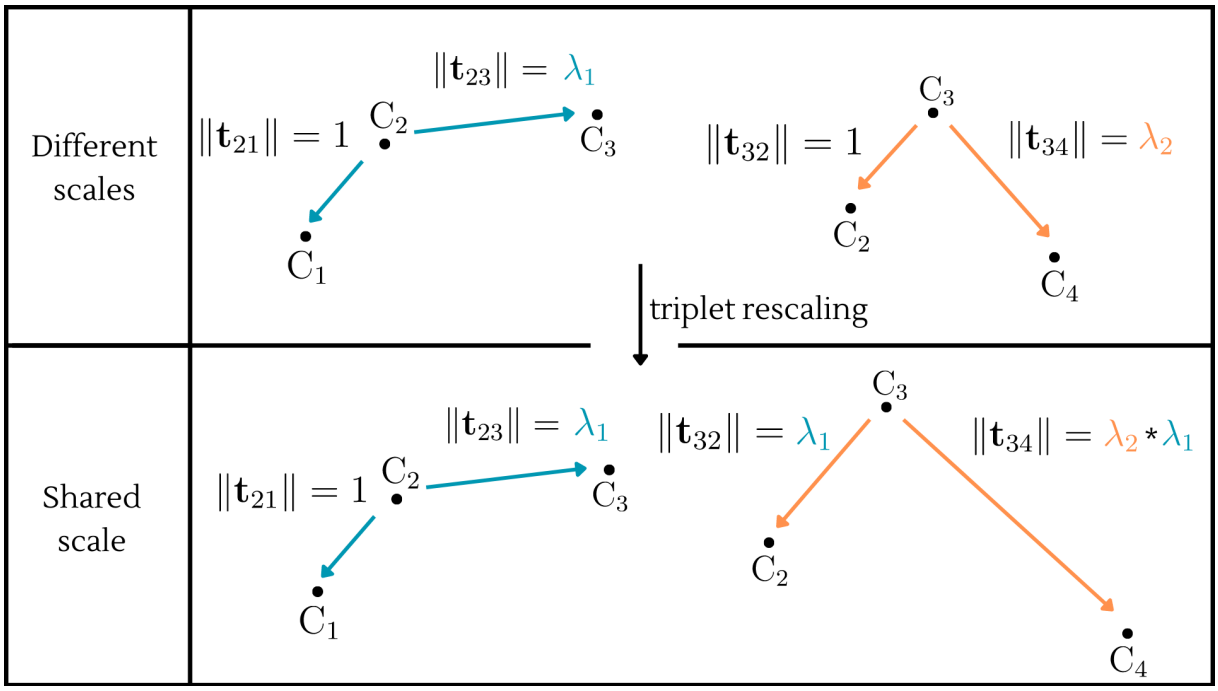


Figure 5.3: Two overlapping triplets before and after their scales are matched. Since $\mathbf{t}_{23}$ is estimated twice in different scales, we can match the scale of $\mathbf{t}_{32}^{2,3,4}$ to that of $\mathbf{t}_{23}^{1,2,3}$.

As a result, the camera poses $[\mathrm{R}_{21}^{1,2,3}|\mathbf{t}_{21}^{1,2,3}]$, $[\mathrm{R}_{23}^{1,2,3}|\mathbf{t}_{23}^{1,2,3}]$, $[\mathrm{R}_{34}^{2,3,4}|\lambda_1\mathbf{t}_{34}^{2,3,4}]$ share a common scale that preserves the proportions induced by the real scene. This is achieved even if the scale does not match the ground truth scale (e.g. meters). The key is that the proportions of the real 3D scene are preserved without any distortion. We can now drop the triplet notation and refer to the camera poses as $[\mathrm{R}_{21}|\mathbf{t}_{21}]$, $[\mathrm{R}_{23}|\mathbf{t}_{23}]$, $[\mathrm{R}_{34}|\lambda_1\mathbf{t}_{34}]$.

2.3 **Pose chaining** The camera poses computed in step 2.2 are a sequence of relative poses between neighbouring cameras. The plane sweep algorithm requires the camera pose between the reference frame I_{ref} and each target frame $\{\mathrm{I}_{tar_i}\}_{i=1}^{n-1}$. Let image I_3 be the reference frame. Then, the plane sweep requires $[\mathrm{R}_{31}|\mathbf{t}_{31}]$, $[\mathrm{R}_{32}|\mathbf{t}_{32}]$, $[\mathrm{R}_{34}|\lambda_1\mathbf{t}_{34}]$. We are thus missing: $[\mathrm{R}_{31}|\mathbf{t}_{31}]$ and $[\mathrm{R}_{32}|\mathbf{t}_{32}]$.
We can recover $[\mathrm{R}_{32}|\mathbf{t}_{32}]$ by inverting the pose $[\mathrm{R}_{23}|\mathbf{t}_{23}]$, which we already computed. Similarly, we can also recover $[\mathrm{R}_{31}|\mathbf{t}_{31}]$. This can be achieved by chaining available poses. In this case, we can chain the pose $[\mathrm{R}_{32}|\mathbf{t}_{32}]$ to $[\mathrm{R}_{21}|\mathbf{t}_{21}]$. As before, $[\mathrm{R}_{32}|\mathbf{t}_{32}]$ is obtained by inverting $[\mathrm{R}_{23}|\mathbf{t}_{23}]$. The inversion and chaining transformations are defined in section 2.1.1. Figure 5.4 shows the chaining of camera poses from two different triplets with matching scales.

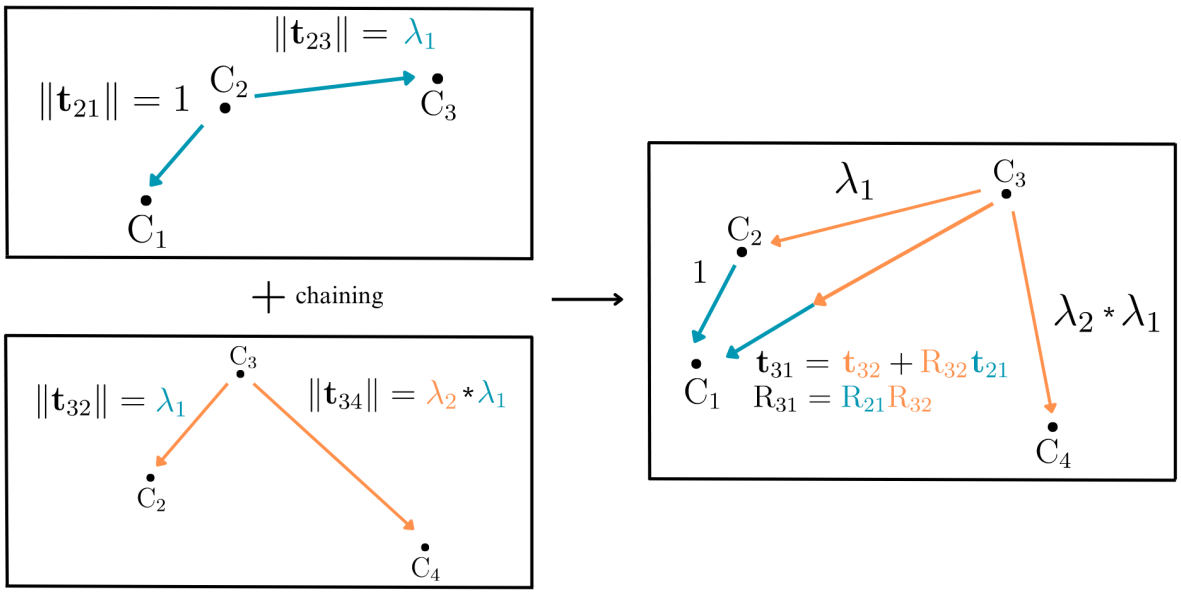


Figure 5.4: Neighbouring camera poses are chained to obtain camera pose between reference and target frames. $[\mathrm{R}_{31}|\mathbf{t}_{31}]$ is computed by chaining $[\mathrm{R}_{32}|\mathbf{t}_{32}]$ to $[\mathrm{R}_{21}|\mathbf{t}_{21}]$.

5.2.3. Depth estimation

The inputs to this step are a sequence of relative camera poses $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^n$ between the reference frame and the target frames. The outputs are a depth map D for the reference frame I_{ref}, obtained by leveraging the $n-1$ target images. The depth map is normalised with respect to the scale of $\mathbf{t}_1$.

Since the camera poses $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^n$ are normalised and now share a common scale, the proportions of the real 3D scene are preserved without distortion. Hence, the n-view plane sweep algorithm can be applied to construct the cost volumes V_i and regress the depth map D.

- **Plane sweep** Plane sweep is carried out by warping the target feature maps F_i through:

$$\tilde{\mathbf{x}} \sim \mathrm{K}\left[\mathrm{R}_i \mid \mathbf{t}_i\right] \begin{bmatrix} (\mathrm{K}^{-1}\mathbf{x})\, d \\ 1 \end{bmatrix}$$

$$\tilde{\mathrm{F}}_i(\mathbf{x}) = \mathrm{F}_i\left(\tilde{\mathbf{x}}\right)$$

  where $[\mathrm{R}_i|\mathbf{t}_i]_{i=1}^n$ are our normalised camera poses. The plane sweep algorithm samples $\mathrm{S} = 128$ depth plane hypotheses.

- **Cost volume construction** One cost volume V_i is obtained for each couple of

reference and target images. The features are aggregated following [17], yielding a 4D cost volume such that $\mathrm{V}_i \in \mathbb{R}^{H \times W \times 2Ch \times S}$, where Ch is the channel size of F_i. As [17], the pairwise cost volumes V_i obtained between each couple of reference and target image are averaged to produce a single cost volume V, which contains all depth information collected between the n views.

- **Depth map regression** The 4D cost volume V is regularized through a series of convolutional layers to output a 3D cost volume $\mathrm{V} \in \mathbb{R}^{H \times W \times S}$. The depth map is regressed using the probability distribution V$\mathbf{x}$) following:

$$\mathrm{D}(\mathbf{x}) = \sum_{s=1}^{\mathrm{S}} d_s \ \mathrm{V}(\mathbf{x}, d_s)$$

5.3. Implementation details

We train our network on the KITTI Depth training dataset. The training is carried out for only 6 epochs as it is very time consuming. [16] performs 10 epochs on the same dataset, which takes 40 hours on 8 NVIDIA V100 GPU cards. We instead perform the training on a single NVIDIA A100 GPU. We use PyTorch with mixed precision training to lower the computational burden. We crop the images to the size $\mathrm{H} = 256, \mathrm{W} = 768$ and set a batch size of 4. During training, we use $n = 3$ images. The initial learning rate is set to 0.0005 and dropped by half at 3 and 8 epochs. The convolutional networks that perform feature extraction, cost volume construction and depth map regression follow the same architecture as [6][16][17].

5.4. Discussion

We provide a comparison between our method and previous ones, discussing its strenghts and weaknesses.

1. **Trifocal tensor and Fundamental matrix** Compared to [16], our method has the advantage of handling the n-view case. As the authors make use of the Fundamental matrix F to estimate the camera pose, it simply is not possible to recover the relative scales between cameras from different couples. The only thing that [16] can do is normalise the scale. However, if they proceed to use the plane sweep with $n > 2$, the geometry of the scene would be distorted, as it would not respect the proportions between the ground truth camera poses.

2. **Point correspondences** To estimate the Trifocal tensor T, our method requires

correspondences over three views. As we use the optical flow to match points, we are left with two sets of corresponding points for every triplet. By intersecting this set however we decrease the number of available point correspondences. In some situations, where the available matches are already scarce, this could lead to worse pose estimates.

3. **Computational Cost** While [17] requires multiple iterations to recover and specifically to refine the initial pose, our method focuses on providing a precise pose without the need of further expensive computations. Since handling 4D cost volumes is expensive both in terms of memory and computation, we can reasonably expect the camera pose estimation step to be more light-weight with respect to [17].

4. **Well-posedness** By estimating normalised depths rather than real-scale depth maps, our method does not need to estimate the ground truth scale. This means that we are solving a well-posed problem, rather than an ill-posed one. Therefore, we can expect our network to be less biased by the dataset scale, as it is not forced to learn it.

6 | Experimental Evaluation

In this chapter we illustrate the experiments to validate the performance of our proposed pipeline. Section 6.1 provides some details on the general setting of our pipeline. Section 6.2 first illustrates the KITTI Depth and ETH3D datasets as well as the metrics used to measure the performance of our proposed SfM pipeline, then reports our main results and findings. Section 6.3 provides some additional experiments on the Herz-Jesu and Fountain datasets to compare the implemented Trifocal tensor estimation algorithms and motivate on our choice.

6.1. General setting

As the KITTI Depth dataset consists of frames from a video sequence, we have some freedom on the choice of target frames used to predict the reference frame. In particular, we consider two main configurations: (i) a *frame buffer* configuration and (ii) a *centered* configuration. To predict the depth map of reference image I_i, the n-view frame buffer strategy select frames $\mathrm{I}_{i-1}, ..., \mathrm{I}_{i-n+1}$. The centered strategy instead samples the frames around I_i, providing the frames $\mathrm{I}_{i-\frac{n}{2}}, ..., \mathrm{I}_{i+\frac{n}{2}-1}$. Notice that this step is independent from the triplet choice strategy, which acts on the selected target frames with the sliding window. In general, unless specified differently, we will use the centered configuration. As the distance between the reference frame and last target frame is of $\frac{n}{2}$ frames, rather than n as happens in the frame buffer configuration, we can expect this to help during the plane sweep algorithm.

Following the experiments detailed in section 6.3, our proposed pipeline estimates the Trifocal tensor through the Fundamental-T routine. This is the algorithm described in section 2.4.5. Given triplet $(\mathrm{C}_a, \mathrm{C}_b, \mathrm{C}_c)$, we estimate F_{ba} and F_{bc} using the RANSAC algorithm. Then, we use the 3-view correspondences $\{\mathbf{x}_j \leftrightarrow \mathbf{x}'_j \leftrightarrow \mathbf{x}''_j\}$ to recover the correct relative scale between $\mathbf{t}_{ba}$ and $\mathbf{t}_{bc}$.

Unless stated differently, the camera pose is always estimated through this algorithm. In some cases, which we denote as *Oracle pose*, the ground-truth camera pose is used instead.

This can be useful to understand the upper bound on the performance of our pipeline.

6.2. Proposed pipeline

We now illustrate our main results on the KITTI Depth and ETH3D datasets. Section 6.2.1 defines the main metrics, 6.2.2 discusses the KITTI Depth and ETH3D datasets, while sections 6.2.3 and onwards provide experimental evalutaion of the proposed pipeline.

6.2.1. Metrics

The main metrics used to evaluate a depth map D with respect to the ground truth depth map D_{gt} are the *Absolute relative difference*, the *Squared relative difference* and the *RMSE*. They can be computed as:

$$\text{Absolute rel. diff.} = \frac{1}{n}\sum_{i=1}^{n}\|\mathcal{D}_i - \mathrm{D}_{i,gt}\|_1/\mathrm{D}_{i,gt}$$

$$\text{Square rel. diff.} = \frac{1}{n}\sum_{i=1}^{n}\|\mathrm{D}_i - \mathrm{D}_{i,gt}\|_2^2/\mathrm{D}_{i,gt}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\|\mathrm{D}_i - \mathrm{D}_{i,gt}\|_2^2}$$

$$\text{RMSE}_{log} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\|log(\mathrm{D}_i) - log(\mathrm{D}_{i,gt})\|_2^2}$$

Additionally, the α^i metric can be used to evaluate the performance at different levels of precision. This is commonly defined as the percentage of elements in D which are sufficiently near to D_{gt}:

$$\alpha^i = \%\text{ of D s.t. } \max\left(\frac{\mathrm{D}}{\mathrm{D}_{gt}}, \frac{\mathrm{D}_{gt}}{\mathrm{D}}\right) < \delta^i$$

Three levels of precision widely adopted, each with its threshold, are:

- *High precision*: $\delta^1 = 1.25$
- *Medium precision*: $\delta^2 = 1.25^2$
- *Coarse precision*: $\delta^3 = 1.25^3$

6.2.2. Datasets

We now provide an overview of the datasets on which we run the experimental evaluation of our proposed pipeline. As we are mainly comparing our pipeline to [16] and [17], we use the same datasets. In particular, we run our experiments on KITTI Depth [2] and ETH3D [13]. The results from [16] and [17] are taken directly from their work and we do not re-run their pipelines.

KITTI Depth

The KITTI Depth dataset is mainly designed to evaluate the performance of monocular depth estimation algorithms in self-driving cars. A popular benchmark used as test set is the Eigen split, which consists of 697 individual frames. By extracting neighbouring frames, one can use this benchmark to test SfM algorithms with any number of views. For example, [16] uses paired nearby frames to compute pose and depth. As the dataset is collected on public roads, it contains lots of dynamic objects such as cars, buses and pedestrians. In some sequences, the camera is nearly static, for example when stopping at a red light. As these scenarios break the main assumptions of SfM, [16] proposed a subset of the Eigen split, namely the Eigen SfM split. This split contains 256 frames filtered from the original 697 such that no large dynamic objects are present and all frames contain at least a displacement in the camera of 0.5 meters. The raw images have a variable resolution but are generally resized to a fixed resolution of H = 256 W = 512 during training while H = 370 W = 1240 during evaluation.

As can be seen in figure 6.1 the difference between images from the KITTI Eigen split and KITTI Eigen SfM split is important. Indeed, figure 6.1b contains multiple dynamic objects that perturb the plane sweep, while figure 6.1a is a static scene.

(a) A frame of a static scene from KITTI Depth Eigen SfM split. All cars are parked and still.

(b) A frame of a dynamic scene from KITTI Depth Eigen split. Cars, buses and pedestrians are moving.

Figure 6.1: Comparsion between static and dynamic scenes in KITTI Depth dataset.

Figure 6.1a instead is static hence it verifies the main assumptions of SfM algorithms.

An important characteristic of the dataset is the baseline between consecutive frames. The baseline is the distance $\|\mathbf{t}_{ab}\|$ between the two cameras C_a and C_b that took the images I_a and I_b. Figure 6.2 shows that the baseline in the KITTI Depth dataset is generally quite small. Such conditions are optimal for neural network based optical flow estimators.

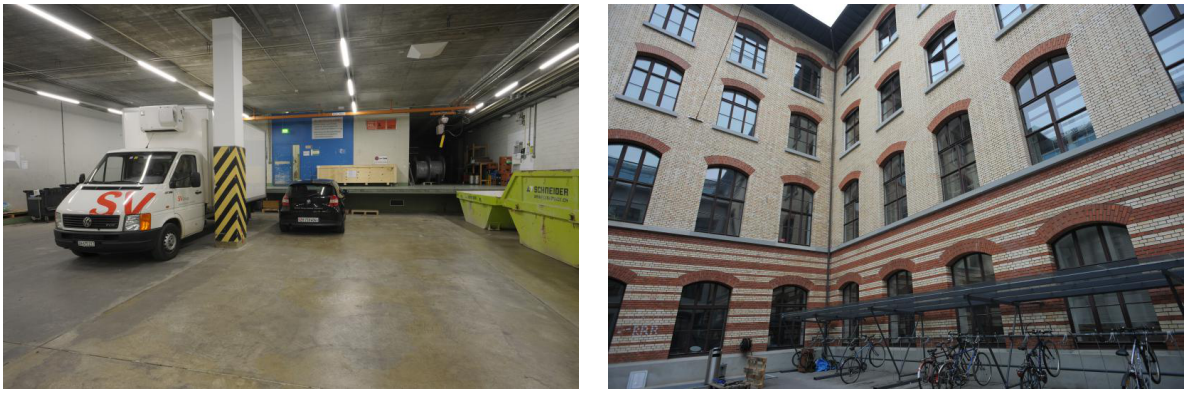
Figure 6.2: Two consecutive frames from KITTI Depth dataset

Overall, the KITTI Depth dataset has some recurring patterns in the scene: the depth tends to increase following the road, the horizon and deepest points are always in the top of the image, and nearby objects tend to be at the right and left borders of the image. This geometry can be different from other datasets and could potentially bias the neural network.

ETH3D

The ETH3D dataset contains both indoor and outdoor scenes collected with the aid of very accurate laser measurements. As such, the 3D groundtruth points are reliable and account for a solid benchmark. While the original dataset provides 3D point clouds of different scenes, it is possible to obtain depth maps from the 3D point clouds by projecting the 3D points on the provided camera planes. We use the splits made available from [5], which contain a total of 454 frames over 13 different scenes. The images are provided in high resolution but we follow previous works and downsize them to a fixed H = 540 W = 810 pixels.

Figure 6.3 shows two sample frames taken from the *Courtyard* and *Delivery Area* sequences of ETH3D. In this case, the geometry of the scene has greater variability with respect to KITTI Depth.



(a) *Delivery Area* (b) *Courtyard*

Figure 6.3: A comparison between different sequences from ETH3D.

While the scene is static, the baseline between cameras is bigger than what can be observed in KITTI. This is clear from figure 6.4 which shows two consecutive frames from the same sequence from ETH3D dataset. This can be a challenge to the deep optical flow computation as it works best in regimes with smaller baseline.

Figure 6.4: Two consecutive frames from the *Courtyard* sequence of ETH3D dataset

6.2.3. KITTI Depth

We report the results obtained on the two KITTI Depth splits, starting off with the KITTI Depth Eigen split and subsequently focusing on the KITTI Depth Eigen SfM split. For both splits, we set $d_{min} = 1.0$ and choose $S = 128$ depth samples. As the depth hypothesis samples are obtained through $d_l = \frac{Sd_{min}}{l} \; \forall \; l \in \{1, 2, ..., S\}$, the depth hypothesis range is $[1, 128]$. Note that the depth hypotheses are expressed with respect to the normalised camera pose.

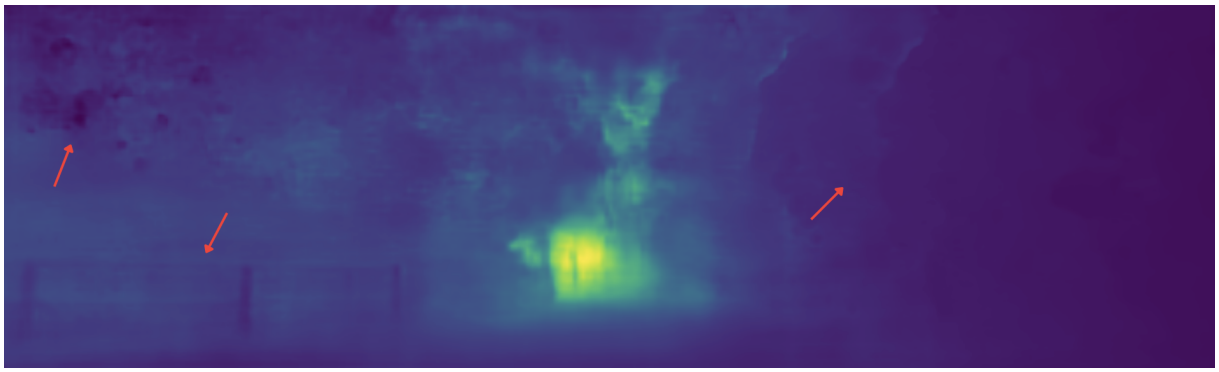
KITTI Depth - Eigen split

Pipeline	*lower is better*				*higher is better*		
	Abs Rel	Sq Rel	RMSE	$RMSE_{log}$	α^1	α^2	α^3
2-ViewRevisited [16]	**0.055**	**0.224**	2.273	**0.091**	**0.956**	**0.984**	**0.993**
Ours, $n = 3$	0.056	0.261	**2.264**	0.095	0.951	0.983	0.992
Ours, $n = 4$	0.090	0.419	2.670	0.128	0.899	0.968	0.987

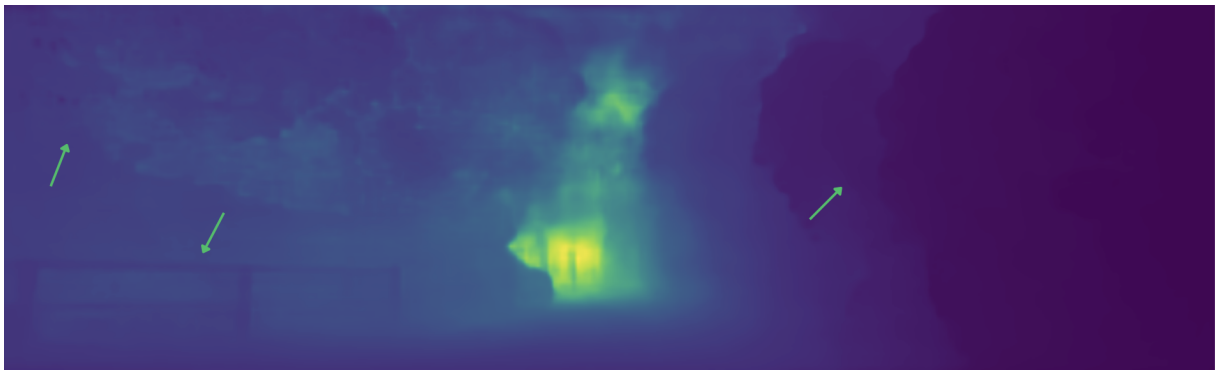
Table 6.1: Results on KITTI Depth Eigen split.

We run our pipeline with different configurations of n. The numerical results of table 6.1 show very similiar performance to previous work [16] when $n = 3$ and a worse performance when $n > 3$. To better understand the obtained results, we now report some samples of estimated depth maps. Through a visual inspection, it is possible to notice two main patterns: (i) depth map estimates in static regions of scene seem to improve when $n > 2$ and (ii) dynamic objects seem to increasingly perturb the depth map estimates when $n > 2$.

Figure 6.5 shows the estimated depth map of a static scene when using different values of n. Some interesting details can be observed. First, the depth maps obtained with $n = 3$ contain much less noise with respect to frames obtained with $n = 2$. The noise in the depth map can be observed from the irregular patterns that emerge in neighbouring pixels. While such pixels should have almost identical estimated depth, there tends to be too much variability in the estimated values. Similarly to noise, one can also observe the presence of depth artefacts. In this case, figure 6.5a displays inconsistent depth in the top left corner and the grey railing presents artefacts. Both the noise and artefact issues can be traced back to the 2-view setting and tend to disappear with increasing n.



(a) Depth map obtained with 2 frames

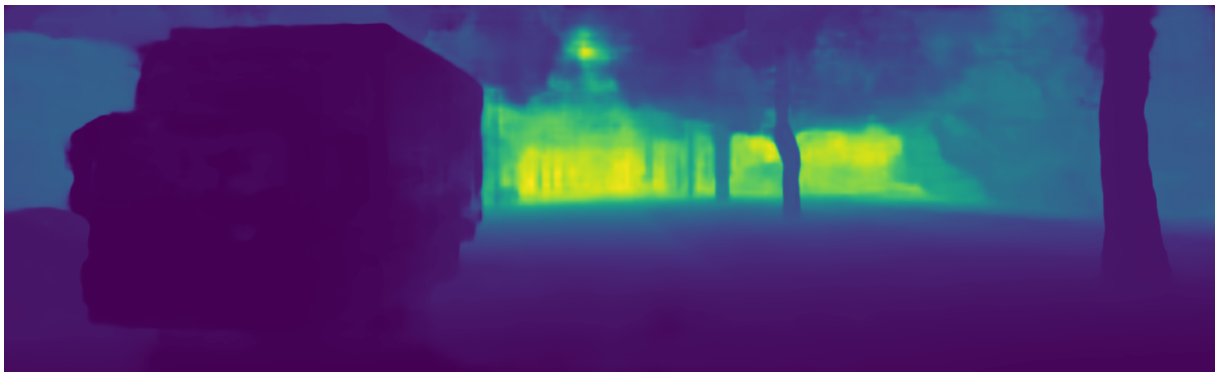


(b) Depth map obtained with 3 frames

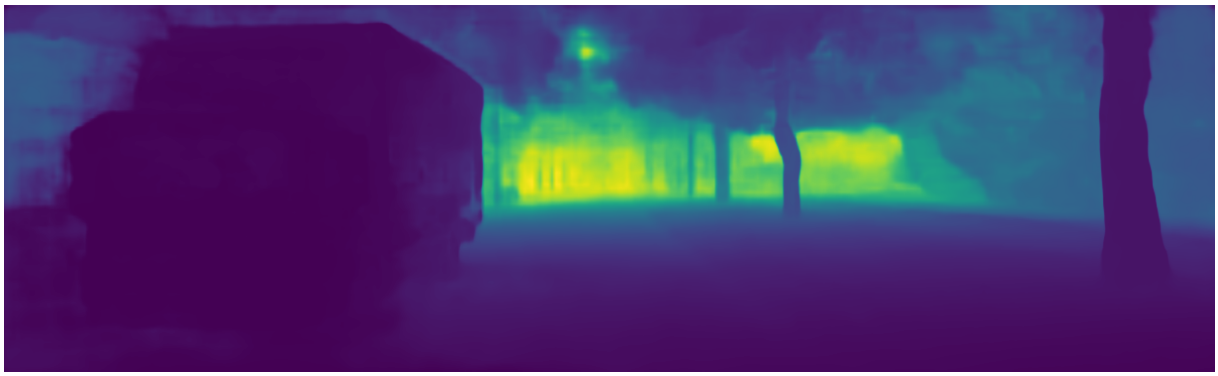
(c) Reference image

Figure 6.5: Depth maps obtained with $n = 2$, $n = 3$ and the original image. In the depth maps, dark color represents low depth, i.e. low distance between camera and object, brighter colors imply higher depth. The noise present when $n = 2$ is strongly reduced setting $n = 3$. The grey railing and bushes have better depth estimates and contours when $n = 3$.

While increasing n yields multiple benefits in static scenes, dynamic scenes display the opposite behaviour. A clear example of this can be observed in figure 6.6.



(a) Depth map obtained with 2 frames



(b) Depth map obtained with 3 frames

(c) Difference between depth maps

Figure 6.6: Depth maps obtained with $n = 2$, $n = 3$ and their difference. Dynamic objects estimated with $n = 3$ show significant artefacts. The reverse happens for static objects.

This happens mainly due to a bad cost volume construction. Indeed, when building the cost volumes through plane sweep, points are not matched correctly. As the warping accounts for the camera motion but not for the specific motion of each object, points belonging to a dynamic object are not transferred correctly from the reference image to the target images. In this case, the front of the truck is moving in opposite direction to that of the camera. The points near the boundary of the truck are incorrectly matched.

This effect worsens with the number of views and produces a characteristic blur which becomes especially clear on the boundaries of the dynamic object.

KITTI Depth - Eigen SfM split

We proceed to test our proposed pipeline on the KITTI Depth Eigen SfM split. As this split is designed to only select sequences that respect the main SfM assumptions, we can expect our pipeline to achieve better performance compared to the Eigen split. The results obtained in table 6.2 show a very good performance of our proposed pipeline, which is capable of improving by around 18% to 47% over [16] when setting $n = 3$. Given the promising performance, we test up to a total of $n =?$ views. Figure **??** shows the results obtained with respect to the number of views n.

Pipeline	*lower is better*				*higher is better*		
	Abs Rel	Sq Rel	RMSE	RMSE_{log}	α^1	α^2	α^3
2-ViewRevisited [16]	0.034	0.103	1.919	0.057	0.989	0.998	0.999
Ours, $n = 3$	**0.027**	0.071	1.617	**0.048**	**0.992**	0.998	0.999
Ours, $n = 4$	0.028	**0.070**	1.599	0.049	0.992	0.998	0.999
Ours, $n = 5$	0.031	0.073	**1.593**	0.052	0.991	0.998	0.999

Table 6.2: Results on the KITTI Depth Eigen SfM split. Increasing n yields benefits ranging from 18% to 47 %.

While the results show an average improvement of the metrics, applications of this technology also require the pipelines to be reliable. For example, a pipeline which achieves better results on average but completely blunders in certain situations can be dangerous in certain situations. We provide a brief analysis of the performance of our pipeline at different levels. Figure 6.7 displays the Absolute Relative error for each frame, sorted in ascending order by value. We can see that the performance of our proposed pipeline improves across the whole spectrum of errors.

Pipeline	Quantile (Abs Rel Error)								
	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
2-View Revisited [16]	0.023	0.024	0.026	0.028	0.030	0.032	0.037	0.045	0.072
Ours, $n = 3$	**0.019**	**0.021**	**0.022**	**0.023**	**0.025**	**0.026**	**0.029**	**0.033**	**0.050**

Table 6.3: Quantiles of the Absolute Relative Error. Our pipeline improves over [16] at all quantiles considered.

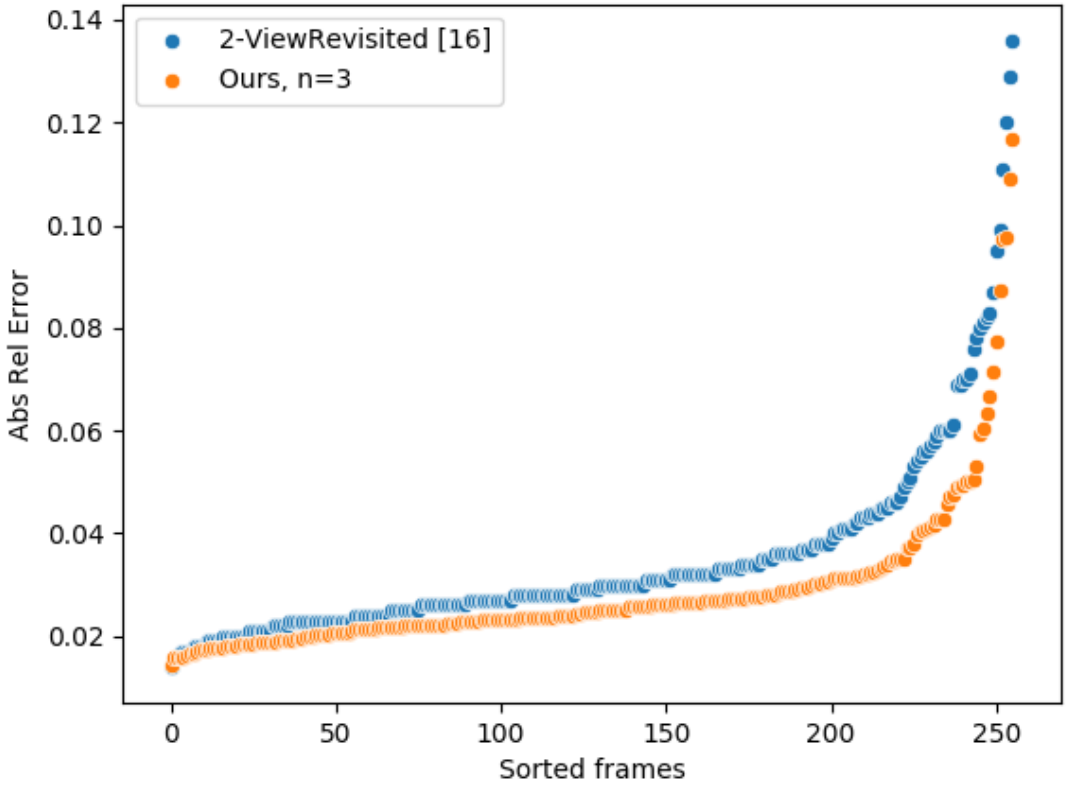
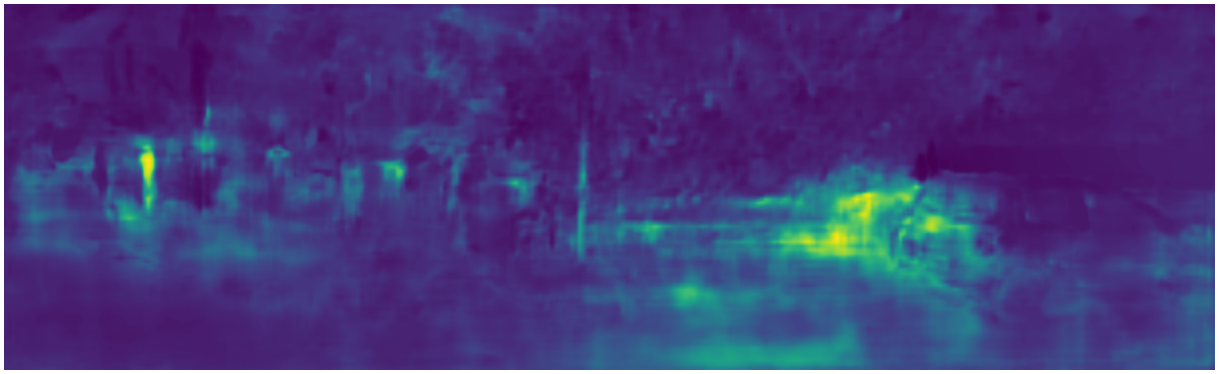


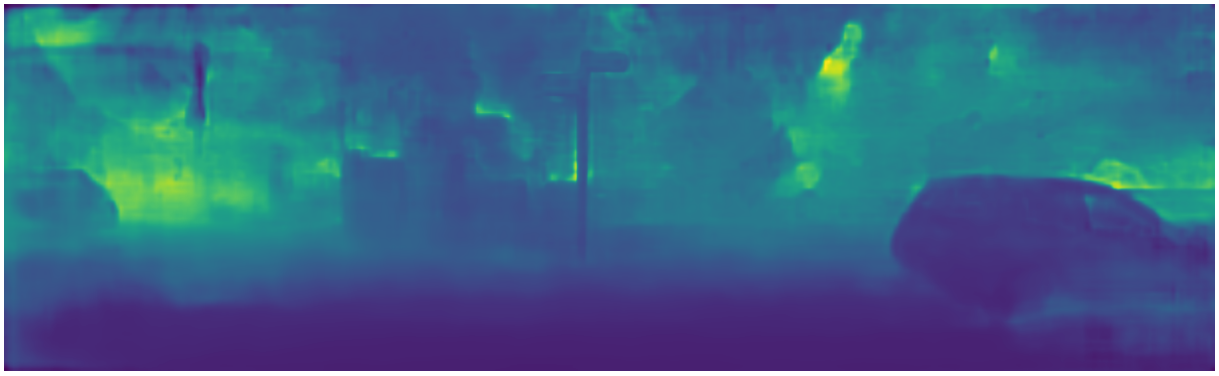
Figure 6.7: A comparison between the sorted Absolute Relative errors obtained on the KITTI Depth Eigen SfM split dataset. The 3-view pipeline consistently provides better and more reliable depth estimates compared to the 2-view pipeline [16].

Both the best and worse case scenarios improve when adding more views, providing a more reliable pipeline. A rather extreme case of improvement can be observed in figure 6.8. Here, the 2-view camera pose estimate is imprecise due to low displacement between the frames and negatively affects the depth map estimate. By choosing $n = 3$, the camera pose estimates are less affected by the reduced motion and the cost volume averaging

reduces the relevance of wrong estimates.



(a) Depth map obtained with 2 frames



(b) Depth map obtained with 3 frames

Figure 6.8: A comparison between depth maps for the same frame obtained with $n = 2$ and $n = 3$. The frame is an extreme case of bad camera pose estimates damaging depth estimation.

6.2.4. ETH3D

In this section we provide the numerical results obtained by testing our pipeline on the ETH3D dataset. While [17] and others train their pipeline on the DeMoN datasets, which are similar to ETH3D, we do not perform any other training and use the same pipeline trained on KITTI Depth. This represents a notable handicap for our pipeline when compared to the other works in the literature as both the optical flow and depth network are not fine tuned to handle indoor scenes and wider camera baselines. Indeed, the training of [16] on DeMoN datasets has a duration of 85 hours on eight parallel GPUs. On our hardware, this would roughly translate to more than three weeks of continuous training.

Camera pose	Pipeline	*lower is better*				*higher is better*		
		Abs Rel	Sq Rel	RMSE	$RMSE_{log}$	α^1	α^2	α^3
Oracle	2-View Revisited [16]	0.210	0.601	1.869	0.282	0.708	0.876	0.940
	Ours-3	**0.190**	**0.519**	**1.763**	**0.251**	**0.739**	**0.901**	**0.956**
	Ours-4	0.196	0.537	1.810	0.259	0.722	0.894	0.953
Estimated	2-View Revisited [16]	0.456	2.702	4.055	0.508	0.374	0.645	0.812
	Ours-3	**0.277**	**1.078**	**2.466**	**0.339**	**0.590**	**0.824**	**0.922**
	Ours-4	0.290	1.156	2.611	0.353	0.554	0.811	0.917
Estimated	COLMAP [12]	0.324	/	2.370	0.349	0.865	0.903	0.927
	DeMoN [14]	0.191	0.365	1.059	0.240	0.733	0.898	0.951
	DeepSfM [17]	**0.127**	**0.278**	**1.003**	**0.195**	**0.841**	**0.938**	**0.969**

Table 6.4: Results on ETH3D sequences. Increasing n yields benefits, especially when the camera pose is noisy. Our pipeline is not fine-tuned for this dataset and performs worse with respect to other fine-tuned pipelines.

From table 6.4 we can see that increasing n has an important effect on the performance of our pipeline. The effect seems particularly strong when the camera pose estimates are noisy. In such case, our pipeline provides an improvement which ranges from 49% to 150%. When using the ground-truth camera poses instead, the improvement of $n > 3$ views over the 2-view pipeline from [16] is reduced to a smaller 6% to 15%. Of course, as our pipeline has not been explicitly fine-tuned on the DeMoN datasets, it would be best to run these experiments with the fine-tuned pipeline and check that the obtained results still hold.

6.2.5. Detecting pose drift

In the majority of our experiments, our proposed pipeline definitely improves over [16] when $n = 3$, then provides minor improvement until around $n = 4$ and $n = 5$. Beyond this, the pipeline starts to struggle significantly and produces worse results. We believe that this can happen for two reasons, namely (i) there is a pose drift during the pose chaining or (ii) the cost volumes produced by target frames with big distance from the reference frame damage the pipeline.
The first issue could be caused by errors in the pose estimation accumulating through the chain, while the second could be due to the neural network being trained only in a small

range of camera poses. We show that (*ii*) is the most likely issue by running the same experiments with the ground truth pose. Figure 6.9 compares the trend of the Absolute relative error with respect to n on the KITTI Depth Eigen SfM split. As the error has the same trend with respect to n both using ground truth camera poses and estimated camera poses, we can conclude that pose drift is not the cause of this problem.

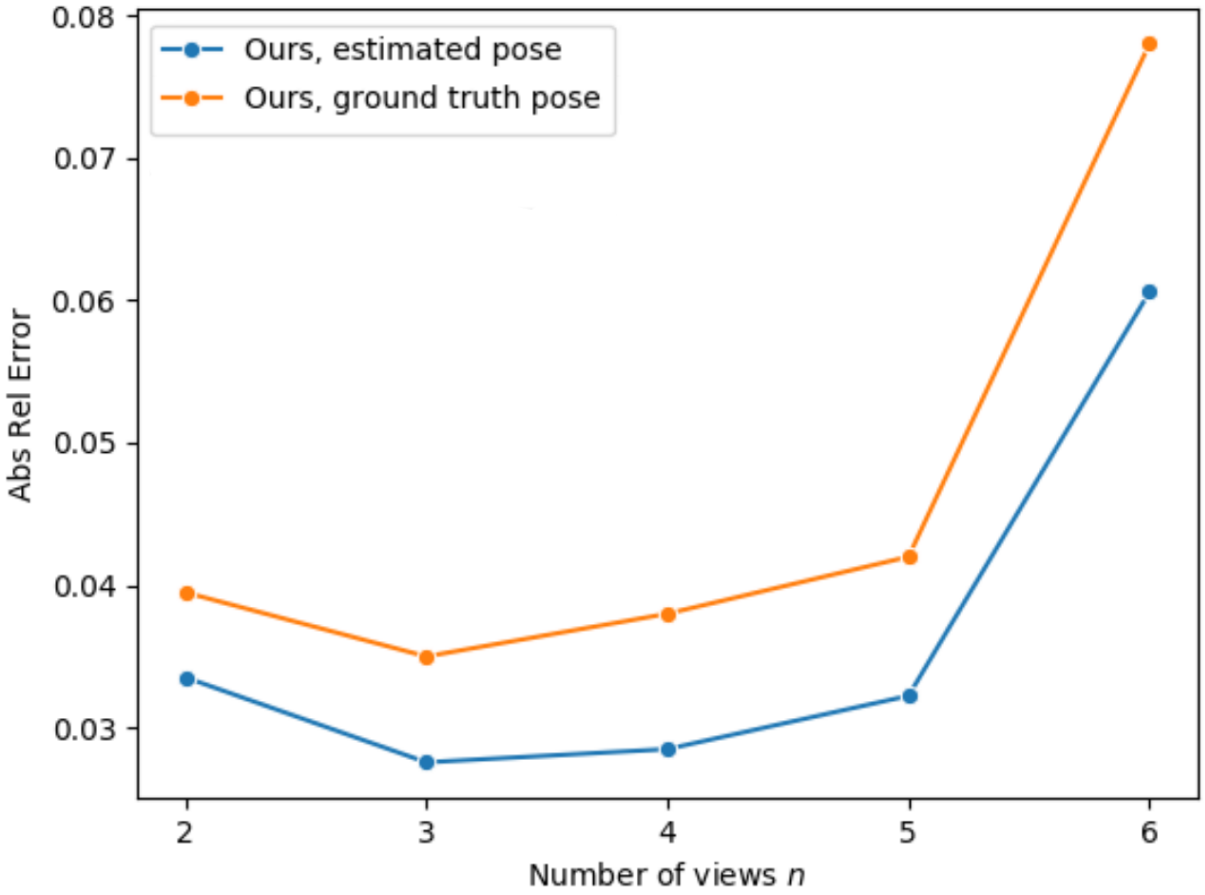


Figure 6.9: A comparison of the Absolute Relative error between the estimated pose and the ground truth pose, when using different n. Both show the same trend, meaning that pose drift is not the cause of worse performance when $n \geq 5$.

6.3. Trifocal tensor estimation

In this section we compare different algorithms to estimate the Trifocal tensor. We are mainly interested in the performance of the algorithms with respect to 3 main metrics: angular error of R, angular error of $\mathbf{t}$ and error on the norm of $\mathbf{t}$.
Let $[\mathrm{R}, \mathbf{t}]$ and $[\mathrm{R}_{gt}, \mathbf{t}_{gt}]$ respecitvely be the estimated and ground truth camera pose. We define the metrics as:

- Angular error on rotation: $\alpha = \arccos(\frac{tr(\mathrm{R}_{gt}^{-1}\mathrm{R})-1}{2})$
- Angular error on translation: $\beta = \arccos(\frac{\mathbf{t}\cdot\mathbf{t}_{gt}}{\|\mathbf{t}\|\|\mathbf{t}_{gt}\|})$
- Scale error: $\max(\frac{\|\mathbf{t}_{gt}\|}{\|\mathbf{t}\|}, \frac{\|\mathbf{t}\|}{\|\mathbf{t}_{gt}\|}) - 1$

The main implementations we test are:

- Linear-T: a linear estimate as described in section 2.4.3
- Gold-T: an estimate using the Gold Standard as described in section 2.4.4.
- RANSAC-F: estimate obtained from two Fundamental matrices as described in section 2.4.5. The Fundamental matrices are estimated through RANSAC.
- RANSAC-T: RANSAC algorithm performed with Linear-T and Gold-T as described in section 2.6. Linear-T provides the initialization of T.

We carry out the experiments on the two datasets provided by [7], namely *Herz-Jesu* and *Fountain*. As can be seen in the example frames of figure 6.10 and 6.11, the baseline between images is enough to prevent degenerate configurations which could lead to collinear cameras. While the scenes are static, potentially reducing the benefits of RANSAC, the point matches provided contain some noise which can perturb the Trifocal tensor estimates obtained without RANSAC.

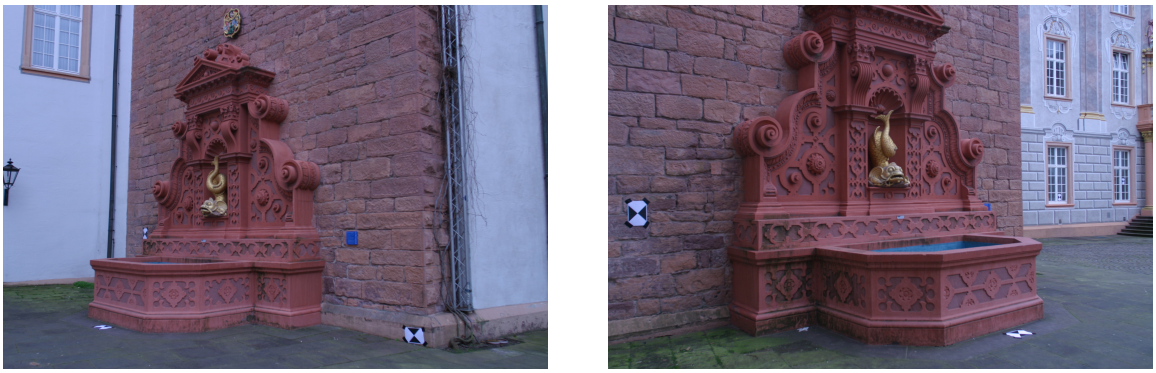


Figure 6.10: Two frames from *Fountain* dataset

Figure 6.11: Two frames from *Herz-Jesu* dataset

The results obtained in table 6.5 show that algorithms using RANSAC match or outperform the ones without it. While the number of inliers is very high, indicating that the matches are very accurate, the performance of Fundamental-T is very good and can be attributed to better performance of the estimation of the two fundamental matrices F. Indeed, we obtain the estimates of F using an implementation provided by [16], which has been optimized for the task with more complex algorithms and supports parallel CUDA computation to perform more computations in the same amount of time. For this reason, the algorithms employing this routine turned out to be significantly faster than the others.

Dataset	Algorithm	R error (°)	**t** rot error (°)	**t** scale error
Fountain	Linear-T	0.128	0.477	0.004
	Gold-T	0.122	0.498	0.004
	RANSAC-T	0.117	0.463	0.003
	RANSAC-F	0.081	0.190	0.003
Herz-Jesu	Linear-T	0.436	0.741	0.009
	Gold-T	0.381	0.618	0.008
	RANSAC-T	0.352	0.581	0.008
	RANSAC-F	0.246	0.429	0.008

Table 6.5: Pose estimation results by estimating the Trifocal tensor on Fountain and Herz-Jesu datasets.

7 | Conclusions and Future work

We now summarize important findings from this work and explore potential research directions.

7.1. Developed pipeline

We started the present work by introducing the general problem of Structure-from-Motion, focusing on the applications of supervised deep learning algorithms inside the SfM pipeline. We then reviewed the recent state-of-the-art work in the literature, detailing the current limitations. In particular, we highlighted that the majority of deep SfM pipelines tackling the n-view case of SfM make limited or inefficient use of well know 3D geometrical constraints and try to solve the ill-posed problem of recovering the real scale of the scene. We then introduced our pipeline with the aim of solving such issues. We achieved this by leveraging the Trifocal tensor to obtain camera pose estimates in triplets of cameras and by proposing a pose chaining algorithm which recovers the configuration of all the n cameras. Overall, the proposed pipeline strikes a balance between the use of neural networks during the optical flow estimation and cost volume construction together with the use of 3D geometrical constraints.

We then provided experimental evaluation of the pipeline on the KITTI Depth and ETH3D datasets. We showed that using higher values of n in presence of dynamic objects can lead to worse depth map estimates. We argue that this can be traced back to the cost volume construction, which only accounts for the movement between cameras while it disregards the motion of the single objects.
When the assumptions of the SfM problem hold and the scene is static, we showed significant improvements over 2-view state-of-the-art pipelines. In particular, the performance of our pipeline on the KITTI Eigen SfM split displays improved performance when setting $n \geq 3$. The improvements on the main metrics range from 18% to 47%. Together with quantitative measurements, we provide qualitative assessments of the output depth maps, showing reduced noise and depth artefacts. On top of this, we note that the proposed pipeline is more stable and less sensitive to adverse situations such as reduced camera

motion, improving on relevant metrics throughout all the sprectrum of quantiles.
Our tests on ETH3D provide promising results. While our pipeline was not fine-tuned for this dataset, we show that using more views has a strong positive effect, especially when the camera pose estimate is noisy. We also argue that the computational cost of our pipeline is reduced with respect to that of other state-of-the-art pipelines, a desirable feature especially considering potential applications of the technology.

7.2. Future work

In this section we discuss some research directions which we believe could provide further improvements to the algorithms.

As we have reported the negative effect of dynamic objects on the plane sweep algorithm, we believe that an improved version of this algorithm could be proposed. In particular, when projecting the pixels from the reference frame to each target frame, one could account for the motion of each object. Of course, this would require to segment the scene in two different categories, namely dynamic and static objects. For each dynamic object, assuming it is rigid, one should estimate its motion between reference and target frames. Overall, this should prevent dynamic objects from degrading the depth estimates, especially near their boundaries.

Another useful improvement could be to provide loop constraints on the camera poses. Indeed, by estimating the camera poses in such a way to obtain a cyclic graph, some optimization can be carried out to reduce pose drift for higher values of n. By adding further constraint on the camera poses, this could improve their consistency and reduce the depth map estimate errors.

Finally, it would be interesting to adopt more advanced strategies when aggregating cost volumes from different views. While we just take an average of the single cost volumes, advanced strategies could leverage criteria based on overlap, distance between the cameras and other features.

Bibliography

[1] 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*, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL https://dl.acm.org/doi/10.1145/358669.358692.

[2] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[3] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, 2 edition, 2004. ISBN 9780521540513. doi: 10.1017/CBO9780511811685. URL https://www.cambridge.org/core/books/multiple-view-geometry-in-computer-vision/0B6F289C78B2B23F596CAA76D3D43F7A.

[4] W. Hartmann, S. Galliani, M. Havlena, L. Van Gool, and K. Schindler. Learned Multi-Patch Similarity, Aug. 2017. URL http://arxiv.org/abs/1703.08836. arXiv:1703.08836 [cs].

[5] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang. DeepMVS: Learning Multi-view Stereopsis, Apr. 2018. URL http://arxiv.org/abs/1804.00650. arXiv:1804.00650 [cs].

[6] S. Im, H.-G. Jeon, S. Lin, and I. S. Kweon. DPSNet: End-to-end Deep Plane Sweep Stereo, May 2019. URL http://arxiv.org/abs/1905.00538. arXiv:1905.00538 [cs].

[7] L. F. Julià and P. Monasse. A Critical Review of the Trifocal Tensor Estimation. In M. Paul, C. Hitoshi, and Q. Huang, editors, *Image and Video Technology*, Lecture Notes in Computer Science, pages 337–349, Cham, 2018. Springer International Publishing. ISBN 9783319757865. doi: 10.1007/978-3-319-75786-5_28.

[8] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[9] T. Moons, L. Van Gool, and M. Vergauwen. 3d reconstruction from multiple images: Part 1 - principles. *Foundations and Trends in Computer Graphics and Vision*, 4: 287–404, 01 2009.

[10] L. Perelli. PyTFT. URL `https://github.com/LeoPerelli/PyTFT`.

[11] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. URL `http://arxiv.org/abs/1505.04597`. arXiv:1505.04597 [cs].

[12] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] T. Schöps, T. Sattler, and M. Pollefeys. BAD SLAM: Bundle adjusted direct RGB-D SLAM. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[14] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. DeMoN: Depth and Motion Network for Learning Monocular Stereo. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5622–5631, July 2017. doi: 10.1109/CVPR.2017.596. URL `http://arxiv.org/abs/1612.02401`. arXiv:1612.02401 [cs].

[15] J. Wang, Y. Zhong, Y. Dai, K. Zhang, P. Ji, and H. Li. Displacement-invariant matching cost learning for accurate optical flow estimation. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15220–15231. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/add5aebfcb33a2206b6497d53bc4f309-Paper.pdf`.

[16] J. Wang, Y. Zhong, Y. Dai, S. Birchfield, K. Zhang, N. Smolyanskiy, and H. Li. Deep Two-View Structure-from-Motion Revisited, Apr. 2021. URL `http://arxiv.org/abs/2104.00556`. arXiv:2104.00556 [cs].

[17] X. Wei, Y. Zhang, Z. Li, Y. Fu, and X. Xue. DeepSFM: Structure From Motion Via Deep Bundle Adjustment, Aug. 2020. URL `http://arxiv.org/abs/1912.09697`. arXiv:1912.09697 [cs].

[18] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan. MVSNet: Depth Inference for Unstructured Multi-view Stereo, July 2018. URL `http://arxiv.org/abs/1804.02505`. arXiv:1804.02505 [cs].

[19] Q. Zhu, C. Min, Z. Wei, Y. Chen, and G. Wang. Deep Learning for Multi-View Stereo

via Plane Sweep: A Survey, July 2021. URL http://arxiv.org/abs/2106.15328. arXiv:2106.15328 [cs].

List of Figures

List of Tables

ACKNOWLEDGEMENTS