A FRAMEWORK FOR ESTIMATING RELATIVE DEPTH IN VIDEO

by

Richard Rzeszutek
Master of Applied Science (M.A.Sc), Ryerson University, 2009
Bachelor of Engineering (B.Eng), Ryerson University, 2007

A dissertation
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Program of
Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2014

Author's Declaration for Electronic Submission of a Dissertation

I hereby declare that I am the sole author of this dissertation. This is a true copy of the dissertation, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this dissertation to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my dissertation may be made electronically available to the public.

Abstract

A Framework for Estimating Relative Depth in Video

Richard Rzeszutek

Doctor of Philosophy, Electrical and Computer Engineering

Ryerson University, Toronto, Ontario, Canada, 2014

This dissertation proposes a novel framework for recovering relative depth maps from a video. The framework is composed of two parts: a depth estimator and a sparse label interpolator. These parts are completely separate from one another and can operate independently. Prior methods have tended to heavily couple the interpolation stage with the depth estimation, which can assist with automation at the expense of flexibility. The loss of this flexibility can in fact be worse than any advantage gained by coupling the two stages together. This dissertation shows how by treating the two stages separately, it is very easy to change the quality of the results with little effort. It also leaves room for other adjustments.

The depth estimator is based upon well-established computer vision principles and only has the restriction that the camera must be moving in order to obtain depth estimates. By starting from first principles, this dissertation has developed a new approach for quickly estimating relative depth. That is, it is able to answer the question, "is this feature closer than another," with relatively little computational overhead. The estimator is designed using a pipeline-style approach so that it produces sparse depth estimates in an online fashion; i.e. a depth estimate is automatically available for each new frame presented to the estimator.

Finally, the interpolator applies an existing method based upon edge-aware filtering to generate the final depth maps. When temporal filters are used, the interpolation stage is able to very easily handle frames without any depth information, such as when the camera was stationary. However, unlike the prior work, this dissertation establishes the theoretical background for this type of interpolation and addresses some of the associated numerical problems. Strategies for dealing with these issues have also been provided.

Acknowledgements

Well, it's been about five years since I last wrote one of these; back then it was for my masters thesis. Professor Androutsos has remained my supervisor and because I've stayed at Ryerson, everything I said back then still applies now. Prof. Androutsos has been an excellent supervisor and since I've known him now for the better part of a decade (seven years in all), I'm probably better qualified to make that statement now then I was before. All in all, it's been a good experience.

Unlike masters, there were no industrial sponsors or internal university collaborators this time around. There was just a very general, "work on 2D-to-3D". But I did have the opportunity to work on several different projects and internships on top of my own research. Those helped expose me to new ideas and ways of approaching problems. Combined with being a teaching assistant, I did more than just narrowly focus on my own research topic. The people I worked with on those projects and the students that I instructed were all part of my experience and I would like to recognize that.

I've been at Ryerson, as a student, for a really long time and over the years I've seen new students arrive in the lab and older students graduate. I'm grateful to have had the opportunity to get to know them. Swapping "war stories" always helped to deal with the stresses of being a graduate student. I would also like to acknowledge the department's engineering support staff, with whom I had the pleasure of interacting with on many occasions. They helped keep the undergraduate labs running smoothly when I was a TA and provided much-needed technical advice on a number of my projects.

There are also those that I've met outside of my lab and the department. It's very easy to for the department to turn into its own, self-contained bubble and having some sort of extracurricular activity is really important in maintaining some sort of work/life balance. Well, as much as is possible when working on a PhD.

Finally, my family. They've always been supportive and they've continued to support me during my PhD studies. As time's gone on, it's gotten harder to explain to them what I'm doing but that's what I get for going into a highly technical field. That said, they're very happy that I'm finally done being a student.

Dedication

Dla moich dziadków, wieczny odpoczynek.

Contents

List of Tables

List of Figures

List of Appendices

Chapter 1

Introduction

Video provides a rich source of information about the depth in a scene. Unlike still images, where perspective cues can be either misleading or completely wrong, the motion of the camera itself provides a great deal of information and can help resolve many ambiguous situations. For instance, the simple phenomenon of parallax can very definitively indicate if an object is close or far from the camera. Intuitively an object moving very slowly is likely to be farther away than one moving more quickly. Granted, this makes a number of assumptions about the camera and scene but the point is still valid.

Collectively, image or video features that provide information about the underlying scene depth are known as, unsurprisingly, "depth cues" [1]. These are a number of different elements, such as vanishing lines, increasing texture frequency, fog, relative object size, etc., that all contribute to understand the spatial relationships between objects in a scene. With still images, there is no other information to go on. But with video, provided that *nothing in the scene moves*, the relative motion of features between frames provides a good indication of how far they are from the camera. In fact, the way in which features appear to move is well-defined. Since features are tracked across multiple frames it is possible to aggregate/integrate the information over multiple frames to help resolve ambiguities.

Having depth information is useful for a number of different reasons. Among other things, it makes it possible to know how far an object was from the camera and the relative ordering of objects, with respect to the camera. A good example is foreground/background object segmentation. Given some user input it is possible to obtain a reasonable segmentation [2, 3] simply because the strong edges near the user's labelling tend to be those of the foreground object. However, without any labelling the best that one can hope for, without any prior knowledge, is to ensure that visually distinct regions are grouped together [4, 5]. However when depth information *is* available it makes the task significantly easier, particularly in automated applications. The reason is quite simple: foreground objects are, by definition, closer to the camera. This information, coupled with edge information, can simplify the segmentation problem.

1.1 Representing Depth

Depth information, either for images or video, is represented through depth maps. These are monochromatic images that relate how close, or how far, a pixel is from the camera. Figure 1.1 shows an example disparity map for an image. A disparity map is not quite a depth map though, they are directly related; this will be elaborated upon in Chapter 3. However, as can be seen, brighter points are closer to the camera while darker points are farther away. The black "shadows" present in the disparity map are artifacts from how the map was obtained[1].

(a) Image (b) Disparity Map

Figure 1.1: An example of depth representation. Points closer to the camera are represented as brighter regions in Figure 1.1b while those farther away are darker. The "shadows" are artifacts from the capture system. Both the image and disparity map have been obtained from [7] and are publicly available at `vision.middlebury.edu/stereo`.

Disparity/depth maps are important because they provide information about *dense* depth in the scene for the particular camera position. They are not the same as a 3D reconstruction which is, more or less, a complete 3D model of the scene. For stereo applications they are extremely important because they relate the stereo images to one another.

It should be noted that a disparity/depth map is not the only method to represent depth. For instance, layered depth images [8] represent depth information as a series of layers with associated colour data and were intended to simplify the representation of complex scenes and rendering of novel viewpoints. That said, for the purposes of this dissertation, only depth maps will be considered.

Finally, the terms "depth" and "disparity" will be used more or less interchangeably. From a strictly semantic viewpoint, "depth" refers to the distance from the camera to a particular object

[1] The disparity map was obtained using structured light [6] and no disparity information is available for where shadows were cast.

while "disparity" is the apparent observed offset of that object when the camera's position is changed. The reason for this is that, generally speaking, disparity is inversely proportional to depth. Sometimes, however, the two will refer to different concepts but in these instances the usage will be clarified to avoid confusion.

1.2 Motivation

The primary motivation behind this dissertation is to address a number of issues with respect to depth estimation. In many cases, such as foreground/background segmentation, there is no need to have knowledge of the absolute depth. Instead the relative depth is sufficient. This is true in other applications as well, such as 2D-to-3D image conversion where rendering novel viewpoints is done under the assumption that the original depth information is unknown. Namely this dissertation will focus on a novel method for estimating relative depth in a much more computationally efficient and simpler way than with existing methods. This is intended to complement the existing work rather than replace it. As such, the method in this dissertation is presented in the context of a more general framework.

The need for this type of depth estimation was driven, primarily, by the author's prior work in 2D-to-3D image conversion [9, 10, 11]. Semi-automatic conversion is a relatively straightforward process as all it requires is that the user indicate the relative depths for different regions in a scene. However doing this for video is extremely time consuming because it requires labelling, if not every frame, a rather large number of them. While the depth maps are intended for conversion purposes, they can be used in other situations as well, e.g. applying depth-dependent effects such as depth-of-field. The fact that this dissertation is primarily presented in the context of 2D-to-3D conversion is not a limitation.

Currently, many estimation methods attempt to generate a full reconstruction of the observed scene. That is, the method will find the *actual* 3D structure of the scene, up to some scale and rotation factor (since these are impossible to determine without taking physical measurements of the scene itself). Obtaining a depth map from the reconstruction is very simple because it becomes a matter of determining where the camera had been located and measuring the distance from the camera to the scene element. However, as will be argued in this dissertation, this level of information is unnecessary in cases where only relative depth information is needed, such as with depth-based image rendering.

There are a number of different approaches for extracting depth information, all with their own particular benefits and drawbacks. There are two common, somewhat related, requirements that these methods must meet. The first is that the relationship between cameras must be known in advance. For instance, the vast majority of disparity estimation methods in stereo vision require that the cameras be parallel to one another since it simplifies the estimation procedure. This procedure is either done manually by physically ensuring the camera alignment or as a post-processing step

later on; this ties into the second requirement.

Alternatively, instead of knowing the camera relationship ahead of time a depth estimation method requires that the relationship between the camera views can be established later on; i.e. it is possible to find a transform that moves the camera from one position to another. For video, or large image collections for that matter, this means that there must be a large number of features that can be detected and tracked between frames. How this is done depends on the method and is usually straightforward tracking [12], some form of feature detection and matching [13, 14, 15] or a combination of the two [16, 17]. The purpose is to establish relationships between cameras and it often requires a minimum number of feature correspondences. Furthermore, feature detection is inherently noisy so the majority of these methods are statistical in nature and the more available features, the better likelihood of obtaining a good estimate.

The proposed method will be subject to the second requirement because of the nature of the geometry that is produced by multiple cameras or views. This is an inherent limitation in *any* sort of automatic depth estimation. Certain approaches such as one proposed by Torresani et al [18] can handle this but they are significantly different from more conventional methods and outside of the scope of this dissertation.

1.3 Dissertation Objectives

In keeping with the original research motivation, the primary goal of the dissertation has been to develop a depth estimation technique capable of being integrated into a semi-automated conversion framework. However, the depth maps can in fact be applied elsewhere, not just in 2D-to-3D conversion. The advantage of this approach is that the results of the proposed method can be adjusted/corrected by the user at some later time. This is only something that has been explored recently by other researchers.

There are a number of secondary objectives for the dissertation as well. The first of these is to develop a computationally efficient online approach to generating dense depth maps. Existing approaches[2] tend to be, almost exclusively, in the domain of offline processing. That is, they require *complete* access to all of the frames in a video sequence and perform multiple passes over the data. With the proliferation of mobile computing platforms with limited memory and computational power it makes it very difficult to use these methods with these devices[3]. It is beneficial to develop methods and techniques to work in these conditions since these algorithms will also be significantly faster on traditional desktop computers.

The other objective was to obtain relative depth in such a way that it requires minimal information about the camera used to image the scene. In existing approaches, the internal parameters

[2]More information on these approaches will be presented in Chapter 2.

[3]Some recent work, as of the time of writing, uses sensor fusion to speed up the process but it is still quite computationally difficult.

of the camera must either be determined ahead of time or estimated through a process known as "calibration" (Chapter 2.1.1). This is necessary when finding the true 3D reconstruction but unnecessary for simply finding the relative depth.

1.4 Contributions

This dissertation makes a number of contributions to the existing body of literature that are summarized here. These contributions represent the concrete solutions to the objectives outlined in the previous section (Chapter 1.3). They are as follows:

A novel depth estimation method.

Chapter 4 presents a novel method for estimating sparse, relative depth. The method is straightforward to apply to video and produces depth estimates very similar to more complicated approaches. The key to the method is that short term feature tracks provide enough information on their own to provide an estimate of depth at any particular frame. A full reconstruction is unnecessary for this.

Decoupling depth map generation from depth estimation.

In much of the prior work, estimating sparse depth and obtaining the dense depth maps was done *jointly*. That is, any initial sparse depth estimates were used as just one of several constraints in a large optimization problem. The depth maps cannot be generated independently from those initial estimates. Some recent 2D-to-3D work (as this dissertation was being written) have attempted to separate the two but there is still a tendency to treat estimation and generation as being intimately connected. Here these are treated as two, completely unrelated problems and it leads into the next contribution.

An extensible framework for depth map generation.

Because depth estimation and map generation are essentially two separate problems, the work in this dissertation is more properly a *framework* rather than any single algorithm or method. There is no requirement on how the sparse depths are estimated or on how the maps are created. This flexibility is demonstrated in Chapter 7 where the quality of the output can be selectively changed.

A framework for *online* depth map generation.

The sparse depth estimation in particular is designed *specifically* for online processing. The total memory and processing requirements, per frame, are relatively constant during processing. It should be stressed here that "online" simply means that only access to the current video frame is required; it does not necessarily mean that the method is "real-time" (though there is no reason why it cannot be). But because it is online, it can process video one frame at a time without requiring the entire video to be loaded into memory.

Providing a method for user-guided corrections.

While the proposed framework is completely automatic, its design makes it amenable to user-guided corrections. This is made possible through the use of edge-aware filters and *does not* require any modifications to the framework itself. Rather all it requires is just a second pass through the frame sequence after the user has provided their corrections. This does require that the original depth estimates are stored but as they are sparse estimates, they require substantially less storage than the video being processed.

Providing a theoretical background for filter-based sparse label propagation.

While the method used for sparse propagation is not new, the theoretical justification for this particular approach is lacking. While it has been shown that edge-aware filtering can simulate anisotropic diffusion, this argument fails to take into account the fact that images are discrete-space signals. As such, there are certain practical numerical issues that need to be considered when using this particular technique. This dissertation will discuss these problems in a theoretical context as well as providing a way to compensate for them.

1.5 A Note on Notation

To avoid ambiguity, there are a few things that should be mentioned about the notation used in this dissertation. First, vectors are treated as tuples so that

$$\vec{v} = (v_0, v_1, \ldots, v_{N-1}),$$

where N is the dimensionality of the vector. Conversely, $N \times M$ matrices are represented as

$$\mathbf{M} = \begin{bmatrix} m_{00} & \cdots & m_{0(M-1)} \\ \vdots & \ddots & \vdots \\ m_{(N-1)0} & \cdots & m_{(N-1)(M-1)} \end{bmatrix}.$$

In both cases zero-indexing is used rather than one-based indexing. This indexing style is used in other contexts as well unless otherwise specified.

It is useful to mix vectors and matrices together when performing computations. For this reason, in matrix form, a vector is defined as a single-column matrix

$$\vec{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{bmatrix}.$$

Therefore taking the transpose of a vector is

$$\vec{v}^T = \begin{bmatrix} v_0 & v_1 & \cdots & v_{N-1} \end{bmatrix}.$$

This is simply treating the N-length tuple as a $1 \times N$ matrix. Finally, some vectors are represented as $\tilde{v}$ rather than $\vec{v}$. The '$\sim$' is used to indicate that the vector has been augmented (it contains an extra dimension). They are still treated as column matrices but will have added properties that are discussed in detail later in the dissertation.

Chapter 2

Related Work

Most work into video depth map estimation has been done from a computer vision perspective. That is, the maps are typically recovered in order to obtain high-quality depth data that can then later be used to recover the full 3D structure of a scene. These methods are effective and well-known. Less work has been done from an image processing perspective. This may seem like a somewhat arbitrary division but the two related fields have different aims.

Image processing, a sub-field of signal processing, applies operators to a signal in order to transform it into some other representation or form. Several operators can be chained together in complex ways to produce an output that is best suited to the particular application. Computer vision is much more focused. Given an image of the world captured by a camera (or some other imaging device), computer vision methods seek to extract some information about what is being seen. Image processing methods are often used because that is the way *in which* computer vision is accomplished.

A classic computer vision problem is recovering a feature's 3D position and the orientation of the camera that took the image, i.e. its pose. This is known as Structure from Motion (SfM) and can be described as the solution to

$$\underset{\mathbf{P}_i}{\operatorname{argmin}} \sum_i D(\vec{x}_i, \mathbf{P}_i \vec{X}).$$

Given a 2D feature position $\vec{x}_i = (x, y)$ in image i, SfM attempts to find the true 3D position of the feature $\vec{X} = (X, Y, Z)$. This is done by minimizing the distance $D(\cdot)$ between a projection of $\vec{X}$ onto image i and the feature's position in the image, $\vec{x}_i$.

The utility of computer vision methods, such as SfM, is that once the structure and poses are recovered, the depth information for any particular image in the set of images can be found fairly easily. The biggest hurdle is always establishing those initial spatial relationships between the cameras and the feature positions. In the context of 2D-to-3D conversion, this has been used by a number of recent methods as a way of "bootstrapping" the initial depth maps.

The related work can be loosely categorized into the following groups:

Disparity Estimation

Direct measurement of the differences between two or more camera views, with the multiple (more than two) view case being referred to as Multiview Stereo (MVS).

Structure from Motion (SfM)

Recovering the structure (3D feature positions) and motion (camera positions and orientations) from known feature correspondences.

Edge-aware Filtering

A class of blurring filters that are sensitive to strong edges in an image.

2D-to-3D Conversion

Generating perceptually consistent depth maps/estimates either automatically or from user input for the purpose of generating novel views.

The categories above are not mutually exclusive, but rather serve to provide some structure for how the rest of this chapter is organized. For instance, SfM is often used with MVS because it can obtain the information necessary to correctly perform disparity estimation. The same is true with 2D-to-3D conversion. As will be discussed, some methods used SfM to provide constraints that are later coupled with user input to produce depth maps.

2.1 Disparity Estimation

Conceptually, disparity estimation is very similar to optical flow [19]. Optical flow seeks to find

$$\underset{\vec{u}(\vec{x})}{\operatorname{argmin}}\Big\{ I'(\vec{x}) - I(\vec{x} - \vec{u}(\vec{x})) \Big\},$$

where $\vec{u}(\vec{x})$ is a flow field indicating the mapping between $I(x, y)$ and $I'(x, y)$. If $\vec{x} = (x, y)$ and

$$\vec{u}(\vec{x}) = \big(u(x, y), v(x, y)\big)$$

then disparity estimation can be described as optical flow with $v(x, y) = 0$. This small assumption is crucial as it make an implicit assumption regarding the underlying camera geometry, namely that it only varies only the horizontal axis. Optical flow assumes that the motion is completely arbitrary while disparity estimation assumes that the observed motion is due to a particular camera configuration (discussed in Chapter 3). This makes disparity estimation the solution to

$$\underset{d(x,y)}{\operatorname{argmin}}\Big\{ I_R(x, y) - I_L(x - d(x, y), y) \Big\},$$

where $d(x, y)$ is the disparity map that relates the two views.

This has a number of implications in how disparity estimation algorithms work and is why they are mechanically quite different from optical flow algorithms. Scharstein and Szeliski established a basic taxonomy of disparity estimation techniques [20] to better categorize the different types of available techniques. They define the following categories:

- Local Methods (also known as "Winner Take All")
- Global or Semi-global Optimization
- Dynamic Programming
- Cooperative Algorithms

Essentially, all disparity estimation methods generate a "cost space" $C(x, y, d)$ where for each pixel (x, y) there is a cost to assigning disparity d for that pixel.

Local methods are mostly brute-force matching techniques that look for regions in the other image that best match some neighbourhood around a pixel in the current image. That is, they minimize $C(x, y, d)$ separately for each pixel. While these methods may sound simplistic, they can actually be quite sophisticated, such as using adaptive matching windows [21] or plane-fitting [22] on a per-pixel basis. The more sophisticated local methods begin to resemble the other class of stereo methods: global optimization.

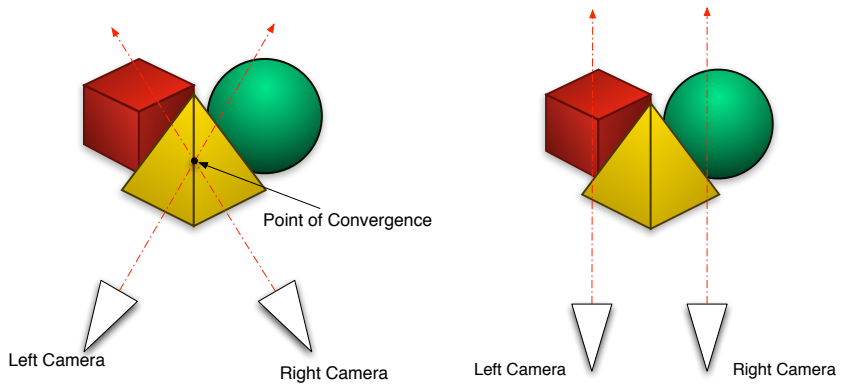
Global optimization approaches try to find a "globally optimal" value for $d(x, y)$. Solutions to this problem are usually found through methods based on belief propagation [23] or graph cuts [24]. Semi-global approaches attempt to do the same except on a more limited scale; semi-global matching by Hirschmuller [25] is an excellent example of such a technique. Dynamic programming and cooperative algorithms are both types of optimizations however Scharstein and Szeliski differentiate between the two because they operate differently from traditional global optimization techniques.

More recently, Lang et al [26] have proposed a dense disparity estimation technique based on sparse feature matching. This is significantly faster than prior approaches because the map is generated by interpolating the sparse estimates through a non-linear filter. A similar approach was taken by Hosni et al [21] where non-linear filters were used to estimate disparity maps in a "winner take all" framework. It should be noted that the work by Hosni et al is slower than that of Lang et al because it relies on dense matching. There will be more discussion on these filters in Section 2.3.

2.1.1 Image Rectification

An important component of disparity estimation is image rectification. Rectification is a pre-processing step that aligns two views such that they appear, after the transformation, they were taken with parallel cameras. This means that the camera lines of sight are parallel to one another.

This is illustrated in Figure 2.1. In this configuration the depth of a pixel becomes inversely proportional to its observed disparity.



(a) Verged Cameras (b) Parallel Cameras

Figure 2.1: An example of misaligned (verged) and aligned (parallel) cameras. Here "misaligned" simply means that the cameras' principle axes are not parallel.

Rectification schemes [27, 28, 29] generally produce a set of linear transforms $\mathcal{H}$ such that

$$\mathcal{H} = \{\mathbf{H}_i | i \in [0, N_V - 1]\},$$

where N_V is the number of views. In the stereo case, there are just two transforms produced but more may be generated when dealing with multiview stereo. Regardless, the result of applying the transforms is a virtual alignment of the cameras so that rows in one view are aligned with rows in another. The main drawback to this approach is the case where the camera has moved forwards or backwards. The transforms in $\mathcal{H}$ represent rotations in three-dimensional space and a forwards/backwards motion cannot be expressed like this. Non-linear warping schemes have been developed [30, 31] to accommodate for this type of motion. Unlike the linear schemes, applying them to images is significantly more complicated but it is much easier to find the transform.

If the intrinsic parameters of the cameras are known, i.e. they are "calibrated", then it is very simple to establish a relationship between point correspondences. This reduces the complexity of the problem because there is now a well-defined relationship between what is seen by the camera and the actual features in 3D space. If the cameras are uncalibrated then this adds an extra layer of complexity. For instance, Fusiello et al [28] derived a simple closed-form for the rectification transforms. But in [29], where the cameras are uncalibrated, Fusiello et al solve a large least-squares optimization problem because the camera parameters must be estimated along with their orientation. Non-linear rectification schemes warp the images based on the observed geometry without any knowledge of the underlying camera and therefore are not as sensitive to uncalibrated cameras. But as previously stated, being a non-linear warping, the methods are more difficult to apply to the images.

2.1.2 Multiview Stereo

The previously mentioned methods have all been for *stereoscopic* disparity estimation. That means each "image" has two views: a left and a right view. However, the idea of disparity can be extended to N-views (multiview). These methods are similar to stereo disparity except that they construct $C(x, y, d)$ across many images. They also differ in that they do not necessarily rectify the images beforehand (it can be problematic in certain cases) and instead search along epipolar lines directly [32].

A key difference with two-view stereo is that the end result of many MVS methods is often a 3D model of the scene [33]. That is because the images have been taken at multiple positions around the scene and so it may be more useful to obtain a single 3D model instead of N depth maps. However, this is not to say that *all* MVS methods generate 3D models. For instance, this is one way to obtain depth maps from misaligned cameras without having to perform rectification first [34]. It also allows the information from multiple cameras to be combined to produce a single depth map.

2.2 Structure from Motion

Structure from Motion (SfM) is a relatively broad term that describes the process of extracting three-dimensional coordinates (structure) from the observed differences caused by a change in a camera's position (motion). SfM is an extremely versatile technique and has been used in a number of different scenarios ranging from 3D reconstruction from unstructured photograph collections [32, 35, 36] to robotic navigation[1] [37, 38].

The method can be loosely summarized as follows:

1. Find two initial views that have a sufficiently large baseline (they were taken relatively far apart).
2. Find initial 3D coordinates through triangulation [39].
3. Update camera pose and feature position estimates based on other camera views through bundle adjustment [40, 41].
4. If the cameras were uncalibrated, use feature correspondences and camera pose estimates to autocalibrate [42, 43] the cameras.

The final step takes the reconstruction from a **projective** reconstruction to a **metric** reconstruction. Without this last step the reconstruction will appear distorted, e.g. features that should

[1] In robotics, SfM is often referred to as "Visual Simultaneous Localization and Mapping" (V-SLAM). The techniques are identical but the application is focused specifically on building a model of a robot's environment, as opposed to a more general theoretical problem.

be parallel to one another will not be. A metric reconstruction *is* the observed scene up to some similarity transformation (rotation and scaling). For a more complete description of the procedure, please refer to [44].

A variety of methods exist for performing SfM. For instance, Parallel Tracking and Mapping (PTAM) by Klein and Murray [45, 46] is able to reconstruct a scene in real-time by progressively re-running the bundle adjustment in parallel with the tracking. This is an online approach as it tracks features over many frames and once tracking has been performed long enough, it sends them to the bundle adjustment module to update the reconstruction. Newer approaches, such as the one by Tanskanen et al [47], incorporate information from a mobile phone's sensor to improve the reconstruction accuracy by providing hints on the camera's orientation and position.

These are examples of online processing because the features are updated incrementally as new images are made available. This approach is quite useful if the images are "structured", i.e. there is a well-defined relationship between them. For video it is reasonable to assume that there is little change between frames and so makes tracking straightforward. This is much more difficult to do with the images are "unstructured", such as a random collection of images from the internet with different resolutions.

A good example of offline processing is the photo tourism project[2] by Snavely et al [36, 48, 49]. Because the photos can come from anywhere, sparse feature matching has to be used instead of tracking. The reconstruction itself can be updated incrementally but the initial feature matching must be done on every image in the database. Adding new images may or may not update the reconstruction because it will depend on whether or not existing features are matched to the new ones.

Regardless of how the feature detection is done, the underlying process is still the same: obtain features, initialize reconstruction, add new cameras and refine the estimation. In order to produce a proper metric reconstruction, as it is often the desired result, these methods require the intrinsic properties of the camera. Namely, they need the focal lengths and the amount barrel distortion. Knowing the focal length makes it possible to convert a projective reconstruction into a metric one. Knowing the barrel distortion ensures that straight features map to straight lines in the images. If they do not, there will be error in the resulting reconstructions.

There are several ways to obtain the camera intrinsics. First, they can be calibrated beforehand; tools are available to do this. Second, they can be autocalibration can recover them from a projective reconstruction [39] (Ch. 19). Finally, image metadata, such as EXIF tags, can provide information such as focal length. Please note that these are not mutually exclusive. For example, photo tourism uses the EXIF data to provide a set of initial values and further refines them. The Autodesk 123D® Catch service[3] can generate a 3D model from multiple images with enough fidelity to be used with a 3D printer.

[2] Better known by its commercial name, "Photosynth" (`http://photosynth.net`).

[3] `http://www.123dapp.com/catch` (Retrieved July 29, 2013)

While SfM can produce very impressive results, it comes at a high computational cost. Photo tourism required a tremendous amount of computing resources to produce its reconstructions [50]. The feature matching alone turned into what could be termed as a "big data" problem. Similarly, even though Klein and Murray were able to port PTAM onto a second-generation iPhone (iPhone 3G), they had to make a number of concessions in the process [46]. While Tanskanen et al was aided by the sensors on a high-end mobile device, it was still not enough to run in real time (between two and three seconds to generate a reconstruction).

2.2.1 Feature Detection and Tracking

As alluded to in the previous section, processing video with SfM is a slightly simpler problem than with unstructured image collections. This is because with video, it can be assumed that features in one frame are most likely going to be visible in the next frame. Therefore, feature correspondences do not have to be re-established each time. Instead, they can be updated (tracked) from the current frame to the next one.

Methods designed specifically for video, such as [51], work in much the same way as PTAM. Given the current set of features, when a new frame arrives they look for where the features should be in the new frame. For image collections, feature detection methods, such as SIFT [13] and SURF [14], are the only truly effective way to match features in one image to another. With video, however, a simple KLT tracker [12] can often be more than sufficient.

In practice KLT loses features very quickly as they move out of the frame, are occluded or experience tracking errors for one reason or another. For this reason video-based SfM methods generally spend a fair bit of time ensuring high quality tracking because it greatly improves the final SfM results. In an SfM context, tracking methods attempt to find and track the features that are most representative of the scene's structure. This can be done in two ways.

First, the tracking can be done in tandem with the SfM estimation; this is what PTAM does. A similar approach was done by Cordes et al [17], where pairwise matching was performed using SIFT. The descriptors were stored so that if a feature became occluded, it could be queried later on to determine if it had reappeared. To ensure good tracking quality, features that did not respect the current estimate of the scene structure were treated as outliers and rejected.

Alternatively, the long-range and short-range tracking approaches can be hybridized, as done Zhang et al [16]. Feature matching is used across the entire sequence but over short time periods (10 to 20 frames), KLT tracking is used instead. To ensure that the features being matched were "reasonable", they perform planar motion segmentation on the tracks and attempt to transform features in one frame to the other. This ensures that features on the same surface move together. Because, like Cordes et al, they have SIFT features for each track, they can join features from multiple videos or cameras to extend their tracks even further.

2.2.2 Factorization

An important method related to SfM is projective factorization [52, 53]. This is a "simplified" version of SfM and can be somewhat viewed as a precursor to modern bundle adjustment-based approaches. It relies on the observation that

$$\lambda_i \vec{x}_i = \mathbf{P}_i \vec{X} \; \forall \; \vec{i},$$

or that for each projection of $\vec{X}$ onto image i, $\vec{x}_i$, it is really a scaled version of $\vec{X}$. Unlike SfM, projective factorization offers a *closed form* estimate of the scene's structure. This makes it rather simple to implement when given a set of feature tracks, such as those available from video. However, it requires that the tracks are dense (all features must be visible) and is less extensible than bundle adjustment, in the sense that new features cannot be easily added to refine an existing SfM estimate.

Factorization is mentioned simply because of how it deals with tracks is relevant to this dissertation. In particular, Triggs introduced the notion of "parallel" and "serial" chains in [53]. These are important because these methods are used as part of the depth estimation procedure in this dissertation.

2.3 Edge-preserving Filters

Edge-preserving filters are important to disparity estimation because they are often used to filter the "cost volume", an N-dimensional space that relates the cost of assigning a particular disparity value for each pixel. While there are a number of different filter types, they all share the following property: the filter kernel is dependent on the image content. That is, the filter changes itself so that strong edges are preserved while spurious features are smoothed out. An advantage to this approach is that these filters can use *another* image to control the filter kernel instead of the image being filtering; this is often known as "guided" or "joint" filtering.

The best known edge-preserving filter is the bilateral filter [54]. It is defined as

$$I_f(\vec{x}) = \frac{1}{W(\vec{x})} \sum_{\vec{p} \in \Omega} G_{\sigma_s}(\|\vec{x} - \vec{p}\|) G_{\sigma_r}(|I(\vec{x}) - I(\vec{p})|) I(\vec{x}),$$

where $W(\vec{x})$ is a normalization constant so the weights of the filter sum up to '1'. The terms $G_{\sigma_s}(\cdot)$ and $G_{\sigma_r}(\cdot)$ are known as the spatial and range terms, respectively. They are both zero-mean Gaussian functions with different values of σ that control how wide the Gaussian is in the spatial and range domains. The spatial parameter σ_s ensures that the filter response decreases the farther away from the current pixel $\vec{x}$. The range parameter σ_r controls the strength of the filter with respect to the difference in pixel values. If $|I(\vec{x}) - I(\vec{p})|$ is very large (indicating a strong edge), then the corresponding filter weight will be small to avoid affecting it. Conversely, if the difference

is small, it implies the region is homogeneous and safe to filter. Because the range term does not have to be derived from the image being filtered, the edge information in one image can be used to control the filtering in another.

While the bilateral filter is probably the best known of the edge-preserving filters, it is not the only one. One particular problem with the bilateral filter is that a naive implementation is quite slow. As such, work has gone into making it faster [55]. Unfortunately this requires making an approximation to the filter. A variety [56, 57, 58] of edge-preserving filters have been recently proposed in order to avoid this problem, two of which will be discussed here.

He et al [58] proposed their "Guided Image Filters" that can be efficiently implemented using nothing more than integral images [59] and element-wise arithmetic. The filters assumed that images are locally linear (little variation in colour around a pixel) and so it attempts to enforce that constraint. Alternatively, Gastal and Oliveira [57] proposed filters based on a so-called "domain transform" that would make similar colours spatially closer in a transformed domain (hence the name). Unlike the guided image filter, the Domain Transform (DT) filter is only defined on one dimension and is explicitly designed as a separable filter. Because the filter is non-linear, it requires multiple vertical-horizontal passes. The advantage, though, is that it requires very little memory to run and is very fast.

2.4 2D-to-3D Conversion

2D-to-3D conversion attempts to create stereoscopic (left-right image pairs) or multi-view images from monoscopic (single-view) images or video. There are a variety of different ways to perform conversion but the basic process is this:

1. Obtain a depth map for particular image.
2. Render a new image from the depth map to simulate the view from a slightly different camera position.

The second step is known as Depth-based Image Rendering (DBIR) [60]. DBIR is the last part of conversion where the single-view image or video is converted into its stereoscopic or multi-view form. It is essentially a warping with some inpainting to fill in regions that were exposed after the warping was performed.

More interesting, and relevant to this dissertation, is the first stage where the depth map is produced or generated. The depth map is key to conversion because it controls how the novel views are generated. What is particularly surprising about conversion is that the depth maps *do not* have to be accurate, merely plausible [61]. This has created a large group of methods where the depth maps produce pleasing stereo pairs but have no relationship to reality. Alternatively there are a class of methods that use computer vision-based approaches to obtain the depth map.

Conversion can be grouped into two relevant categories: semi-automated and automated. Semi-automated methods are those that rely on some degree of user input and generate maps from that. Automated methods are those that attempt to obtain the maps without *any* user input. Recently a group of methods half-way between semi-automated and fully-automated have been proposed but for the purposes of this discussion they will be considered to be semi-automated.

2.4.1 Automated Methods

Automated methods generate depth maps without any guidance at all from a user. The complexity depends on the quality of the results and the current state of the art is lacking in many ways. Part of the problem is that automated methods cannot be (easily) corrected if any errors are spotted. The only way to affect the outcome is to adjust the runtime settings and hope for the best.

Some of the more image processing-based methods have used information in the image as a proxy for depth. For instance, Tam et al [62] used the Cr chroma channel in a YCbCr image as a depth map. Similarly, Angot et al [63] performed joint bilateral filtering with a very simple depth map and an image so that the image edges would be visible in the depth map[4]. In both cases, the goal was not to create a real depth map, simply one that would look plausible.

Another approach has been to use machine learning methods to use depth cues (shading, vanishing lines, etc) in an image. Make3D by Saxena et al [64] trained a classifier using data from a laser range finder to determine how visual features were correlated to depth. Also, it is possible to learn depth from existing stereo sequences [65]. More recently, Karsch et al [66] and Konrad et al [67] have used nearest-neighbour matching to find images with known depths and merge them together to obtain a depth map for an image with unknown depth. It should be pointed out that visual cues can also be used directly without any machine learning, such as what was done by Tian et al [68].

Computer vision methods can be applied directly to this problem too. Knorr et al [69] used SfM to obtain camera positions from a video and rendered novel views by warping existing frames. Something similar was done around the same time by Zhang et al [70]. However, neither method actually generated a depth map and so if the existing footage is not conducive to generating new frames then even if the reconstruction was good, the result will not. An example was provided in [70].

However, other methods *have* generated depth maps using SfM/MVS. The most well-known of these is the work by Zhang et al [71] on consistent depth recovery from video. They first perform a sparse reconstruction using SfM and use MVS to estimate the dense disparity at each frame. A global optimization is used to enforce temporal and photometric consistency for the entire video. They used this method for depth-based effects rendering [72] with a post-processing step to accommodate for errors in the depth maps caused by moving objects.

[4]Depth edges *must* match object edges for the depth to appear plausible.

2.4.2 Semi-automated Methods

Semi-automated methods accept user input and generate a depth map from that. These methods are conceptually similar to user-guided image segmentation and for a good reason: depth map generation can be viewed as a labelling problem. In fact, this is how disparity estimation is formulated. Each pixel has a particular cost of being labelled with a particular disparity value and disparity estimation seeks to find the best label for each pixel. The difference is that disparity estimation is wholly automatic and does not need any user input.

There have been several semi-automated methods proposed recently, of which Guttmann et al [61] was the first. Their approach was to have a user mark up select frames in a video and then interpolate them through a combination of solving a weighted least squares problem and a support vector machine classifier. Since then a number of different approaches have been developed. The StereoBrush method by Wang et al [73] interpolates user-provided strokes by solving a sparse linear system. Phan et al [74] combined Graph Cuts [2] and Random Walks [3] to perform the interpolation and also allowed for temporal propagation similar to Guttman et al. These methods can all be traced back to the colourization work of Levin et al [75] where sparse user inputs were used to add colour to black-and-white images and video.

The main problem with semi-automated methods is the label process itself, specifically when working with video. Generally speaking the labelling itself needs to be available for each frame or a "ghosting" effect can occur [9]. Doing this for video can be quite time-consuming, even if it is easier than doing the conversion completely manually.

As such there have been several methods proposed recently to help with this. The first such method is Depth Director by Ward et al [76]. Depth Director is built around Graph Cuts image segmentation. The user is able to extract different scene elements in order to change their depth. To ease the burden on the user, the footage can be first passed through an SfM stage to generate some preliminary depth labels that can be adjusted later.

Video Stereolization by Liao et al [77] does much the same thing as Depth Director except that it attempts to extract depth information from the optical flow as well as what SfM produces. Constraints can then be added by the user to provide extra cues on what the depth should be. Movie Dimensionalization by Becker et al [78] also uses SfM as an input prior but can revert to user labels if need be.

Of what has been discussed in this Chapter, this dissertation is conceptually similar to Depth Director, Video Stereolization and Movie Dimesionalization. However, this is not quite correct. The method is built to be *autonomous-first* with an option for user correction later on. This makes it more flexible in how it operates and allows it to handle footage that SfM methods cannot. It can also produce high-quality depth maps without any costly global optimizations. However, unlike those methods, it is not a full conversion pipeline as it omits the DBIR step and does not focus on user interaction.

Chapter 3

An Introduction to Projective Geometry

Prior to discussing how depth information is encoded inside of a video, it is important to formally define what occurs when a camera images a scene. The subject is known as "projective geometry" as it describes the effects of projecting a higher dimensional space onto a lower dimensional space. Specifically, as releated to this dissertation, the goal is to examine what happens when a 3D scene is imaged on a 2D plane. Hartley and Zisserman [39] provide an excellent, in depth overview of the math behind projective geometry. A similarly good overview is provided by Szeliski [79]. This section is mainly intended to gently introduce the reader to the topic and not replace those resources.

3.1 Pinhole Cameras

The vast majority of footage being captured is done with cameras that are well-represented by the pinhole camera model [79](Ch. 2, pp. 49–50). A nice feature of the model is that the front end of the human visual system, i.e. our eyes, are essentially pinhole cameras. The model itself is quite simple and is is illustrated in Figure 3.1.

The pinhole camera model is very simple: an imaging surface rests behind an opaque barrier with only a single, infinitely small opening (the pinhole). The purpose of the pinhole is to avoid all possible light rays from intersecting the imaging surface. Instead, the pinhole only allows a single light ray to enter into the camera and hit the imaging surface. This mirrors the image in both the horizontal and vertical axes but it is trivial to correct. For completeness, it should be noted that in practice the pinhole has some measurable diameter known as the "aperture". How exactly the aperture effects the resulting image is beyond the scope of this dissertation but it can be loosely summarized as, "the smaller the aperture, the sharper the image." For readers familiar with photography, this is corresponds to an aperture of f/∞.

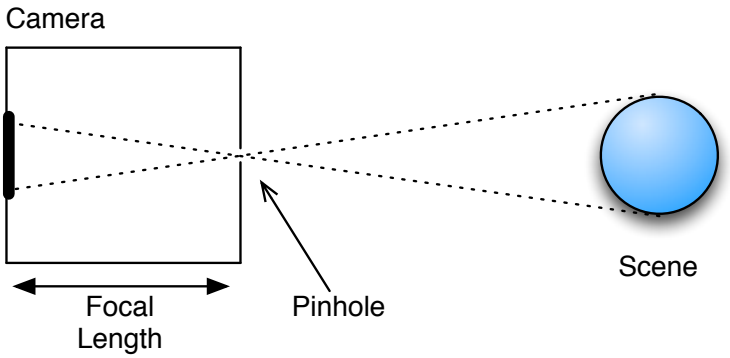


Figure 3.1: A basic illustration of a pinhole camera. In this model, the camera is an enclosed volume with a single, infinitely small "pinhole" to allow in light from the scene. The scene itself is projected onto the rear surface of the camera. The camera's focal length is defined as the distance from the imaging surface to the pinhole.

An observation of the model is that the rays from the object to the imaging surface form similar triangles. That is, the triangle from the object to the pinhole is a scaled version of the triangle from the pinhole to the imaging surface. Furthermore, it is convenient to consider that the object is *behind* the imaging surface rather than in front if it. This effectively makes the pinhole the origin of a coordinate system and the imaging surface a plane some fixed distance from the origin. This is shown in Figure 3.2. As seen in the figure, the camera points down the positive z-axis[1]. The convention that the image y-axis (y' in the figure) points also poses no problems; it just requires mirroring the image along the y-axis.

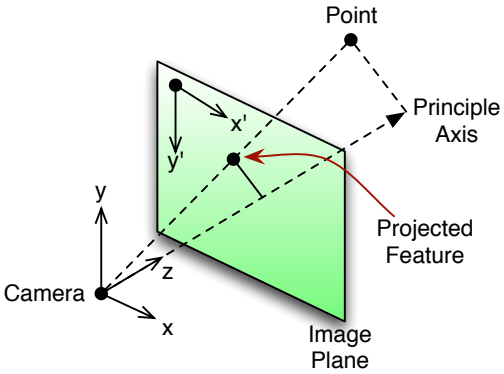


Figure 3.2: The camera coordinate system. The direction towards the imaging surface is considered to be in the $+z$ direction. This is also considered to be the camera's principle axis.

The advantage to this approach is that determining where a point $\vec{p} = (x, y, z)$ in the scene will be projected onto the image, $\vec{q}$, becomes very straightforward. Figure 3.3 shows the geometry

[1]The choice of sign is completely arbitrary; it simply determines the handedness of the coordinate system. For instance, OpenGL uses the convention that $-z$ is in front of the camera (a right-handed coordinate system).

of projection for a single dimension. Because this configuration produces similar triangles, the mapping of $\vec{p} \to \vec{q}$ is

$$\vec{q} = \frac{f}{z}(x + c_x, y + c_y), \tag{3.1}$$

where c_x and c_y is the camera's optical centre in *image coordinates*. In other words, $\vec{q}$ is a scaled and shifted version of $\vec{p}$ with the last coordinate being dropped. The shift is necessary because the convention for images is that the image origin is the top-left corner.

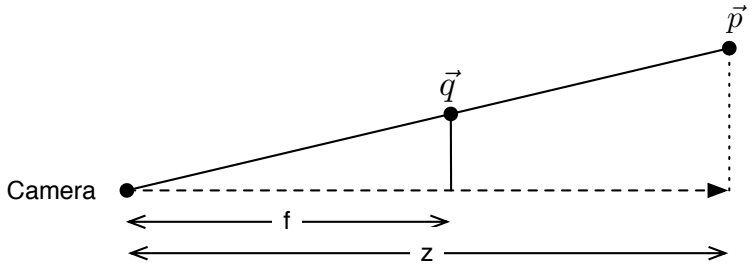


Figure 3.3: Similar triangles produced by the pinhole camera model.

Two things should be noted about (3.1). First, f and $\vec{c} = (c_x, c_y)$ are independent of the scene; they are properties of the camera itself. As such, these parameters are known as the **intrinsic parameters**[2] of the camera. Second, the total amount of scaling is inversely proportional to the distance of the point from the camera. The result is that points far from the camera appear close together and points close to the camera have an almost one-to-one mapping. This is what causes the effect of parallel lines appearing to meet at infinity (Figure 3.4). The focal length, in particular, is an important parameter because it affects *how much* perspective distortion is observed. The relationship is very simple: the greater the focal length, the less distortion observed.

3.2 Homogeneous Coordinates

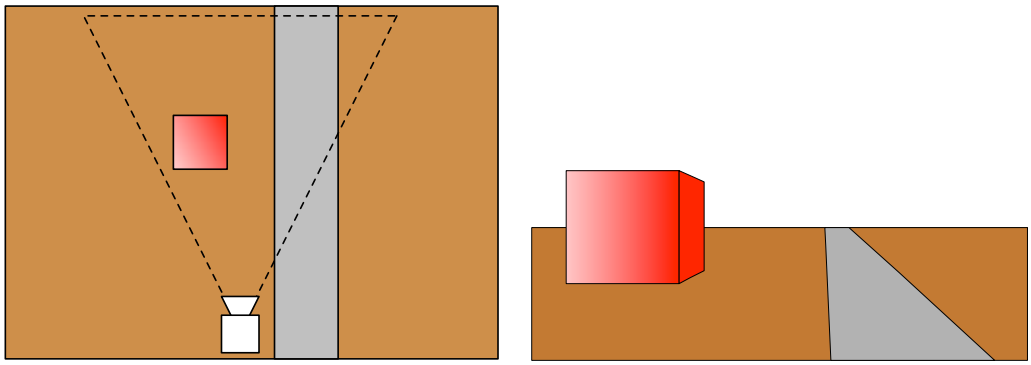
The "division by z" is formally described using **homogeneous coordinates** where a homogeneous vector $\vec{x}$ is defined as

$$\tilde{x} = (x, y, w), \tag{3.2}$$

where w is an arbitrary scale factor. The complementary **inhomogeneous** form is given by

$$\vec{x} = \left(\frac{x}{w}, \frac{y}{w}\right). \tag{3.3}$$

[2]Also considered as an intrinsic parameter is the *lens distortion*. This is a non-linear effect caused by the camera optics that makes straight lines appear curved and is generally most noticeable with wide-angle lenses. It will not be discussed here because it is corrected through pre-processing and would unnecessarily complicate the camera model.



(a) Overhead View (b) Camera View

Figure 3.4: An example of perspective distortion. In the overhead view (Figure 3.4a), all lines are either parallel or perpendicular to one another. However, from the camera's perspective (Figure 3.4b), the lines remain straight but are no longer parallel. Points farther away from the camera appear to be closer together.

This essentially states that $\tilde{x}$ and $\vec{x}$ only differ by scale. Note that this is very similar to (3.1) where the division by z occurs due to the similar triangles in Figure 3.3. For this reason homogeneous coordinates are used to represent the relationship between the world and what has been imaged by a camera.

Homogeneous coordinates also have some very useful properties besides being able to represent projective geometry. First, any point (x, y) can be trivially converted to a homogeneous form by

$$\tilde{x} = (x, y, 1). \tag{3.4}$$

This is computationally/mechanically useful in that it makes it possible to include a translation as part of a linear transform. Consider an invertible 2×2 matrix $\mathbf{A}$ and some vector $\vec{t}$. When using inhomogeneous coordinates, transforming and translating a point (x, y) is

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{A} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}. \tag{3.5}$$

The disadvantage to this form is that the translation is separate from the rotation/scaling operation. Because both are linear operations, it would be convenient to represent the transformation as a single invertible transform $\mathbf{H}$. However, by using homogeneous coordinates, this is trivial to do. If

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \vec{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.6}$$

then

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{3.7}$$

where $\mathbf{0}$ is an $N \times M$ matrix of zeros to "pad" the row and columns. Another advantage to this representation is that $\mathbf{H}$ can represent *any* invertible transform. This introduces the notion of a "homography" or a transformation between planes. Note that in the general case the homogeneous coordinate does not have to be '1' for either $\tilde{x}$ or $\tilde{x}'$.

Another important property of homogeneous coordinates is that they allow points and lines to be represented in exactly the same manner. Consider the standard form for the equation of a line

$$ax + by + c = 0, \tag{3.8}$$

where a, b and c are constants. In homogeneous vector form (3.8) becomes

$$\tilde{l} \cdot \tilde{x} = 0, \tag{3.9}$$

where $\tilde{l} = (a, b, c)$ and '$\cdot$' is the dot-product operator. Therefore, if for any given $\tilde{x}$ (3.9) is 0 then $\tilde{x}$ can be said to be a point on that line. Similarly, finding the intersection between two lines can now be done using this vector notation. Given two lines $\tilde{l}_a$ and $\tilde{l}_b$, the two will intersect at some point $\tilde{i}$. Therefore $\tilde{i}$ must satisfy

$$\tilde{l}_a \cdot \tilde{i} = 0 \tag{3.10}$$

and

$$\tilde{l}_b \cdot \tilde{i} = 0. \tag{3.11}$$

Intuitively, this means that $\tilde{i}$ is vector perpendicular to both $\tilde{l}_a$ *and* $\tilde{l}_b$. This is, by definition, the vector cross-product where the result is a vector orthogonal to two other vectors. Therefore $\tilde{i}$ is given by

$$\tilde{i} = \tilde{l}_a \times \tilde{l}_b, \tag{3.12}$$

where '$\times$' is the vector cross-product operator. By the same logic, given two points $\tilde{x}_a$ and $\tilde{x}_b$, the line through though the points is found by

$$\tilde{l} = \tilde{x}_a \times \tilde{x}_b. \tag{3.13}$$

The ability to swap between lines and points is known as duality [39](Ch. 2, pp. 29–30) and is useful in a number of different situations.

One thing that should be noticed, particularly about (3.12), is that when using homogeneous coordinates an intersection point can be found for *any* set of lines, even parallel lines. This is

somewhat counter-intuitive but can actually be explained as a property of projective geometry: given a set of parallel lines, they can be said to intersect "at infinity". Consider again (3.1). As z increases, the apparent distance between points will begin to decrease. For parallel lines, there will eventually be a point where they appear to meet[3]. Therefore, if $w = 0$ then it can be said that the point exists "at infinity". Furthermore, due to the division by zero, there is no inhomogeneous representation[4]. This makes it possible to use homogeneous coordinates to represent situations not normally possible with inhomogeneous coordinates.

3.3 Relating Multiple Cameras

In and of itself, projective geometry is useful because it relates the location of scene features to those imaged by a camera; however, it can be used to relatesthe scene as viewed by multiple cameras. This makes it possible to extract depth information based upon how the apparent position of a scene feature change between views.

Describing a camera is done through its **camera matrix**

$$\mathbf{P} = \begin{bmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \vec{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{KR} & \mathbf{K}\vec{t} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{3.14}$$

where $\mathbf{R}$ and $\vec{t}$ are the camera's rotation matrix (orientation) and translation vector (position) in the world coordinate frame, respectively. The matrix $\mathbf{K}$ is the camera's **calibration matrix** and is defined as

$$\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.15}$$

It should be noted that there are alternate definitions of $\mathbf{K}$, particularly ones that do not assume that the focal length is the same for both the horizontal and vertical axes. This occurs when the image has been scaled non-uniformly.

The purpose of $\mathbf{P}$ is two-fold. First, it positions the camera correctly in 3D space, relative to the observed features. Second, it ensures that a scaling and translation are done to account for the camera itself, as in (3.1). In other words, it directly models the effect of projecting $\vec{p}$ to $\vec{q}$ such that

$$\tilde{q} = \mathbf{P}\tilde{p}, \tag{3.16}$$

where $\tilde{q}$ is the homogeneous form[5] of $\vec{q}$ and $\tilde{p} = (x, y, z, 1)$, or the homogeneous form of $\vec{p}$. The

[3]This point is known as a "vanishing point" and exists on the "vanishing line" or plane, depending on the nature of the scene.

[4]These points are formally known as "ideal points" in the literature.

[5]Here $\tilde{q}$ is properly a 4-vector but because the use of homogeneous coordinates for $\tilde{p}$ is for mathematical conve-

homogeneous form is used simply to make the math convenient and $\mathbf{P}$ invertible.

When given two cameras there will be two camera matrices, $\mathbf{P}_a$ and $\mathbf{P}_b$, describing them. This in turn will mean that the feature $\tilde{p}$ will be projected onto two separate locations such that

$$\tilde{q}_a = \mathbf{P}_a \tilde{p}, \tag{3.17a}$$

$$\tilde{q}_b = \mathbf{P}_b \tilde{p}. \tag{3.17b}$$

However, it is useful to also know how to map $\tilde{q}_a \rightarrow \tilde{q}_b$ and vice-versa. This can be done very simply back-projecting $\tilde{q}_a$ to $\tilde{p}$ and projecting that result to $\tilde{q}_b$. Let

$$\tilde{p} = \mathbf{P}_a^{-1} \tilde{q}_a. \tag{3.18}$$

Therefore,

$$\begin{aligned} \tilde{q}_b &= \mathbf{P}_b \tilde{p} \\ &= \mathbf{P}_b \left(\mathbf{P}_a^{-1} \tilde{q}_a \right) \\ &= \underbrace{\left(\mathbf{P}_b \mathbf{P}_a^{-1} \right)}_{\mathbf{H}} \tilde{q}_a. \end{aligned} \tag{3.19}$$

Here $\mathbf{H} = \mathbf{P}_b \mathbf{P}_a^{-1}$ is a homography, or general transformation, that maps points in the image plane of camera a to points in the image plane of camera b. Since the last row of $\mathbf{P}$ is $\begin{bmatrix} \mathbf{0} & 1 \end{bmatrix}$, $\mathbf{H}$ represents a straightforward affine transformation between the two image planes in 3D space. But, due to the effects of the projection, the result is a *distortion* from the perspective of the cameras. Furthermore it can be an arbitrary 2D homography, i.e. a 3×3 matrix, without requiring any knowledge of the cameras. Unless otherwise stated, $\mathbf{H}$ is assumed to refer an arbitrary 2D homography rather than a 3D transformation.

Something that a homography cannot do is adequately explain the observed scene because it maps planes to planes. More specifically, the homography derived in (3.19) relates the *image planes* to one another. That means that if the actual features are not coplanar then it will not properly transfer features from one image plane to another. This is problematic during estimation because it implies a homography can only describe the relationship between certain types of scenes. What is needed is some method that will explain *how* features move between cameras for arbitrary scenes.

Figure 3.5 shows the geometry when two cameras image a scene. These can be either separate cameras or the same camera at different positions in time. Both are equivalent provided that the scene itself does not change, e.g. there are no moving objects. Let $\vec{c}_a$ be the location of camera 'A', $\vec{c}_b$ be the location of camera 'B' and $\vec{p}$ is the location of the feature. All of the vectors are inhomogeneous and are expressed in the world coordinate frame. The first thing to observe is that

nience, the fourth element is ignored.

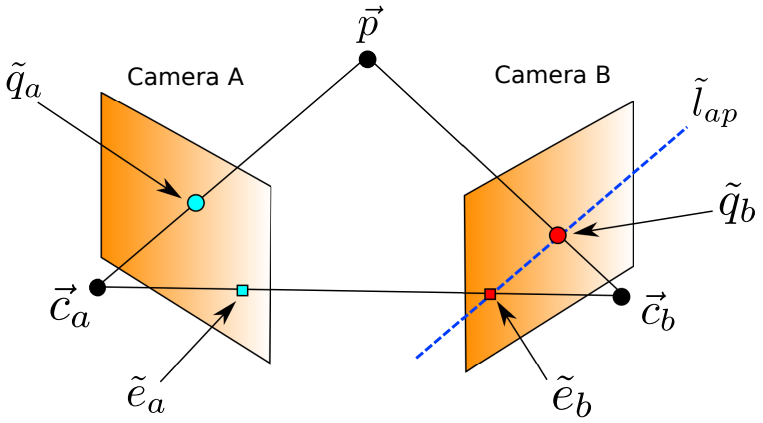


Figure 3.5: Geometry of a two-camera setup. The cameras can either be separate cameras or the *same* camera in two different positions; both are equivalent provided that the scene is static.

$\vec{c}_a \leftrightarrow \vec{p} \leftrightarrow \vec{c}_b$ forms a triangle and the line $\vec{c}_a \rightarrow \vec{p}$, $\tilde{l}_{ap}$, is visible from camera 'B'. Another thing is that the line from $\vec{c}_b \rightarrow \vec{c}_a$ intersects with the image plane of 'B' at $\tilde{e}_b$ (the point of intersection does not actually have to be present inside the boundary of the image). Finally, $\tilde{l}_{ap}$ passes through both the point of intersection and the location of the projected feature.

The main element that can be inferred from these observations is that the location of $\vec{p}$, as seen by camera 'B', is located somewhere along the line from camera 'A' to $\vec{p}$ projected onto the image plane of 'B'. This is known as the **epipolar constraint** and defines *how* features are transferred between cameras. $\tilde{l}_{ap}$ is known as an **epipolar line** because it represents the geometric interpretation of the constraint.

The relationship can be formalized as follows. First let

$$\tilde{e}_b = \mathbf{P}_b \tilde{c}_a, \tag{3.20}$$

where $\tilde{c}_a$ is just the homogeneous form of $\vec{c}_a$. To find $\tilde{l}_{ap}$, (3.13) can be used so that

$$\tilde{l}_{ap} = \tilde{e}_b \times \tilde{q}_b. \tag{3.21}$$

The cross-product can actually be represented by a skew-symmetric matrix [79](p. 42) where if $\vec{x} = (x, y, z)$ then $\left[\vec{x}\right]_\times$ is defined as

$$\left[\vec{x}\right]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}. \tag{3.22}$$

Therefore (3.21) becomes

$$\tilde{l}_{ap} = \left[\tilde{e}_b\right]_\times \tilde{q}_b. \tag{3.23}$$

However, because of the relationship derived in (3.19) it is possible to say that

$$\begin{aligned} \tilde{l}_{ap} \cdot \tilde{q}_b &= 0 \\ \tilde{l}_{ap}^T (\mathbf{H}\tilde{q}_a) &= 0 \\ \left(\left[\tilde{e}_b\right]_\times \tilde{q}_b \right)^T (\mathbf{H}\tilde{q}_a) &= 0 \\ \tilde{q}_b^T \left[\tilde{e}_b\right]_\times^T \mathbf{H}\tilde{q}_a &= 0. \end{aligned} \tag{3.24}$$

Recall that $\left[\cdot\right]_\times$ is skew-symmetric and therefore $\left[\cdot\right]_\times^T = -\left[\cdot\right]_\times$. Since the right side of the equation is zero, the negative sign (and by extension the transpose) can be ignored. Therefore the expression becomes

$$\tilde{q}_b^T \mathbf{F} \tilde{q}_a = 0, \tag{3.25}$$

where

$$\mathbf{F} = \left[\tilde{e}_b\right]_\times \mathbf{H} \tag{3.26}$$

is known as the **fundamental matrix**.

F is important because it relates how features in one image are transferred to another image. Unlike a homography, **F** maps points to lines rather than points to points. This can be seen in (3.25) where $\tilde{q}_b$ rests on the line given by

$$\tilde{l}_{ap} = \mathbf{F}\tilde{q}_a. \tag{3.27}$$

In practical terms, **F** is a point-to-line transformation that indicates the region where the corresponding feature, $\tilde{q}_b$, will be found. This is particularly important for disparity estimation because it restricts the search space when looking for the correspondence. Another useful feature of **F** is that it is symmetric, or given **F** from a known $\tilde{q}_a \rightarrow \tilde{q}_b$ mapping, the reverse mapping is simply

$$\tilde{q}_a^T \mathbf{F}^T \tilde{q}_b = 0. \tag{3.28}$$

This can be seen from the fact that (3.25) can be viewed as a dot product

$$\tilde{q}_b \cdot (\mathbf{F}\tilde{q}_a) = 0 \tag{3.29}$$

and since the dot product is commutative

$$\begin{aligned} \tilde{q}_b \cdot (\mathbf{F}\tilde{q}_a) &= (\mathbf{F}\tilde{q}_a) \cdot \tilde{q}_b \\ \ldots &= (\mathbf{F}\tilde{q}_a)^T \tilde{q}_b \\ \ldots &= \tilde{q}_a^T \mathbf{F}^T \tilde{q}_b, \end{aligned} \tag{3.30}$$

which is the same result as (3.28).

An important element of this derivation is it does not directly depend on the camera intrinsics, i.e. $\mathbf{K}$ is unknown. $\mathbf{H}$ was assumed to be a generic homography between image planes. Because the camera information is not available, the best it can do is provide a **projective reconstruction** where the relative distances will be correct but the *absolute* distances and angles between lines will not. If $\mathbf{K}$ is known, it is possible to obtain a **metric reconstruction** by upgrading an existing projective reconstruction so that the features will be identical to their actual 3D locations up to some rotation and scaling. In this case $\mathbf{F}$ is known as the **essential matrix** and has a slightly different form from (3.26).

One parameter that $\mathbf{F}$ is sensitive to is the offset between the cameras $\vec{b} = \vec{c}_b - \vec{c}_a$, more commonly known as the **baseline**. If the magnitude $\|\vec{b}\|$ is too small then the cameras are more or less in the same position. This means that there simply is not enough difference between the two images for $\mathbf{F}$ to be useful. $\mathbf{F}$ maps points to lines and so is a much weaker constraint than a homography which maps points to points, i.e. a one-to-many versus a one-to-one mapping.

Something that has not been mentioned yet is the nature of the camera projection from (3.20). This is a special construction known as an **epipole** and is the point where all of the epipolar lines intersect, i.e. they form a pencil of epipolar lines. Put another way, the lines "radiate" from the epipole. By definition the epipole is subject to

$$\tilde{q}^T \mathbf{F} \tilde{e}_b = 0 \; \forall \; \tilde{q} \in \text{Image B}. \tag{3.31}$$

Because this is a dot product, this implies that $\mathbf{F}\tilde{e}_b = \vec{0}$ and therefore $\tilde{e}_b$ is the (right) nullspace of $\mathbf{F}$. This same procedure can be used to show that $\tilde{e}_a$ is the left nullspace of $\mathbf{F}$. Since the mapping between 'A' and 'B' is easily swapped by transposing $\mathbf{F}$, the image that the epipole maps to depends on the convention being used. This also means that once $\mathbf{F}$ is known, finding the epipoles is simply a matter of finding its nullspaces, something that is easily done using a singular value decomposition (SVD) [39, 79]

3.4 Relationship Between Depth and Disparity

The exact relationship between disparity and depth has not yet been properly introduced. As was alluded to in Chapter 1.1, depth is inversely proportional to disparity, where disparity is the measured offset in feature correspondences in two or more images. This statement, however, is only true under a very specific set of circumstances and is why rectification (Chapter 2.1.1) is so important to disparity estimation.

First, assume two arbitrarily positioned cameras as was shown in Figure 3.5 and that they have identical calibration matrices $\mathbf{K}_a = \mathbf{K}_b = \mathbf{K}$. Also assume that camera 'A' is located at the origin of the world's coordinate system, i.e. $\mathbf{P}_a = \mathbf{K}$. If camera 'B' has some arbitrary rotation

$\mathbf{R}$ and translation $\vec{b}$ (the baseline) then its camera matrix has the same form as (3.14). With this information the feature point $\vec{p} = (x, y, z)$ is

$$\vec{p} = z\mathbf{K}^{-1}\tilde{q}_a. \tag{3.32}$$

This can be shown by rewriting (3.1). If $\vec{q}_a = (x_a, y_a)$ then

$$x_a = \frac{f}{z}(x + c_x) \tag{3.33a}$$

$$y_a = \frac{f}{z}(y + c_y). \tag{3.33b}$$

Reversing this it becomes

$$x = \frac{z}{f}(x_a - c_x) \tag{3.34a}$$

$$y = \frac{z}{f}(y_a - c_y). \tag{3.34b}$$

Therefore (3.32) is just the matrix form of the above expressions. An important point is that $\tilde{q}_a = (x_a, y_a, 1)$, or its homogeneous coordinate is '1'; this is fixed because $\vec{q}_a$ is the 2D feature coordinate in the image itself. This is also true for $\tilde{q}_b$, the location of the feature in camera 'B'. Therefore it can also be said that

$$\tilde{p} = z\mathbf{P}_b^{-1}\tilde{q}_b, \tag{3.35}$$

or

$$z\tilde{q}_b = \mathbf{P}_b\tilde{p}. \tag{3.36}$$

Again, as with the derivation of $\mathbf{H}$ in (3.19), $\tilde{q}_b$ and $\tilde{p}$ are nominally 4-vectors but the fourth dimension will be ignored at the end. This is a consequence of $\mathbf{P}_b$ being 4×4 and the last row of the matrix being $\begin{bmatrix}\mathbf{0} & 1\end{bmatrix}$ for mathematical convenience. However, (3.36) can be expanded into a more explicit representation with only 3-vectors by

$$z\tilde{q}_b = \mathbf{KR}\vec{p} + \mathbf{K}\vec{b}. \tag{3.37}$$

By substituting (3.32) into (3.37) it becomes

$$z\tilde{q}_b = z\mathbf{KRK}^{-1}\tilde{q}_a + \mathbf{K}\vec{b}$$

$$\tilde{q}_b = \mathbf{KRK}^{-1}\tilde{q}_a + \mathbf{K}\frac{\vec{b}}{z}. \tag{3.38}$$

The term $\mathbf{KRK}^{-1}$ is known as the **infinite homography**, $\mathbf{H}_\infty$, and has a special meaning that will be discussed shortly (Section 3.4.1).

The expression in (3.38) relates the observation of $\vec{p}$ in camera 'A' and 'B' to its distance from the camera, z. An important feature of (3.38) is that it is the expression of depth for *general* motion. That is, there is some arbitrary rotation $\mathbf{R}$ between the two viewpoints. However, if there is no rotation, $\mathbf{R} = \mathbf{I}$, then the expression reduces to

$$\tilde{q}_b = \tilde{q}_a + \mathbf{K}\frac{\vec{b}}{z}. \tag{3.39}$$

If the movement is further restricted such that it is only along the image's x-axis, i.e. $y_b - y_a = 0$, then the expression is further reduced to

$$x_b = x_a + f\frac{b_x}{z}. \tag{3.40}$$

With some simple re-arranging it finally becomes the traditional disparity expression[6]

$$d = f\frac{b}{z}, \tag{3.41}$$

where $b = b_x$ and $d = x_b - x_a$. Restricting motion to be along the x-axis is purely a computational convenience; it is much easier to search along horizontal rows than lines with arbitrary slopes. This also replicates the nature of the human visual system, where our eyes are horizontally offset. Image rectification is then simply the procedure that takes $\mathbf{R} \rightarrow \mathbf{I}$ to align the rows. Put another way, it makes the epipolar lines parallel to one another.

3.4.1 The Infinite Homography

The statement, "depth is inversely proportional to disparity," is a result of (3.41). But, as stated at the beginning of this section, this is *only* true if the motion is purely translational. This has a few other implications as well, such as the fact that (nominally) disparity values for translational motion cannot be negative since it implies that the imaged feature is somehow behind the camera![7] However, if $\mathbf{R} \neq \mathbf{I}$ then $d < 0$ is a valid solution. In fact, in the rectified case $d = 0$ implies that $z \rightarrow \infty$ while for general motion $d = 0$ is the plane of convergence. This is illustrated in Figure 3.6.

Essentially, there is some surface [80] where $d = 0$ and where objects *behind* the surface have negative disparity and those *in front* of the surface have positive disparity. The surface is not necessarily planar but provided that the degree of vergence is not too large, it will be approximately planar. If the convergence surface is interpreted as a display device, such as a 3DTV or other stereoscopic-capable device, then $d < 0$ would appear to be *inside* the screen while $d > 0$ would appear to be *outside* the screen. One thing to note is that this assumes a left-to-right camera

[6] The camera centre has been ignored because the camera parameters do not change and is essentially a constant offset.

[7] As stated before (Section 3.1) the sign depends on the choice of coordinate system. Here it is such that $z = 0$ is the location of the camera and $z > 0$ is in the direction of the scene.

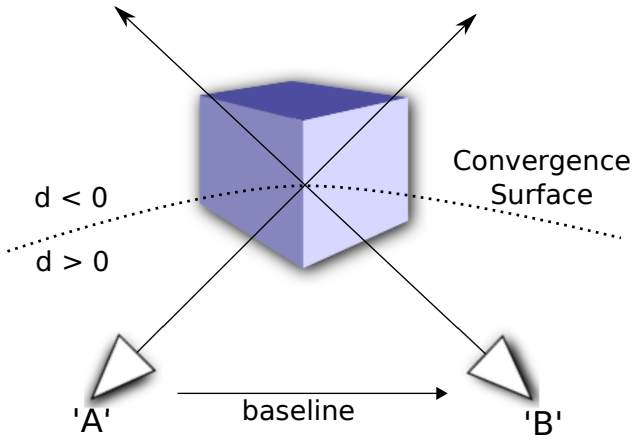


Figure 3.6: The basic disparity relationship in the case of general motion. The camera (or cameras) have undergone general motion with some arbitrary rotation.

motion. If the camera moves in the opposite direction then how the signs are interpreted must be reversed. This is important when attempting to automate the process of depth estimation.

The convergence surface is special in that it is a direct result of the infinite homography. Without any rotation, $\mathbf{H}_\infty = \mathbf{I}$ has no effect. The result is that the cameras' lines of sight are converging "at infinity" and so there is a simple, inverse relationship between depth and disparity. But the moment there *is* a rotation, the convergence point is no longer at infinity and so the relationship breaks down. Once the surface is a finite distance from the camera, it is possible for things to be behind it. Essentially $\mathbf{H}_\infty$ is the equivalent of a vanishing line in 3-space[8]. This causes what is a flat plane in projective 3-space (really a higher-order 4-space) to be curved in 3-space.

One other important aspect of $\mathbf{H}_\infty$ is that it is generated for *any* camera rotation, even if it is just about its own axis. A practical example of this would be a camera, mounted on a tripod. Rotating the camera on the tripod would produce a non-identity value for $\mathbf{R}$ and so a non-identity value of $\mathbf{H}_\infty$. However, because the tripod has not been moved, i.e. $\vec{b} = \vec{0}$, there is no disparity information. In this case (3.38) becomes

$$\tilde{q}_b = \mathbf{H}_\infty \tilde{q}_a, \tag{3.42}$$

which is identical to (3.19). To put it plainly: there are certain cases where it is *impossible* to find the depth of a scene even if the camera is moving. There will always be cases, particularly for hand held camera footage, where no good estimate of $\mathbf{F}$ exists because $\vec{b} = \vec{0}$ but $\mathbf{H}_\infty \neq \mathbf{I}$. Fortunately it is possible to detect these degenerate conditions and such a method will be presented in Chapter 4.

[8]It is relatively straightforward to define a projective 3-space like the projective 2-space discussed in this section. Please refer to [39](Ch. 3) for more details.

Chapter 4

Depth from Motion

DEPTH information is contained in, and can be extracted from, the relationship between features tracked over multiple frames. For instance, as something moves away from the camera, it becomes smaller. Similarly, something far away from the camera will move slower than an object close to it. While these are just some heuristics that could be applied to obtaining this information, it is far more useful to understand exactly how an imaged scene changes as a camera moves. As was discussed in Chapter 3, there is a well-defined relationship between apparent motion and depth for a moving camera. This relationship can be used to directly extract depth information.

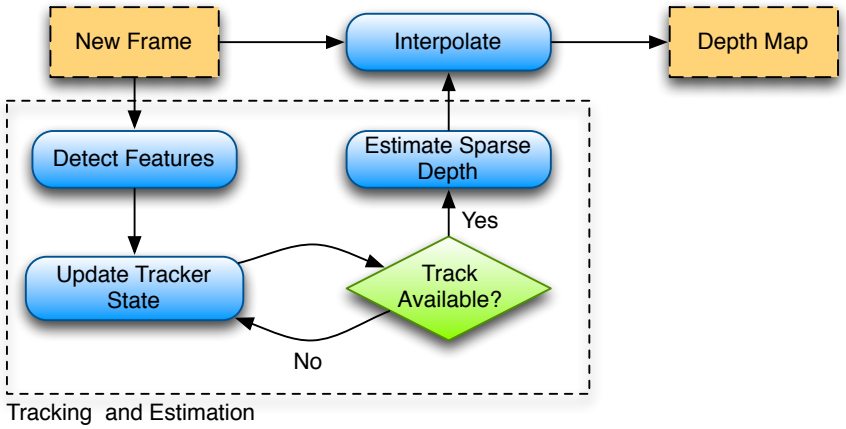


Figure 4.1: A high-level overview of the proposed depth estimation framework.

A flowchart of the proposed depth estimation method is shown in Figure 4.1. The depth estimation is intended to be online so that the only information required is the *current* frame in a video sequence. The framework does not require access to any prior frames. Instead, it buffers feature tracks for a set number of frames and once a set of tracks is complete, it estimates the sparse depth for the current frame. Once the sparse estimate is complete the system can then send the sparse estimates off for interpolation or storage.

This chapter will focus on how the *depth estimates* are obtained; interpolation will be left for Chapter 5. The method superficially resembles, and was inspired by, SfM but is quite different. Depth is being obtained through rectification directly instead of estimating the scene geometry first and *then* getting the depth. The argument being made is that by working from first principles it is possible to obtain those depth values directly. In other words, it is possible to skip a costly SfM reconstruction to generate depth maps that a user can modify or correct later on [76, 77, 78].

4.1 Feature Tracking

There are a number of different strategies for feature tracking, each with their own benefits and drawbacks. There is no one "right" way to do it. Tracking very much depends on the computer vision algorithm being implemented. When tracking for video-based SfM, for instance, these methods [16, 17] need to keep track of features across the *entire* sequence. As such, they need to be able to recover feature tracks even if a feature has been occluded or is temporarily outside of the frame boundaries. This is helpful for bundle adjustment as it provides more observations of a feature's position.

However, when estimating depth, long-term tracks, i.e. tracks that span tens or even hundreds of frames and may have even been occluded, are not as important. Because the goal is to determine the depth of a point for a particular camera view, whether or not a feature has been occluded at some point is irrelevant. The depth of the point depends entirely on the observed offset between views, as was explained in Section 3.4.

Therefore, so long as the tracks are long enough to establish a suitable baseline, then a simple KLT tracking scheme will be sufficient. Occlusion will still be an issue but mainly in that it restricts the length of the tracks. The proposed tracking framework is intended for short to medium range (10 to 30 frames) tracks. This is mainly to provide enough time for tracking while at the same time minimizing the total amount of buffering time. The main property of the tracker is that it maintains tracking for multiple frames *in parallel* and is the key to performing online depth estimation.

4.1.1 Detecting and Tracking Features

A video can be considered to be an ordered set of images

$$\mathcal{I} = \{I_0, I_1, I_2, \ldots, I_{N_F-1}\}, \tag{4.1}$$

where I_k is any particular frame and N_F is the number of total frames in the sequence. For a video *stream*, i.e. such as a television broadcast, N_F is simply a very large number and can be considered, for all practical purposes, infinite.

When performing feature tracking, the tracker must first be initialized with a set of features that can be reliably tracked. Generally, these are well-defined corners surrounded by texture [81].

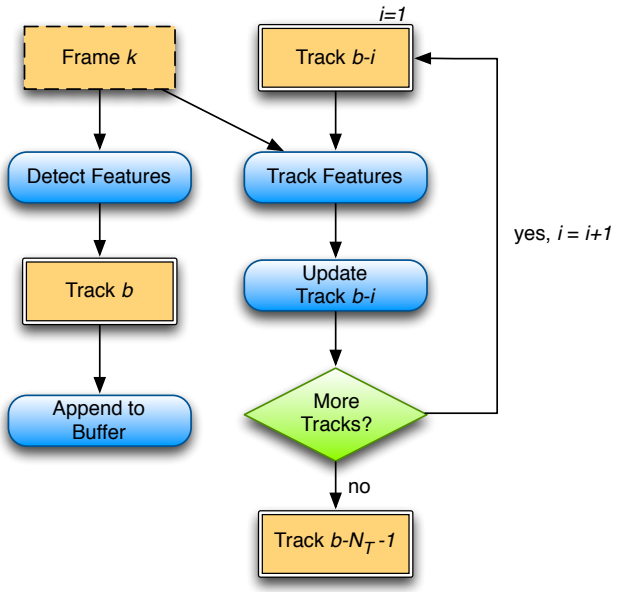


Figure 4.2: An overview of the parallel tracking procedure. Each incoming frame starts a new track and all existing tracks are updated together. The earliest tracking buffer is marked as ready after N_T frames.

The detector used in this work is the FAST corner detector [82] as that has been shown to obtain a large number of high-quality corners very quickly.

The tracking procedure, outlined in Figure 4.2, works as follows. First, the feature detector (e.g. FAST) generates a set of features $\vec{p}_{k|0}[n] \in \mathfrak{f}_0$, where $\vec{p}_{k|i}[n]$ is the 2D position of the n-th feature in frame $k+i$ (the purpose of this notation will become clear shortly). This set of features then becomes the first entry in the next available tracking buffer $\mathcal{F}_b = \{\mathfrak{f}_0\}$, where $b \in [0, N_T - 1]$. When frame $k+1$ arrives, the features in $\mathfrak{f}_0$ are tracked using KLT to produce the new set of features $\mathfrak{f}_1$.

Tracking errors are quite common because features become occluded or leave the frame. Handling errors is straightforward and there is a well-established method for doing this[1]. First, bi-directional tracking is used to verify that a feature tracked from $k \rightarrow (k+1)$ returns *back* to its original position if tracked from $(k+1) \rightarrow k$. If it does not then the feature is rejected. Next, patches around the feature in k and $k+1$ are compared using their L_2-norm. If the norm is too large then that feature is rejected. This is repeated $N_T - 1$ times until buffer b is filled with a set of features

$$\mathcal{F}_b = \{\mathfrak{f}_0, \mathfrak{f}_1, \ldots, \mathfrak{f}_{N_T-1}\}\,. \tag{4.2}$$

After $\mathcal{F}_b$ has been filled, it will contain a set of feature tracks that terminate early due to features being rejected by the error handling step. Only tracks that are contiguous and are N_T

[1]This verification strategy is actually built into off-the-shelf tracking packages such as Matlab's Computer Vision Toolbox (`www.mathworks.com/help/vision/index.html`) or OpenCV (`www.opencv.org`).

frames long are of interest. Therefore a new set $\mathcal{T}_b$ is created such that

$$\mathcal{T}_b = \{\mathfrak{t}_0, \mathfrak{t}_1, \ldots, \mathfrak{t}_{N_T-1}\}, \tag{4.3}$$

where $\vec{p}_{k|i}[n] \in \mathfrak{t}_i$ if and only if that feature was present in all of the entries of $\mathcal{F}_b$.

4.1.2 Tracking Pipeline

The tracker is designed to operate as a pipeline in the sense that each frame has its own set of tracks $\mathcal{T}_b$ running N_T frames into the future. This is done by generating a new set of features $\mathcal{F}_b$ for each frame and storing them into a circular buffer $\mathcal{B}$. When the next frame arrives, each feature set stored in $\mathcal{B}$ is updated, in parallel, using the KLT tracker. The process is illustrated in Figure 4.3. As shown, after waiting N_T frames for the first frame's buffer to fill up, the tracks leading up to frame $N_T - 1$ will be available.

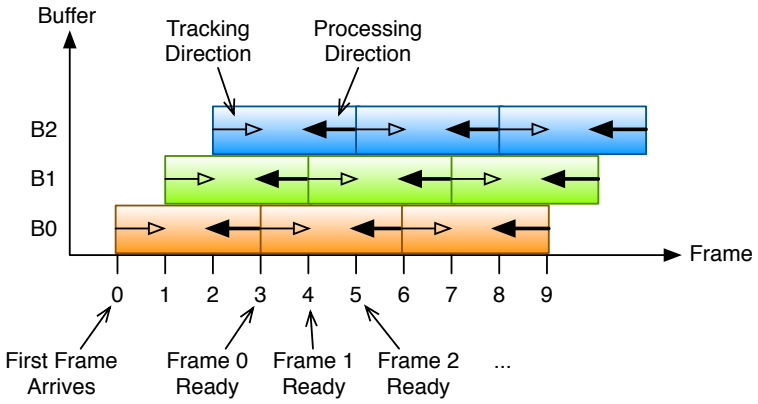


Figure 4.3: The tracker's processing timeline. Each frame is assigned its own buffer $\mathcal{F}_b$ to store feature tracks that originate from it. At each frame the positions are updated inside the buffer. After waiting N_T frames, here $N_T = 4$, the first buffer has been filled and is ready for processing. The features from $k + N_T - 1$ then replace the entry for frame k.

There are two things that should be noted about this tracking scheme. First, the number of frames to store in the buffer is $N_T - 1$. This is because a track will be ready on the N_T-th frame. Once that initial buffering is finished, a new set of tracks will be available for each future frame. Second is that the buffers need to be a uniform length. If the N_T was different for each track, or more specifically if it terminates early, then it would cause a "stall" in the pipeline. This means that N_T should be chosen so that it is long enough establish a baseline but not so long that it results in a very long buffering time.

4.2 Sparse Depth Estimation

Depth estimation is performed through a modified version Pollefeys' epipolar rectification [30] to sparse feature correspondences. The goal of the exercise, like all rectification methods, is to artificially align the cameras such that $\mathbf{R} = \mathbf{I}$ (Section 3.4). If the camera parameters do not change[2] between the two frames then the effect is that, after rectification, the infinite homography becomes negligible, i.e. $\mathbf{H}_\infty = \mathbf{I}$. This makes the observed offset inversely proportional to depth, as given by (3.41). While the exact scaling will be unknown, the relative depths can now be found.

4.2.1 Sparse Epipolar Rectification

Let $\vec{p}_k$ and $\vec{p}_l$ be two features matched[3] in frames k and l. Given these correspondences, provided that there were enough of them and sufficient camera motion, it is possible to obtain the fundamental matrix $\mathbf{F}_{k,l}$ between the two views. An important feature of $\mathbf{F}_{k,l}$, as discussed in Section 3.3, is that it maps points in one image to (epipolar) lines in another. Because the relationship is symmetric, the point-line relationship applies to both images equally. Or, put another way, the epipolar lines themselves can be mapped to one another. Rectification is the process of finding this mapping.

Epipolar rectification [30] does this through a simple Cartesian-to-polar coordinate transformation. The epipoles are contained in the fundamental matrix so that $\mathbf{F}_{k,l} \rightarrow \{\vec{e}_k, \vec{e}_l\}$, where $\vec{e}_k$ is the epipole in frame k and $\vec{e}_l$ is the epipole in frame l. Please note that the inhomogeneous[4] form is being used. Because, by definition, all epipolar lines intersect at an epipole, they can be viewed as the origin of a coordinate system. A transformed vector $\vec{\rho}_k$ can be defined as

$$\vec{\rho}_k = (\rho_k, \theta_k)\,, \tag{4.4}$$

where

$$\rho_k = \|\vec{p}_k - \vec{e}_k\|, \tag{4.5}$$

$$\theta_k = \angle\,(\vec{p}_k - \vec{e}_k)\,. \tag{4.6}$$

The same definition is used for frame l. This is a simple transformation that converts the feature positions into a polar coordinate system with the image's epipole at the centre. However, in the transformed space, the epipolar lines are now *parallel* to one another. Because this transform is

[2] This is a reasonable assumption for video particularly when it comes to hand-held camera footage. The focal length on mobile phone cameras is fixed and any zooming, if done, is a post-processing effect that scales the frame uniformly.

[3] These features can be obtained either through tracking or through direct feature matching; it is not relevant at this point in the discussion.

[4] $\mathbf{F}_{k,l}$ is 3×3 so when the nullspaces are initially obtained, they are represented by homogeneous vectors.

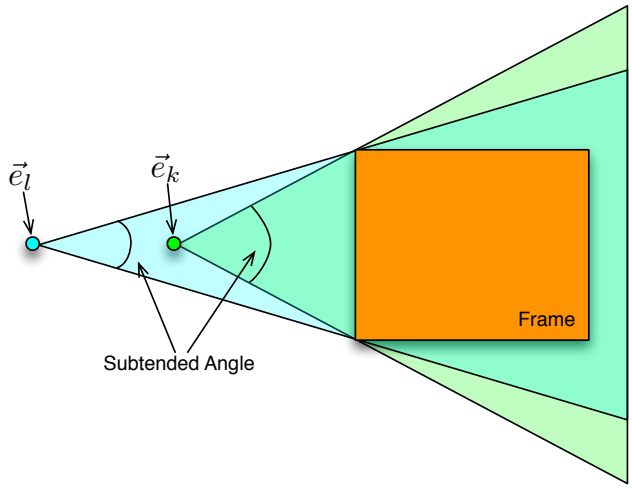


Figure 4.4: Angles subtended by the set of all possible epipolar lines. The maximum and minimum angles depend on how close an epipole is to the image boundaries. Therefore the range of angles is smaller for a more distant epipole and vice-versa.

obtained from the epipoles themselves, there are no singularities if the epipoles are inside of the image boundaries, something that can happen with planar homographies.

The original epipolar rectification method operated directly on the images and much of the work in [30] was rendering the transformed/rectified images and establishing a way to map the results back into the original image coordinate system. Here the downside of a non-linear approach becomes apparent: while finding the transform is trivial, applying it is quite difficult. Fortunately this problem is non-existent when working with *sparse* features. Because a direct mapping between the features already exists, the mapping is still maintained after the transformation.

With sparse rectification, the angular component θ_k has no impact on the actual disparity. The radial component ρ_k encodes how far a feature is from the image's epipole. Therefore, the disparity between features in k and l is defined as

$$\Delta_{k,l} = \rho_l - \rho_k. \tag{4.7}$$

While there *is* a difference between θ_k and θ_l, it is of the form

$$\theta_l = a\theta_k + b, \tag{4.8}$$

where a and b are some constant scaling and translation factors. Consider the case where the epipoles have the same vertical coordinate but different horizontal coordinates; this is illustrated in Figure 4.4. The angles subtended by the epipolar lines differ but their average value does not. Aligning the epipolar lines is a matter of scaling and offsetting the values of θ_k so that it equals θ_l. But when sparse matches are available, that alignment is unnecessary.

While the transformation is insensitive to the epipole locations, the values of $\Delta_{k,l}$ unfortunately are not. Specifically, an epipole's homogeneous coordinate is $\tilde{e} = (x, y, w)$ and for an image pair that is already rectified, $w \approx 0$. This is a consequence of the epipolar lines already being parallel. By definition the epipole is the intersection of all epipolar lines and if the lines are parallel, then that epipole is an ideal point (Section 3.2). The problem this poses is that the epipoles can be either on the same side of the image or on opposite sides; both are "infinitely" far away and so result in parallel epipolar lines.

To accommodate for this, a simple transformation can be applied to the values of ρ_k. This is defined as

$$\rho'_k = \begin{cases} 2\mu_\rho - \rho_k & \left\{|\angle(\vec{e}_k)| < \frac{\pi}{2}\right\} \oplus \left\{|\angle(\vec{e}_l)| < \frac{\pi}{2}\right\} \\ \rho_k & \text{Otherwise} \end{cases}, \tag{4.9}$$

where μ_ρ is the mean value of the set of ρ_k values and $\oplus$ is the exclusive-or operator. This mirrors ρ_k around the centre of the point cloud so that it creates the effect of both epipoles being on the same side of the image. However, this can still result in the disparity magnitudes being inverted between frames because the epipole position depends on the estimate of $\mathbf{F}_{k,l}$ and is notoriously inconsistent[5]. One final adjustment can be applied to the disparity values themselves to ensure that the magnitudes are consistent. Specifically, if the $\vec{e}_k$ is not located on the *right* side of the Cartesian plane then the values of $\Delta_{k,l}$ are inverted (mirrored) by

$$\Delta'_{k,l} = \begin{cases} \max(\Delta_{k,l}) + \min(\Delta_{k,l}) - \Delta_{k,l} & |\angle(\vec{e}_k)| < \frac{\pi}{2} \\ \Delta_{k,l} & \text{Otherwise} \end{cases}. \tag{4.10}$$

While this can result in the disparity estimates being the inverse of what they should be (large values are farther than small values), it at least ensures that the estimates are stable so long as the camera motion itself does not change.

4.2.2 Two-frame Example

Consider the image pair shown in Figure 4.5. Here the camera has undergone a non-trivial motion, i.e. more than just a translation. Therefore, the disparity values will be both positive and negative and the epipolar lines will not be parallel.

Figure 4.6 shows the $\mathbf{F}_{k,l}$-estimate inliers from the SURF [14] feature correspondences. The complex motion is evident in how the features move differently depending on their position in the frame. The purpose here of using sparse feature matching instead of the feature tracking (Section 4.1) is simply to illustrate that *how* the matches are obtained is not particularly relevant to the depth estimation. More detail on how $\mathbf{F}_{k,l}$ is estimated will be given in Section 4.2.3 but for now

[5]The fundamental matrix is rank-deficient and has no inverse. The result is that for any given set of matches between k and l, there can be *multiple* solutions to $\mathbf{F}_{k,l}$, even if no actual solution exists (such as with pure rotation).

(a) Frame 80 (b) Frame 93

Figure 4.5: Two frames from the "Inuksuk" sequence. The camera motion is non-trivial, meaning that it underwent more than just a translation.

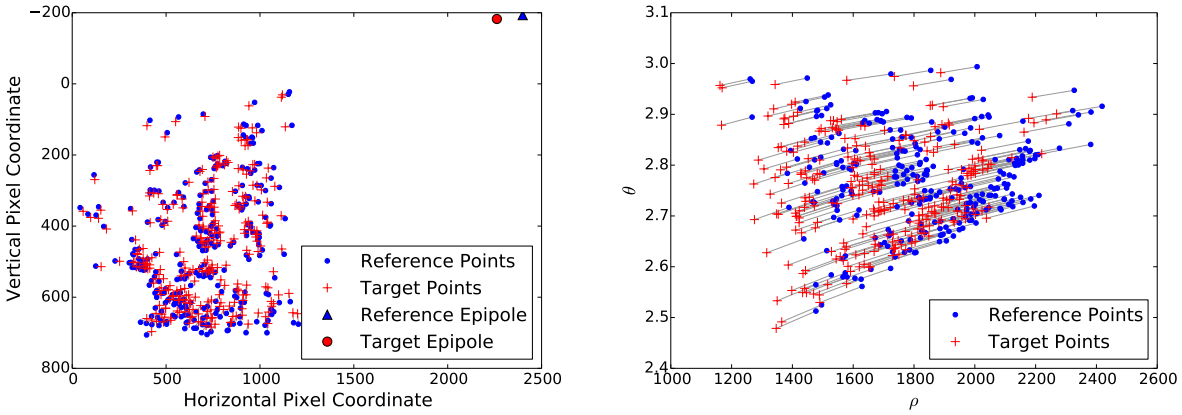
Figure 4.6: Inlier SURF correspondences for the image pair in Figure 4.5. The coloured dots indicate the position of the feature in Figure 4.5b.

it is sufficient to know that the estimation procedure segments the feature matches into inliers and outliers.

The locations of the features, both before and after rectification, are shown in Figure 4.7. In this example, both epipoles are located outside of the image boundaries (Figure 4.7a) but are close enough to cause a noticeable distortion in the rectified results (Figure 4.7b). Note that because of the epipole positions, the range of potential angles is larger in the target frame (Frame 93) than those in the reference frame (Frame 80). It is also slightly "off angle" and so there is a minor difference in the average value of θ between the two rectified feature sets.

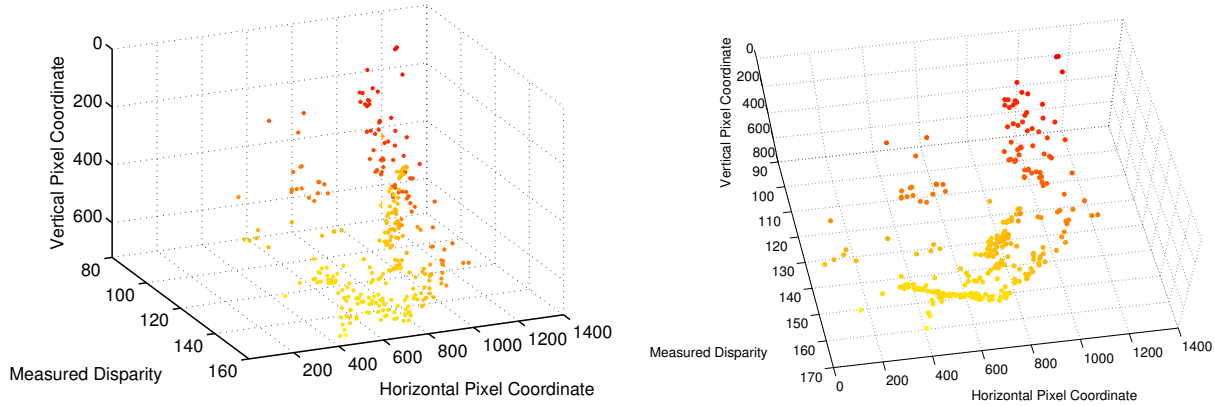
The final disparity estimate for the frame pair is shown in Figure 4.8. For this example the magnitude of the disparity estimates correspond to the expected scene depth: features with a greater disparity are closer to the camera. If the reference and target frames were swapped then small disparity values would correspond to features closer to the camera from the simple fact that it would effectively negate[6] the values of $\Delta_{k,l}$

[6]Even though the $\mathbf{F}_{k,l}$ would be different, the epipole locations in the images would remain relatively constant



(a) Original Positions (b) Rectified Positions

Figure 4.7: Original feature positions versus rectified positions for correspondences in Figure 4.6. The positions of the original features are given in pixels while the for the rectified features, ρ is in pixels and θ is in radians.



(a) Oblique View (b) Top-down View

Figure 4.8: The spatial distribution of the final disparity measurements. The colour coding is such that dark red corresponds to small values and bright yellow/white are large values. Two views have been provided to better demonstrate the approximate 3D structure of the scene.

Please note that for the scatterplots in Figure 4.8, that the value of $\delta_{k,l}$ has been used as the z-coordinate value. This leads to a somewhat distorted view of the scene, e.g. the inuksuk should be perpendicular to the ground. However, this clearly shows that the relative depth ordering has been successfully recovered.

since they are a result of the camera geometry.

4.2.3 $\mathbf{F}_{k,l}$-estimation and Model Verification

So far there has been no discussion on exactly how $\mathbf{F}_{k,l}$ is obtained. Finding it is a well-studied problem [39, 83, 84, 85, 86] but the various methods can all be summarized as such: given two sets of point correspondences, $\mathfrak{f}_k$ and $\mathfrak{f}_l$, find $\mathbf{F}_{k,l}$ such that

$$\tilde{p}_l^T \mathbf{F}_{k,l} \tilde{p}_k = 0 \,\forall\, \tilde{p}_k \in \mathfrak{f}_k \text{ and } \tilde{p}_l \in \mathfrak{f}_l. \tag{4.11}$$

This is simply ensuring that the epipolar constraint is satisfied for all point matches. The minimum number of matches is between 5 and 8, depending on the exact method being used.

In Section 3.3, $\mathbf{F}_{k,l}$ was shown to be a mapping between points in one image to lines in another image. As such, $\mathbf{F}_{k,l}^{-1}$ does not exist because the fundamental matrix is of rank-2 [39](Table 9.1). It is for the same reason why the reverse mapping is given by the transpose $\mathbf{F}_{k,l}^T$ instead of the actual matrix inverse. Because the mapping is not one-to-one (point-to-point), estimates of $\mathbf{F}_{k,l}$ are quite sensitive to noisy feature matches. Specifically the detected features $\vec{p}^{\,d}$ will be related to the feature's true location by

$$\vec{p}^{\,d} = \vec{p} + \vec{\epsilon}, \tag{4.12}$$

where $\vec{\epsilon} \sim N(\vec{0}, \mathbf{\Sigma})$. Therefore the epipolar line for $\vec{p}$ will be

$$\begin{aligned} \tilde{l} &= \mathbf{F}_{k,l} \tilde{p}^{\,d} \\ &= \mathbf{F}_{k,l} \left(\tilde{p} + \tilde{\epsilon} \right). \end{aligned} \tag{4.13}$$

While the location of the epipole will not change, the angle of $\tilde{l}$ through the epipole *will*. This in turn defines an envelop of *potential* epipolar lines for $\vec{p}^{\,d}$. A full treatment of this subject can be found in [39](pp. 298–302). This however, is for the ideal case where the ground truth, or a very good approximation to the ground truth, $\mathbf{F}_{k,l}$ is available. During estimation $\vec{\epsilon}$ makes finding an accurate $\mathbf{F}_{k,l}$ more difficult. The question then becomes how to best find $\mathbf{F}_{k,l}$ when the data is known to be corrupted.

To find the best possible $\mathbf{F}_{k,l}$ given noisy data, robust estimators are often used [87]. One of the most common and well-known methods is Random Sample Consensus (RANSAC) [88]. RANSAC is a relatively simple iterative procedure that randomly generates a model from a small subset of a larger dataset and determines how much of the remaining dataset agrees with the model. Features that agree with the dataset are called "inliers" while features that do not agree with the model are "outliers". A datum is said to "agree" with a model when the error between it and the model is less than some threshold or confidence value. Generally RANSAC is terminated once enough inliers have been found, i.e. there is a high confidence that model describes the dataset well.

In the context of computer vision, a well-fitting estimate of $\mathbf{F}_{k,l}$ is one where the majority of features in frame l are close to the epipolar lines generated by features in frame k. RANSAC

tends to find these high-quality models, provided that there are enough samples. The approach is generally very effective and not only used widely throughout the literature it is also implemented in many popular computer vision packages.

As stated in Section 3.4.1, the features are transferred from one image to another by the translation of the camera (the baseline $\vec{b}$) and its rotation (the infinite homography $\mathbf{H}_\infty$). If $\|\vec{b}\|$ is small then $\mathbf{H}_\infty$ will dominate and there will be little to no usable depth information. However, if $\|\vec{b}\|$ is suitably large, but not so large that most of the features are no longer visible, then $\mathbf{H}_\infty$ will not be a good model of the observed motion. In fact, $\mathbf{F}_{k,l}$ will provide a much better model because it encodes the notion of features moving relative to their distance from the camera. This poses a problem when estimating $\mathbf{F}_{k,l}$ because while RANSAC can obtain an estimate for any frame pair, it does not mean that the estimate will be of any use. All it guarantees is that this estimate is the one that produces the *best* model given the data. If the data cannot be modelled by the method RANSAC is using then that model will be useless.

However, there is a well-known method developed by Torr et al [89, 90, 91] to handle such a problem. The Geometrically Robust Information Criterion (GRIC) can be used to score models, such as a fundamental matrix, and assess how well they fit the given data. Applying GRIC for model selection is done as follows:

1. Obtain an initial estimation of $\mathbf{F}_{k,l}$ through RANSAC.
2. From the *inliers* of $\mathbf{F}_{k,l}$, generate a homography $\mathbf{H}_{k,l}$ that provides a one-to-one mapping of the inliers in k to the inliers in l.
3. Generate $GRIC(\mathbf{F}_{k,l})$ and $GRIC(\mathbf{H}_{k,l})$ from the two models.
4. Reject $\mathbf{F}_{k,l}$ if $GRIC(\mathbf{F}_{k,l}) > GRIC(\mathbf{H}_{k,l})$.

It is important to use the obtained inliers because if all of the features were used then $\mathbf{H}_{k,l}$ would *always* be the better model. This is a consequence of it being a point-to-point mapping. What this will specifically detect are cases where $\|\vec{b}\|$ is too small for $\mathbf{F}_{k,l}$ to be reliable. This is intended to reduce the likelihood of poor $\mathbf{F}_{k,l}$ estimates from being used in the disparity estimation. A summary of how the GRIC scores are obtained can be found in Appendix A.

It should be noted that this procedure is the simplest possible application of GRIC-style model selection. It is common in SfM methods to also check the quality of the estimated camera projection matrices [35, 69, 92]. This is mainly to minimize the effect of dominant planes during bundle adjustment. But because the camera pose, i.e. $\mathbf{P}$, is not crucial to estimating the depth discussed in Section 4.2.1, this is not as much of a problem.

4.3 Track Processing and Depth Aggregation

Once a track is available, the frames are processed in a pair-wise fashion using what is essentially a **parallel chain** [53]. This term comes from a structure recovery method known as "projective factorization"[7]. A parallel chain refers to the fact that pairs are processed "in parallel", in the sense that the sequence will be $k \rightarrow k+1, k \rightarrow k+2, k \rightarrow k+3,$ etc. This is in contrast to a **serial chain** where the processing is performed with neighbouring frames, i.e. $k \rightarrow k+1, k+1 \rightarrow k+2, k+2 \rightarrow k+3,$ etc. The advantage, in this case, of using a parallel chain is that each pair gradually increases in temporal distance. For instance if the camera is moving slowly, it is unlikely that there will be enough motion between neighbouring frames. However, by using more distant frames, there is a better likelihood that there *will* be enough motion.

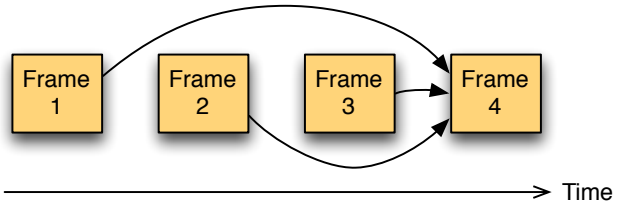


Figure 4.9: Track processing as a parallel chain. Processing is performed forwards in time where the current frame (the last frame in the track) is used as the reference frame.

An outline of the parallel chain being used is shown in Figure 4.9. The last frame in the chain is used as as the reference frame as it corresponds to the latest available frame. This is mainly to ensure that the convention for depth established in Section 4.2 is maintained. This will produce a set of sparse disparity estimates for frame k

$$\mathcal{D}_k = \left\{ \Delta_{k-1,k}, \Delta_{k-2,k}, \ldots, \Delta_{k-(N_T-1),k} \right\}. \tag{4.14}$$

Because the model selection/verification from Section 4.2.3 will reject frames if $\mathbf{F}_{k,l}$ does not adequately describe the motion between the frame pair, the number of elements in $\mathcal{D}_k$ will often be less than $N_T - 1$. In some cases, it is even possible that $\mathcal{D}_k$ will contain *no* disparity measurements. This is simply a limitation of the proposed method; if the camera does not move over the duration of the tracking window then that frame will not have any depth information.

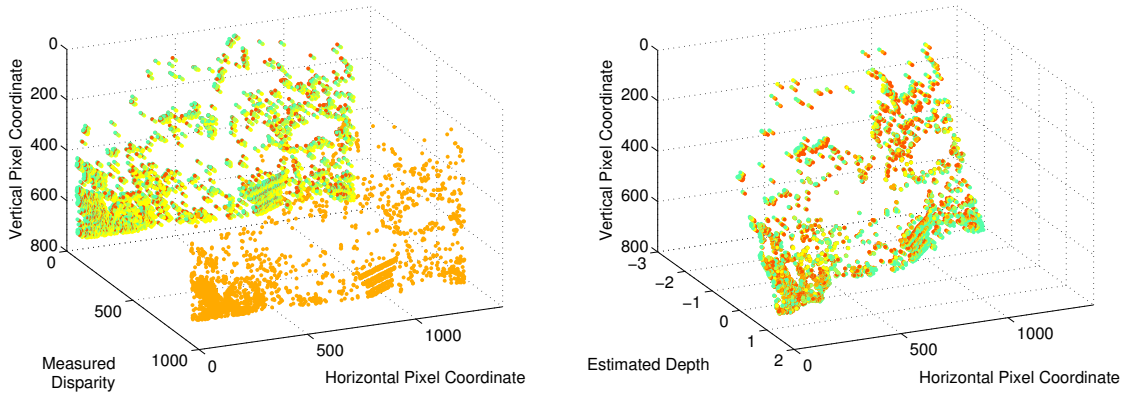
Once $\mathcal{D}_k$ is available, the disparity measurements need to be aggregated into a single set of sparse depth estimates so that a dense depth map can be generated. Prior to the aggregation, however, it is useful to first standardize the elements of $\mathcal{D}_k$ such that

$$\Delta'_{k-i,k} = \frac{\Delta_{k-i,k} - \mu_{k-i,k}}{\sigma_{k-i,k}}, \tag{4.15}$$

[7]Projective factorization recovers structural information like SfM but it is a fundamentally different method. A brief discussion is provided in Section 2.2.2.

where $\mu_{k-i,k}$ and $\sigma_{k-i,k}$ are the mean and standard deviation of the elements in $\Delta_{k-i,k}$, respectively. Please note that $0 \leq i \leq (N_T - 1)$. This is done so that the standardized *depth* $\Delta'_{k-i,k}$ has zero mean and unit variance. In other words, this is computing the z-score of $\Delta_{k-i,k}$.

Figure 4.10 shows why this is done. As the temporal distance increases, so will the magnitudes of $\Delta_{k-i,k}$. Disparity is a measurement of distance between corresponding features and so as they grow further apart, so will $\Delta_{k-i,k}$. Another is that the variability of the epipole locations can result in the magnitudes of $\Delta_{k-i,k}$ to, in some cases, become completely unrelated to the actual motion. This is evident in Figure 4.10a. This mainly occurs when the motion is purely translational and the epipoles are located very far from the image boundaries.



(a) Original $\mathcal{D}_k$ (b) Standardized $\mathcal{D}_k$

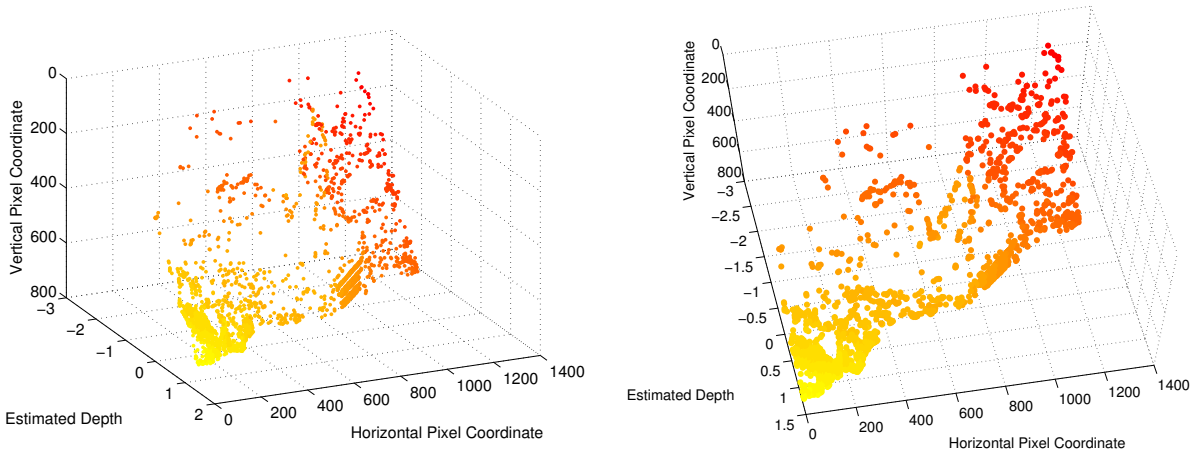
Figure 4.10: Comparison between the original and standardized $\mathcal{D}_k$ values. The magnitudes of the disparity measurements can vary wildly depending on the epipole locations and the overall motion. After calculating their z-scores, the estimates are now mainly in agreement.

The purpose of using z-scores, as opposed to scaling the values to be on $0 \leq \Delta'_{k-i,k} \leq 1$, is that after standardization, each set of $\Delta'_{k-i,k}$ values is guaranteed to have the same scaling and offset after the transformation. This is not true if the values were normalized to be on $[0, 1]$ because both the variance *and* mean would be different for each set of $\Delta'_{k-i,k}$. This in turn would skew the results of the aggregation stage and potentially create outliers where there were none. Recall from (3.41) that

$$d = f\frac{b}{z}. \tag{4.16}$$

The assumption when performing the tracking is that z and f are constant since one is a property of the scene and the other is a property of the camera optics. However, the camera is moving and so the baseline will vary, i.e. $d \propto b$. Computing the z-score is removing the effect of the b on the disparity measurements. This ensures that when the aggregation is performed that the disparity estimates, now depth estimates, are scaled properly.

Calculating the aggregated depth is done by taking the median value of each set of standardized



(a) Oblique View

(b) Top-down View

Figure 4.11: Aggregated depth estimates. The aggregation is performed by applying (4.17) to the tracks.

measurements along a particular track. The final depth estimate for the n-th feature in frame k is defined as

$$d_k[n] = \text{median}\left(\mathcal{D}'_k[n]\right), \tag{4.17}$$

where $\mathcal{D}'_k[n]$ is the set of $\mathcal{D}'_k$ values associated with the n-th feature. The advantage to this is that it is very simple to implement and the median operator is highly effective at removing outliers. In the case where there are two clusters of depth candidates, the median will choose the larger of the two clusters as this is the more likely depth value. This particular technique has been used in a machine learning context to produce a new depth map by combining several candidate depth maps [67].

Figure 4.11 shows the resulting depth estimates after the aggregation has been performed. As expected, this is also similar to the pairwise results presented Figure 4.8. The differences stem from the use of the FAST corner detector to obtain the features being tracked; it looks for image corners while the SURF detector in the earlier example looks for blobs.

Chapter 5

Depth Map Generation

Once the sparse depth estimates have been obtained they can be used to generate dense depth maps based on the underlying image content. The goal is to produce a depth map similar to what was shown in Figure 1.1 back in Chapter 1. Typically this problem is formulated as a form of anisotropic diffusion [93] where some input labelling is propagated throughout an image using strong edges as natural boundaries between labels or smoothly varying between labels in homogeneous regions.

The following definitions will be used throughout this chapter. First, let

$$D_s(x,y) = \begin{cases} d_k[n] & \text{if feature } n \text{ is located at } (x,y) \\ D_{\text{un}} & \text{Otherwise} \end{cases} \tag{5.1}$$

be a "sparse labelling image". This is an image where only a subset of the pixels have known depth values. D_{un} is a placeholder value that is used to indicate that the pixel at (x,y) has no associated depth value. Next, let

$$C(x,y) = \begin{cases} 1 & D_s(x,y) \neq D_{\text{un}} \\ 0 & \text{Otherwise} \end{cases}, \tag{5.2}$$

be an indicator map (binary-valued image) that indicates where pixels have known and unknown depth values. Note that while $D_s(x,y)$ implicitly defines $C(x,y)$, it is convenient to define it directly. Finding the dense depth $D(x,y)$ is then the process of creating some operator $\mathcal{O}$ that, given $D_s(x,y)$, will estimate a depth value for each possible value of (x,y) using the information provides in the image $I(x,y)$.

This is in fact a well-studied problem in image processing: user-guided, or supervised, image segmentation. A user provides some seed labelling to indicate objects of interest in an image and the algorithm will propagate the labelling in such a way that some cost function, often based on image edge information or colour distribution, is minimized. This idea has already been exploited

mainly for used-guided depth map generation [9, 61, 74, 76], including in the author's own prior work in [9, 11].

5.1 Label Propagation Methods

This section will discuss two strategies for propagating the sparse labels in $D_s(x, y)$ in order to produce a dense depth map. The first strategy is to use regularization, also known as variational methods. These are quite popular in image processing as they are able to find globally optimal solutions by by minimizing some energy functional $E(\vec{x})$, where $\vec{x}$ is the state vector, or solution being sought. The other strategy is to use edge-aware filters to perform the propagation. This is a consequence of these filters having a strong relationship to anisotropic diffusions. However, as will be shown, not all edge aware filters are appropriate for this task.

5.1.1 Regularization

In [9] and [11], user labels were converted into dense depth maps using a combination of Random Walks [3] and Graph Cuts [2]. These are global optimization techniques that minimize some objective function by using the user labelling as a set of initial conditions[1]. As demonstrated by Couprie et al [94], both Graph Cuts and Random Walks are ultimately part of the same class of Markov Random Fields (MRF) problems. Their formulation is slightly different than the conventional variational approach as they examine the problem from a graph-theory/combinatorics angle. The resulting system of equations is the same, however.

If the image is an undirected, N-connected graph $G = (V, E)$ composed of vertices (pixels) $v_i \in V$ and edges $e_{i,j} \in E$ then the Graph Cuts/Random Walks problem class is defined as the solution to

$$\underset{\vec{x}}{\operatorname{argmin}} \underbrace{\sum_{v_i \in V} w_i^p \left| x_i - y_i \right|^q}_{\text{Data Term}} + \lambda \underbrace{\sum_{e_{i,j} \in E} w_{i,j}^p \left| x_i - x_j \right|^q}_{\text{Smoothness Term}}, \tag{5.3}$$

where $\vec{x}$ is the solution vector, e.g. $D(x, y)$, and $\lambda > 0$. Here, $w_{i,j}$ is the weight of $e_{i,j}$ while w_i represents the affinity of a particular label to a given vertex (y_i is the label value assigned to v_i). Often this is a binary value where $w_i = 1$ if the pixel is labelled and $w_i = 0$ if it is not.

The p and q terms are special in that they chose *what* algorithm is being performed. Table 1 in [94] summarizes the effects of these parameters. Of particular interest is the case when $q = 2$ and $p > 0$ and finite (Couprie et al provide further information on the other cases). Here the solution to (5.3) becomes

$$\mathbf{M}\vec{x} = \vec{b}, \tag{5.4}$$

[1] Specifically, Random Walks is a solution to the combinatorial Dirichlet problem while Graph Cuts solves the "max-flow/min-cut" problem.

where $\mathbf{M}$ is derived from the graph Laplacian[2] and $\vec{b}$ is a vector with the boundary conditions. The exact contents of $\mathbf{M}$ and $\vec{b}$ will depend on the constraints being imposed on the system.

The key advantage to regularization, i.e. global optimization, is that it is relatively straightforward to impose a variety of different constraints on the final solution. For instance, StereoBrush [73] uses the same basic setup as Random Walks along with two other constraints. First, the relative influence of a label is reduce the farther an unlabelled pixel is from a labelled pixel. Second, edge discontinuities are explicitly modelled as part of the minimization so that propagating a label across a strong edge becomes extremely unlikely. Similarly, Liao et al [77] and Guttmann et al [61] use the Random Walks constraints along with constraints based on optical flow to adjust the depth of moving objects as part of the optimization.

5.1.2 Filtering

An alternative to regularization is edge-aware filtering. Edge-aware filters, the bilateral filter [55] in particular, have been shown to have a strong connection to anisotropic diffusion [95]. However, the propagation process itself does not require edge-aware filters. Using an edge-aware filter is in order to ensure that the underlying image content guides how the labels are propagated, similar to how anisotropic diffusion operates.

Filter-based propagation can be traced back to the Inverse Distance Weighting (IDW) method proposed by Shepard et al [96]. Fattal et al [56], Gastal and Oliveira [57] and Lang et al [26] modified the IDW approach to use general filters, in more or less the same way. This type of interpolation is similar to traditional interpolation (upsampling) in signal processing except that in upsampling the samples are uniformly spaced on a regular lattice. In filter-based propagation this is not the case and the known values are positioned randomly.

To understand this type of propagation, it is easier to first consider the one-dimensional case. Let $x[n] \in \mathbb{R}$ be a real-valued, discrete signal and let $h\{x[n]\}$ be a linear filter that can operate on $x[n]$. It should be noted that no restriction is made on $h\{\cdot\}$ being translation invariant and so the filtered output is $y[n] = h\{x[n]\}$ and not $y[n] = x[n] * h[n]$, where $h[n]$ is some convolution kernel[3]. The labelling signal can then be defined as

$$s_i[n] = L_i \delta[n - p_i], \tag{5.5}$$

where $L_i \in \mathbb{R}$ is the value of the particular seed label, $p_i \in \mathbb{Z}$ is position of the label and $\delta[n]$ is the Kronecker delta, defined as

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}. \tag{5.6}$$

[2]The graph Laplacian is defined as $\mathbf{L} = \deg(\mathbf{A}) - \mathbf{A}$, where $\mathbf{A}$ is the graph's adjacency matrix and $\deg(\mathbf{A})$ is a diagonal matrix containing the degree of each node in the graph.

[3]This is equally applicable to an IIR filter. What has been shown here is a generic FIR filter.

In filter-based propagation, as with regularization, the goal is to obtain some new signal $s_i'[n]$ where each value of n has been assigned a label. This can be done as follows. Because $h\{\cdot\}$ is not translation-invariant, its filter response is different for each value of n. However, because it is linear, then it will follow that the filtered signal $f[n]$ will be

$$\begin{aligned} f[n] &= h\{s_i[n]\} \\ &= h\{L_i\delta[n-p_i]\} \\ &= L_i h\{\delta[n-p_i]\}. \end{aligned} \tag{5.7}$$

The response of the filter at p_i, the $h\{\delta[n-p_i]\}$ term, will vary depending on p_i. For the purposes of interpolation, it is desirable to remove it from the final result. Therefore a second, "confidence" signal can be defined as

$$c_i[n] = \delta[n-p_i]. \tag{5.8}$$

By inspection it can be seen that the filtered output $g[n] = h\{c_i[n]\}$ is $h\{\delta[n-p_i]\}$. Finally, the labelling can be propagated by

$$\begin{aligned} s_i'[n] &= \frac{f[n]}{g[n]} \\ &= \frac{L_i h\{\delta[n-p_i]\}}{h\{\delta[n-p_i]\}} \\ &= L_i. \end{aligned} \tag{5.9}$$

The filtered confidence signal $g[n]$ acts like a normalization term that indicates how far the particular label was propagated. Divided the filtered confidence by the filtered signal removes the effect of the original filter. As can be seen by the result in (5.9), the process propagates the label value L_i to all values of n.

This result can easily be generalized to multiple labels with varying confidence values[4]. Let

$$s[n] = \sum_i C_i L_i \delta[n-p_i] \tag{5.10}$$

and

$$c[n] = \sum_i C_i \delta[n-p_i], \tag{5.11}$$

where $C_i \in [0,1]$ and indicates the weight of the particular label. For instance, a low value of C_i will result in it contributing less to the overall normalization. Following the same derivation as in

[4]This indicates how much influence each label has. An example of how this can be applied can be found in [26].

(5.9), the multi-label result $s[n]$ is

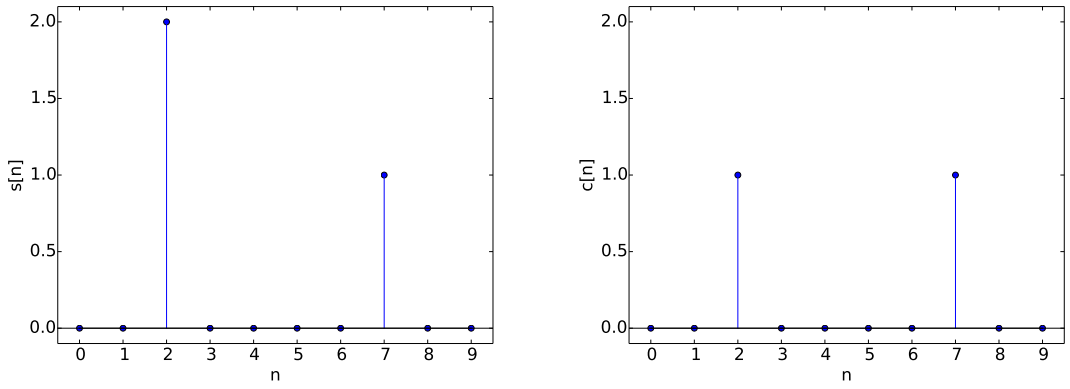
$$s'[n] = \frac{1}{\sum_i C_i h\{\delta[n-p_i]\}} \sum_i C_i L_i h\{\delta[n-p_i]\}. \tag{5.12}$$

This is possible because the filter is linear and so $h\{s[n]\}$ can be written as

$$h\left\{\sum_i C_i L_i \delta[n-p_i]\right\} = \sum_i C_i L_i h\{\delta[n-p_i]\}. \tag{5.13}$$

The same is also true for $h\{c[n]\}$. This result can be easily extended into the N-dimensional case as it only relies on the properties of linear filters.

To better illustrate the propagation process, a simple example will be provided. Figure 5.1 shows the labelling that is to be interpolated. Figure 5.1a shows the labelling signal such that $s[2] = 2$ and $s[7] = 1$. All other values are set to 0 to indicate that they are unlabelled. Figure 5.1b shows the confidence signal derived from this labelling.



(a) Labelling (b) Confidence

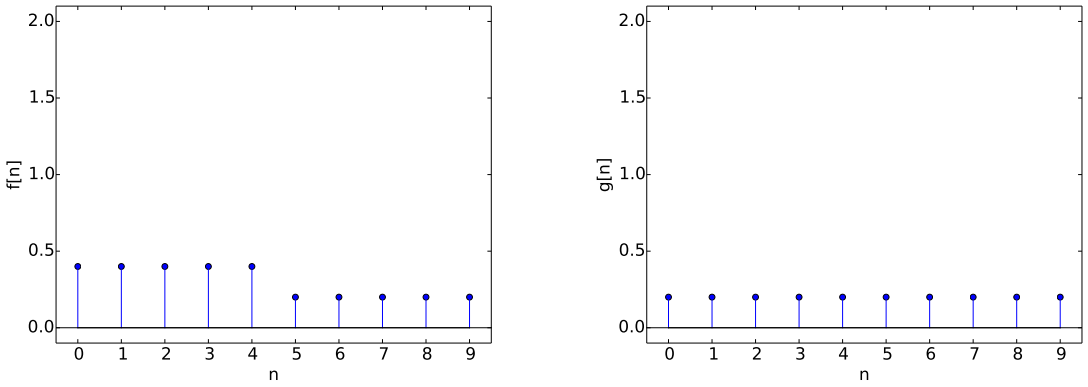
Figure 5.1: A simple 1D label signal. Both the labelling and its associated confidence signal are shown. The labelling is explicitly defined as $s[n] = 2\delta[n-2] + \delta[n-7]$.

For the choice of a propagation filter, let $h\{\cdot\}$ be a simple N-element box filter

$$h\{x[n]\} = \frac{1}{N} \sum_{i=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} x[n-i]. \tag{5.14}$$

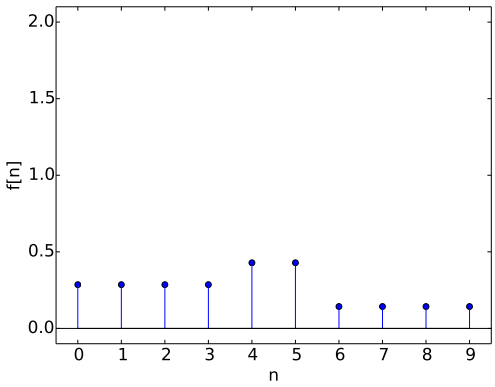
This is a true LTI filter but, as previously discussed, translation-invariance is not a necessary requirement. The results of filtering the signals in Figure 5.1 with $h\{\cdot\}$ are shown in Figure 5.2 for filter widths of $N = 5$ and $N = 7$, with the final propagated results shown in 5.3.

The filter widths were chosen so that in the $N = 5$ case there would be no overlap between

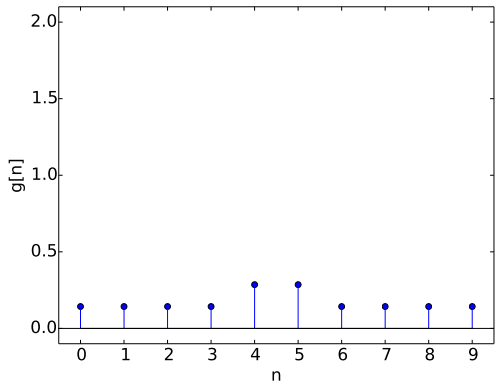


(a) Labels ($N = 5$)

(b) Confidence ($N = 5$)



(c) Labels ($N = 7$)

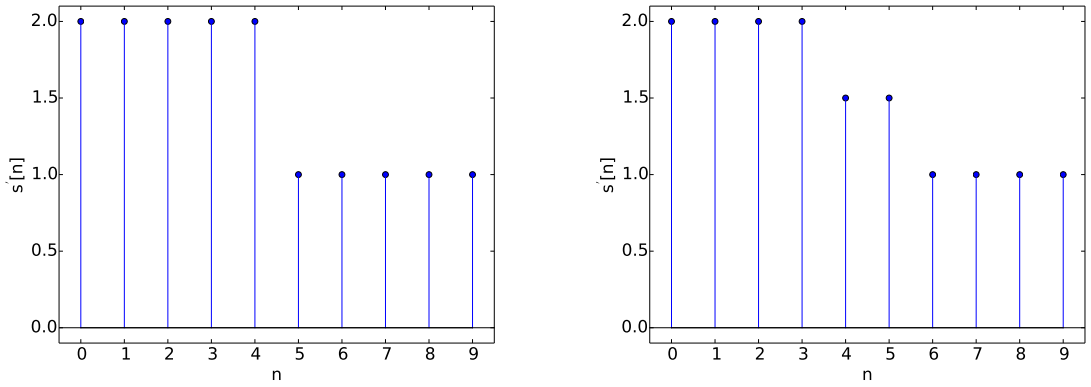


(d) Confidence ($N = 7$)

Figure 5.2: Filtered labelling and confidence signals. The top row ((a) and (b)) shows the results when filtering with a box kernel of width 5. The bottom row ((c) and (d)) shows the results for a kernel width of 7.

the filter kernels and while in the $N = 7$ case there would be. When $N = 5$, the unscaled filter responses, specifically the $h\{\delta[n - p_i]\}$ terms in (5.12), are wide enough to cover all of the values of n. However, when $N = 7$, there is a slight overlap at $n = 4$ and $n = 5$. The fact that $s'[4] = s'[5]$ is an artifact of the box filter itself; another filter could easily produce $s'[4] = 1.67$ and $s'[5] = 1.33$. The key factor is that there is a smooth transition between the two regions.

The filter used in this example has been a simple FIR box filter. As such, finding the optimal filter width requires manual tuning. In the IDW case, a distance transform is used because it ensures that the finally label boundaries are midway between any two labels (a Voronoi tessellation). If the labels are being distributed based on the content of an image, or some other signal, then an edge-aware filter can be used to automatically control the filter width. In a homogeneous region the filter kernel can be made wide while when near an edge the filter can be made more narrow. This process then emulates anisotropic diffusion.



(a) $N = 5$ (b) $N = 7$

Figure 5.3: Propagated labelling from Figure 5.1. The result in (a) shows the propagation when the filter has a width of 5 and the result in (b) is for when the filter width is 7.

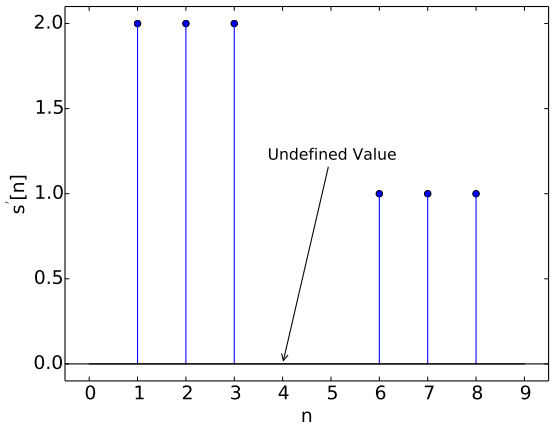


Figure 5.4: Example of the 0/0 condition in a one-dimensional signal. Not only do the labels not propagate far enough but the gaps between labels are undefined (numerically they are evaluated as `NaN` or "not a number").

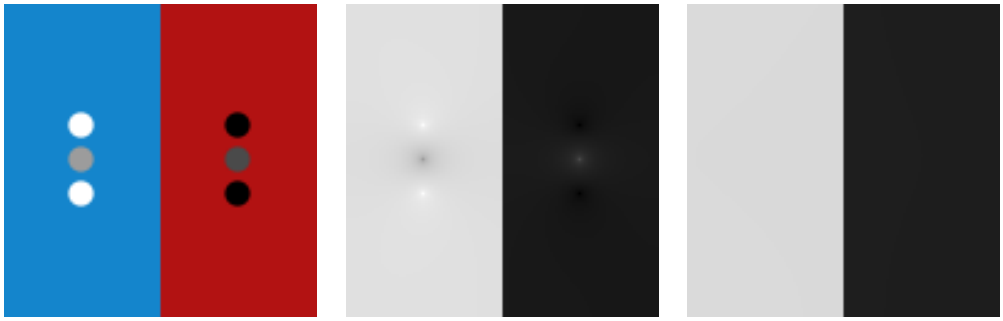
What has not been mentioned up to this point is what happens when the denominator of (5.12) becomes zero. Consider the case when $N = 3$, the results of which are shown in Figure 5.4. Because the filter was not wide enough to cover the gap between the labelling regions, there are still values in both $f[n]$ and $g[n]$ that are zero. These regions are therefore ambiguous in terms of the labelling. No mention of this issue has been made of in the literature. Either the spacing between labels was quite close [26] or an IIR-like filter [57] was used. In both cases, this zero-over-zero condition was unlikely to occur. Therefore strategies for handling this situation will be provided in Section 5.3.

5.1.3 Choice of Approach

Given that there are two classes of propagation methods, which one could be considered best suited for propagating the sparse depth labels obtained from Chapter 4? At first glance regularization would appear to be the better choice. Expressing the propagation as a regularization problem is straightforward, it provides a globally optimal solution and through the use of robust penalty functions can work well with noisy image data. And, as discussed earlier, much of the prior work has taken this approach.

However, regularization assumes that the labels are all correct. While some methods, namely [94], attempt to be robust against errors in the *position* of the labels[5], the label values themselves are always assumed to be correct. A particular problem occurs when one feature has a noticeably different depth than other nearby features; i.e. it is an outlier. It is possible to incorporate some sort of confidence measure into the regularization by adjusting the label affinity w_i in (5.3). Because user input should always be respected, user labels can always have $w_i = 1$ while the sparse depths can have their value of w_i set by the confidence term. However, finding an appropriate confidence term is non-trivial.

An alternative is to use filter-based propagation. Unlike regularization, filter-based propagation does not attempt to make the results as close as possible to the original labelling. If there is an outlier than it will be removed during the filtering process. While a confidence term can be included, it is not necessary.



(a) Image (b) Random Walks (c) Filtering

Figure 5.5: Comparison between propagation methods. The original image (a) has two homogeneous regions where the depths should be 1 and 0, respectively. The labels are one pixel in size and have been exaggerated for clarity. Two outlier labels, one with a value of 0.6 and 0.3, have been added in the regions. The regularization-based approach (b) has a noticeable artifact at the outlier while the filter-based approach (c) does not.

Figure 5.5 shows how filtering can handle outliers without any special considerations. The DT-NC filter was used to produce Figure 5.5c. More information will be given in Section 5.2.

[5]Because [94] is intended for segmentation, it is useful to ensure that slight differences in label placement do not cause large changes in the segmentation boundary.

Consider the left-hand side of the test image. Because the labelling is also the boundary conditions for the regularization (Figure 5.5b), the original labels are visible in the final result. The depth farther away from the labels becomes close to the average of the labels (approximately 0.86). However, when filtering is used (Figure 5.5c) the depths in that region is completely flat. The outlier does cause the depth value to be less than one but this can be easily fixed through intensity adjustment. Furthermore, as has been discussed, the actual depth values are not important. Rather, it is their ordering.

Therefore, for this particular application, filter-based propagation is in fact preferable to regularization. As will be shown in Chapter 6, filtering can be easily modified to enforce temporal consistency and allow for user interaction. While regularization is preferable if *all* of the labels are from the user, filtering is better when some of the labels may in fact be incorrect.

5.2 Propagation Through Edge-aware Filters

Using edge-aware filters for label propagation is straightforward. Let $EF\{A(x,y)|J(x,y)\}$ be an edge-aware filter that filters an image $A(x,y)$ using another image $J(x,y)$ to provide the edge information for the filter. This is known as joint filtering and $J(x,y)$ is referred as a "guidance image" because it guides the filter as it processes $A(x,y)$. Propagating depth labels is then done by

$$D(x,y) = \frac{\text{EF}\{C(x,y)D_s(x,y)|I_k(x,y)\}}{\text{EF}\{C(x,y)|I_k(x,y)\}}, \tag{5.15}$$

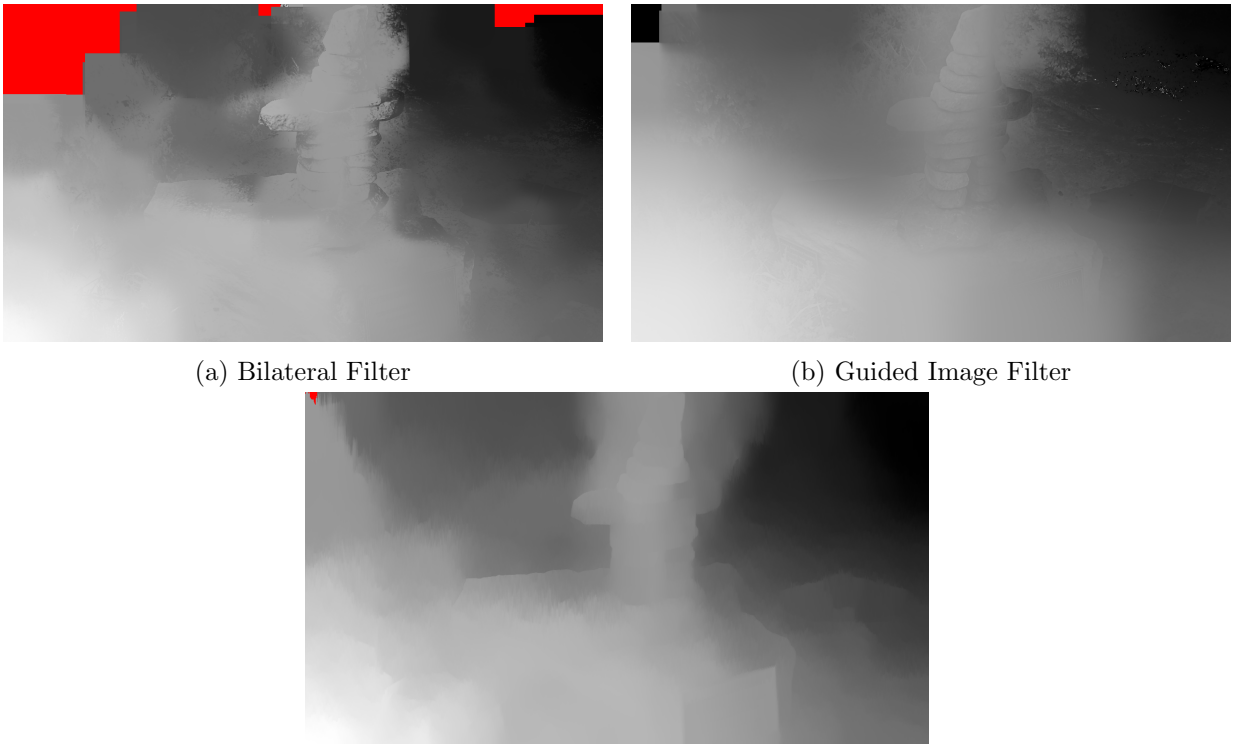
where $I_k(x,y)$ is the image for frame k. Note that this is the exact same form as (5.12) and so division and multiplication are done per-pixel (element-wise).

Figure 5.6: Labelling example image.

Figure 5.6 shows the image and sparse depth labels used for the remainder of this chapter. These are the same labels generated at the end of Chapter 4, as shown in Figure 4.11. The same "heat map" colouring has been used so that brighter labels indicate closer depths. The size of the labels have been exaggerated for visualization purposes.

5.2.1 Choice of Filter

A variety of edge-aware filters have been proposed [55, 56, 57, 58], with the venerable bilateral filter being the best known. In principle any edge-aware filter could be used for propagation, however in practice this is not the case. As was discussed in Section 5.1.2, the filter is assumed to be linear and many edge-aware filters are not. This has an important consequence when using a non-linear edge-aware filter.



(a) Bilateral Filter

(b) Guided Image Filter

(c) DT-NC

Figure 5.7: Comparisons between different edge-aware filters. The red regions indicate where (5.15) has become 0/0. The filtering parameters were: bilateral filter, $\sigma_s = 30$, $\sigma_r = 0.1$ and a filter radius of 180; Guided Image Filter (GIF), $r = 100$ and $\epsilon = 0.1$; DT-NC, $\sigma_s = 1000$ and $\sigma_r = 4$. The parameters were chosen to provide the best results for each filter.

Figure 5.7 compares three edge-aware filters: the bilateral filter [55], He et al's GIF [58] and Gastal and Oliveira's Domain Transform (DT) filter [57]. Both the bilateral filter (Figure 5.7a) and GIF (Figure 5.7b) are non-linear and so structure (edges and surface texture) in $I_k(x, y)$ appears in

the depth map. This is because both filters are *structure preserving* and so tend to transfer edges and texture from the guidance image.

Conversely, the DT filter (Figure 5.7c) is a linear, but translation-varying, filter and so does not exhibit this transferring property. This was noted, but not elaborated upon, by Lang et al in [26] when they found that the bilateral filter produced mediocre results when propagating sparse optical flow estimates. Any linear, translation-varying filter could be used, such as edge-avoiding wavelets [56]. However, the DT filter has a number of useful properties, such as converging to a Gaussian filter response in homogeneous regions, that makes it appropriate for this task.

5.2.2 Domain Transform Filter

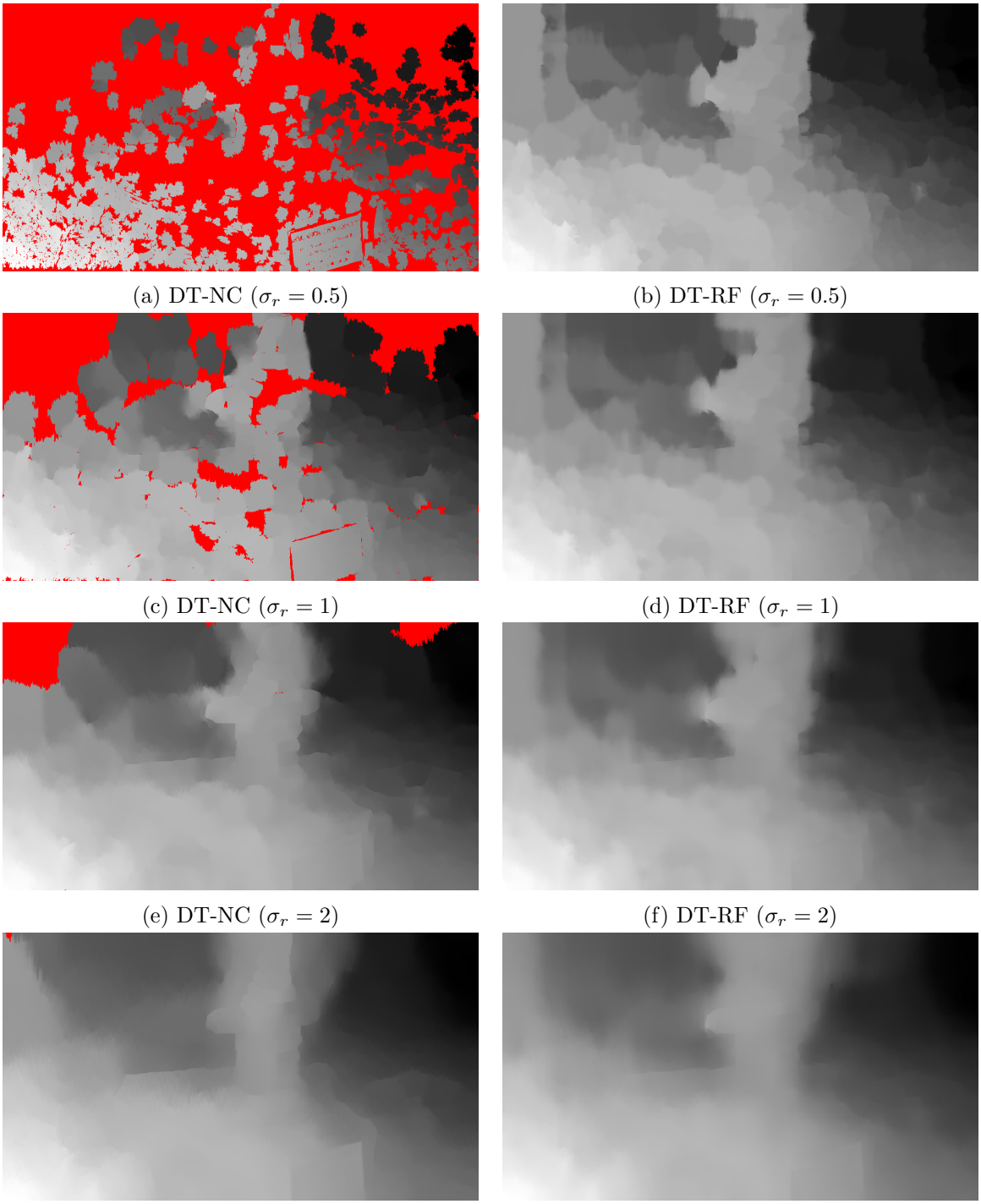
The DT filter, as proposed by Gastal and Oliveira, has three major variants, two of which will be discussed. The domain transform is designed so that similar pixels will be close together and dissimilar pixels will be far apart. The different variants arise from the different filters applied to the transformed image. It is important to note that the filters are all LTI in the transformed domain but translation-varying in the original domain. For more information about the filter and how it operates, please refer to Appendix B.

Of specific interest are the Normalized Convolution (DT-NC) and Recursive Form (DT-RF) variants. The Interpolated Convolution (DT-IC) variant is a minor modification to the DT-NC filter and produces very similar results. A comparison between the DT-NC and DT-RF is shown in Figure 5.8 for various values of σ_r. The σ_r parameter controls the filter width in the transformed domain and so larger values correspond to a greater edge-smoothing effect. The σ_s term, which has been held fixed at $\sigma_s = 1000$, controls the filter's spatial extent. For interpolation, σ_s should be set to a reasonably large value; it has little effect on the results unless it is very small[6].

A key observation between the DT-NC and DT-RF filters is that the DT-RF filter, being an IIR filter and so has infinite extent, is less prone to the 0/0 condition. However, it also exhibits a greater smoothing effect, something that is undesirable when the propagation should respect strong edge boundaries. For depth maps in particular, there should be little depth variation within objects but a strong different between objects.

The DT-NC filter better fits this criteria as it is less likely to smooth over edges. But, it is also more susceptible to 0/0's as it is a variable-width FIR filter. This can be mitigated with a large value of σ_r bit it also increases the amount of smoothing. Therefore the remainder of this chapter will primarily focus on strategies to make the DT-NC filter more robust against the 0/0 condition and allow for the use of small values of σ_r.

[6]The filter has the property that $\sigma_s \to \infty$ does not cause unbounded smoothing provided that σ_r is a finite value.



(a) DT-NC ($\sigma_r = 0.5$)

(b) DT-RF ($\sigma_r = 0.5$)

(c) DT-NC ($\sigma_r = 1$)

(d) DT-RF ($\sigma_r = 1$)

(e) DT-NC ($\sigma_r = 2$)

(f) DT-RF ($\sigma_r = 2$)

(g) DT-NC ($\sigma_r = 4$)

(h) DT-RF ($\sigma_r = 4$)

Figure 5.8: Comparison between the DT-NC and DT-RF filters for varying values of σ_r. The value of $\sigma_s = 1000$ for all examples as it controls the size of the filter kernel in the spatial domain. The red areas indicate where (5.15) becomes 0/0.

5.3 Augmented Filtering Strategies

Two separate strategies will be proposed: iterative propagation and propagation by resampling. The iterative approach applies the propagation method multiple times to propagate information into the 0/0 regions as this just another type of label propagation problem. The resampling approach is somewhat different. It assumes that the input labelling is spread too far apart and so it is necessary to bring the features closer together spatially. The two methods are not mutually exclusive and the resampling approach in fact uses the iterative propagation as a final step. In both instances the objective is to completely label each image pixel while minimizing the amount of blurring done by the filters.

5.3.1 Iterative Propagation

The premise behind the iterative propagation approach is simple: after the initial application of (5.15), apply it again on any pixels that are 0/0, using the previous results as the labelling for the next iteration. The key to this approach is generating a new confidence map, $C_{\text{int}}(x, y)$, based on the unknown pixels and then using the known pixels as the input labelling to (5.15). Specifically, the per-iteration confidence is defined as

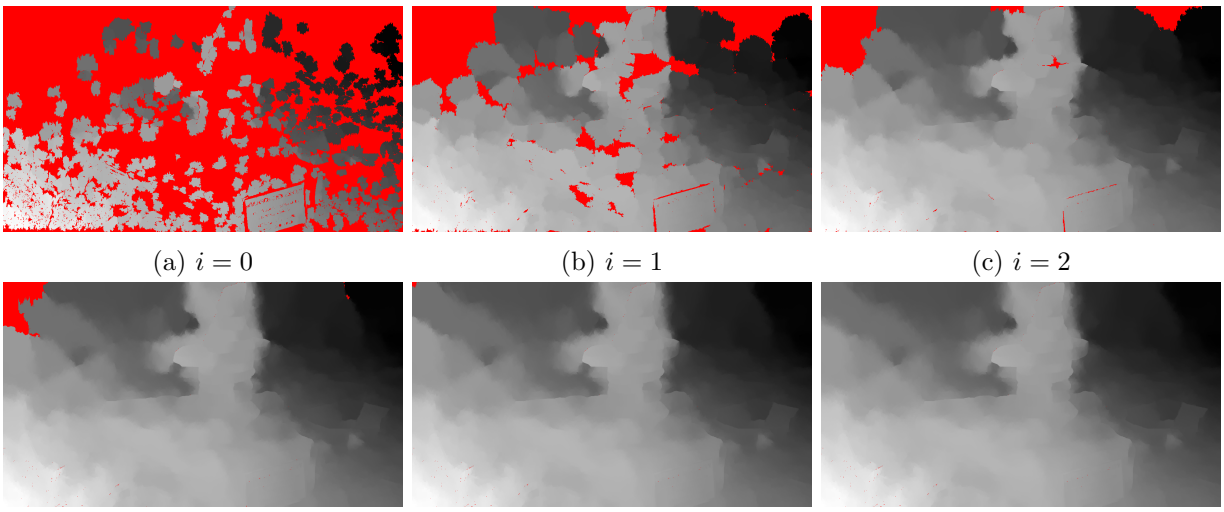
$$C_{\text{int}}(x,y) = \begin{cases} 1 & D_{\text{int}}^{(i-1)}(x,y) \text{ is not } 0/0 \\ 0 & \text{Otherwise} \end{cases}. \tag{5.16}$$

Specifically detecting the 0/0 condition is done with a system-dependent IsNaN(·) function that produces a binary image indicating if a pixel had a 0/0 value. This is because the 0/0 condition will manifest itself as a floating-point error[7]. Logically inverting the output of the function will then indicate which values in the map have been properly labelled.

The whole method can be outlined as follows:

1: $D_{\text{int}}^{(0)}(x,y) \leftarrow D(x,y)$ `// Initialize by using` (5.15) `to obtain` $D(x,y)$.
2: $i \leftarrow 0$
3: **repeat**
4: $\quad i \leftarrow i + 1$
5: $\quad C_{\text{int}}(x,y) \leftarrow \neg$ IsNaN$(D_{\text{int}}^{(i-1)}(x,y))$ `// Negate output of` IsNaN(·).
6: $\quad D_f(x,y) \leftarrow \text{DT}\{D_{\text{int}}^{(i-1)}(x,y) | I_k(x,y)\}$
7: $\quad C_f(x,y) \leftarrow \text{DT}\{C_{\text{int}}(x,y) | I_k(x,y)\}$
8: $\quad D_{\text{int}}^{(i)}(x,y) \leftarrow D_f(x,y)/C_f(x,y)$
9: **until** ZeroCount$(C_{\text{int}}(x,y)) < N_{un}$ `// Finish when enough pixels are labelled.`
10: $D_{\text{int}}(x,y) \leftarrow D_{\text{int}}^{(i)}(x,y)$

[7]Either the numerator and denominator both become vanishingly small and so the fraction approaches 0/0 or only the denominator becomes very small and so the fraction becomes "infinite".



(a) $i = 0$ (b) $i = 1$ (c) $i = 2$

(d) $i = 3$ (e) $i = 4$ (f) $i = 5$

Figure 5.9: Label propagation using the iterative approach. The subfigures show how more and more of the unlabelled regions are filled in with each iteration i. The DT-NC filter parameters used were $\sigma_r = 1000$ and $\sigma_s = 0.5$.

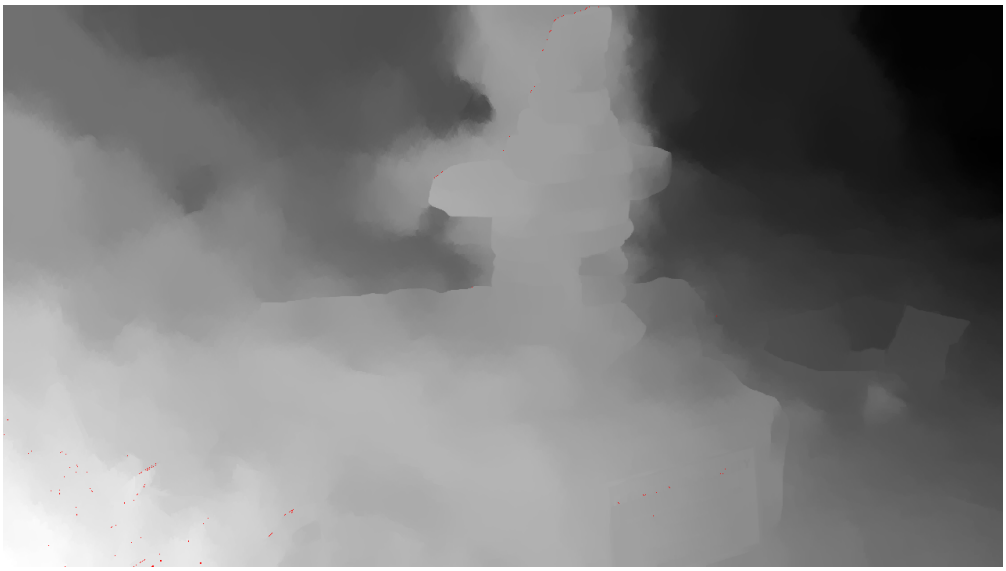
While this loop can continue as long as there are unlabelled pixels, in practice it should be terminated after a set number of iterations as there may be several pixels that can *never* filled in. However, if the number of unlabelled pixels is less than some acceptable threshold N_{un} then the loop should be terminated. It should be noted that prior to filtering any indeterminate values (i.e. $0/0$) in $D_{\text{int}}^{(i-1)}(x, y)$ are set to zero to avoid propagating[8] the floating-point errors throughout the image.

Figure 5.9 shows how the map is progressively filled-in over several iterations. Because the filtering is applied to the *entire* image, not just the unlabelled areas, it helps in smoothing out any artifacts from using very small values of σ_s. Furthermore it tends to finish covering the entire image very quickly as re-filtering the previous results allows them to diffuse into the unknown regions.

A direct comparison between the iteratively filtered result and using a very large filter kernel is shown in Figure 5.10. Not only does the iterative approach preserve more structural detail than using a large σ_r value, there are no large, unlabelled regions. While there are still a few pixels in Figure 5.10b that are unlabelled, they can be easily filled in using a median filter. However, when combined with resampling, this will be unnecessary.

[8]Operations with `NaN`s will cause the result to also have a `NaN` value. Please see Section 6.2 of the IEEE-754 floating point standard [97].

(a) Large Filter Kernel



(b) Iterative Filtering

Figure 5.10: Comparison between using a large filter kernel and the iterative approach. Please note that (a) is the same image as in Figure 5.8g. The DT-NC filter was used to produce the images.

5.3.2 Label Resampling

The second proposed approach is to propagate on a subsampled version of the image and then upsample the result (i.e. resampling). The goal is to reduce the spatial distance between features since interpolating on the downsampled version and upsampling effectively increases the size of the filter kernel. First, the features are scaled so that

$$\vec{q}_n = \frac{\vec{p}_n}{K}, \tag{5.17}$$

where $\vec{p}_n$ is the position of the feature associated with $d_k[n]$ and K is the sampling factor. If any features map to the same pixel then the median depth value is used. The downsampled image is then

$$J_k(u,v) = I_k\left(\frac{x}{K}, \frac{y}{K}\right). \tag{5.18}$$

Downsampling the image is performed using bicubic interpolation to avoid aliasing (this is purely an implementation choice, any anti-aliasing filter would work).

After downsampling, a dense depth map $D_{sub}(u,v)$ is generated from $\vec{q}_n$, with $J_k(u,v)$ as the guidance image, using (5.15) without any modifications. The downsampling brings the feature locations closer together along with removing small-scale image features such as weak edges. Upsampling is performed using a modified version of (5.15) that is defined as

$$D(x,y) = \frac{\mathrm{DT}\left\{D_{up}(Ku,Kv)|I_k(x,y)\right\}}{\mathrm{DT}\left\{C_{up}(x,y)|I_k(x,y)\right\}}, \tag{5.19}$$

where

$$D_{up}(x,y) = \begin{cases} D_{sub}(u,v) & \text{if } (x,y) = (Ku,Kv) \\ 0 & \text{if } (x,y) \neq (Ku,Kv) \text{ or } D_{sub}(u,v) \mapsto 0/0 \end{cases} \tag{5.20}$$

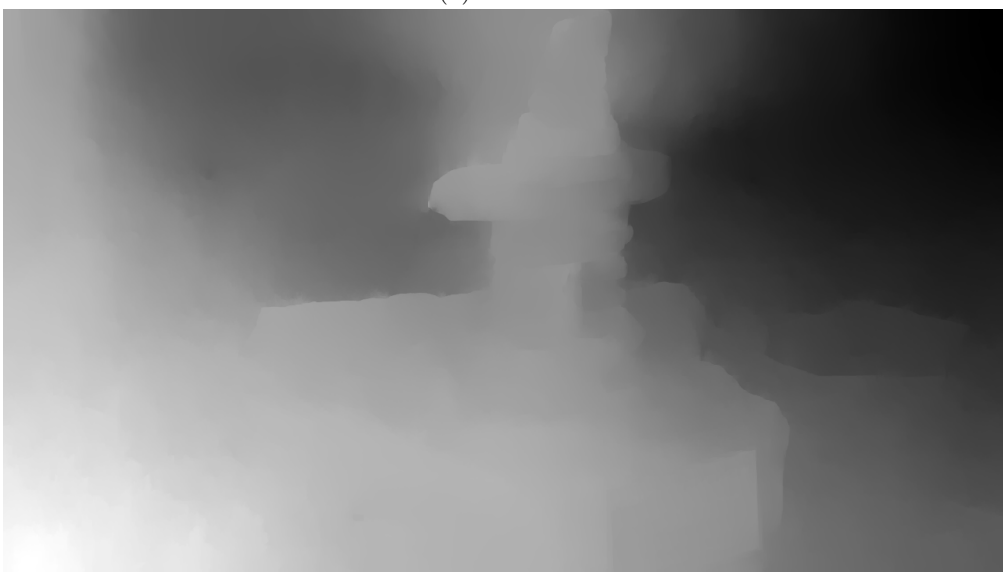
and

$$C_{up}(x,y) = \begin{cases} 1 & \text{if } (x,y) = (Ku,Kv) \\ 0 & \text{if } (x,y) \neq (Ku,Kv) \text{ or } D_{sub}(u,v) \mapsto 0/0 \end{cases}. \tag{5.21}$$

Essentially $D_{up}(x,y)$ and $C_{up}(x,y)$ define a uniform lattice where the features in the subsampled image map to the full-resolution version. Because the $0/0$ condition is *always* possible, the NaN values should not be propagated and so are ignored. As well, because the distance between known values when upsampling is constant (it is a lattice), the $0/0$ is less likely to occur. Essentially, there is a higher likelihood that the filter responses will overlap as shown in Figure 5.3b.

However, it is still possible that a $0/0$ region remains even after the upsampling is complete. This can occur when there is a large region of the image that is has no features and so no information is propagated there during the initial filtering stage. To address this, the iterative method from Section 5.3.1 can be applied to fill in these regions. Because the regions should be relatively small,

(a) DT-NC



(b) DT-RF

Figure 5.11: Comparison between the DT-NC and DT-RF filters when applying the two-stage interpolation with $K = 8$, $\sigma_r = 1$ and $\sigma_s = 1000$. The results are similar but the DT-NC filter (a) better preserves edges. These correspond to (c) and (d) in Figure 5.8.

the number of required iterations will be less than if that method was applied directly.

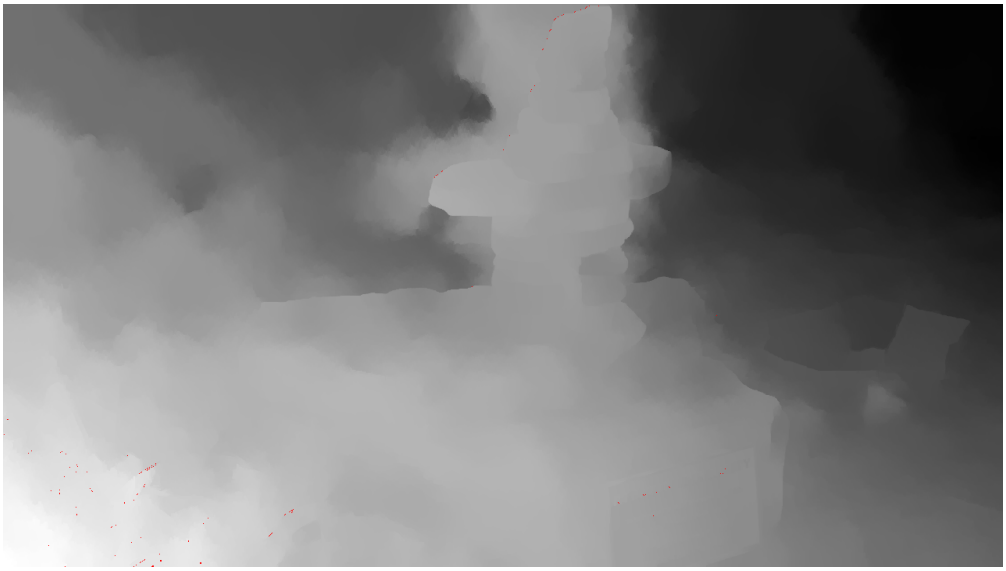
Figure 5.11 shows the interpolation method when resampling is used. The figures can be directly compared to Figures 5.8c and 5.8d as the only difference between them was the use of intermediate resampling step. Note that in both cases there is better definition in the large-scale edges, such as boundary of the inukshuk, and the objects are smoother internally. The difference is that, as mentioned, the DT-NC filter preserves edges better then the DT-RF filter.

5.3.3 Choice of Propagation Method

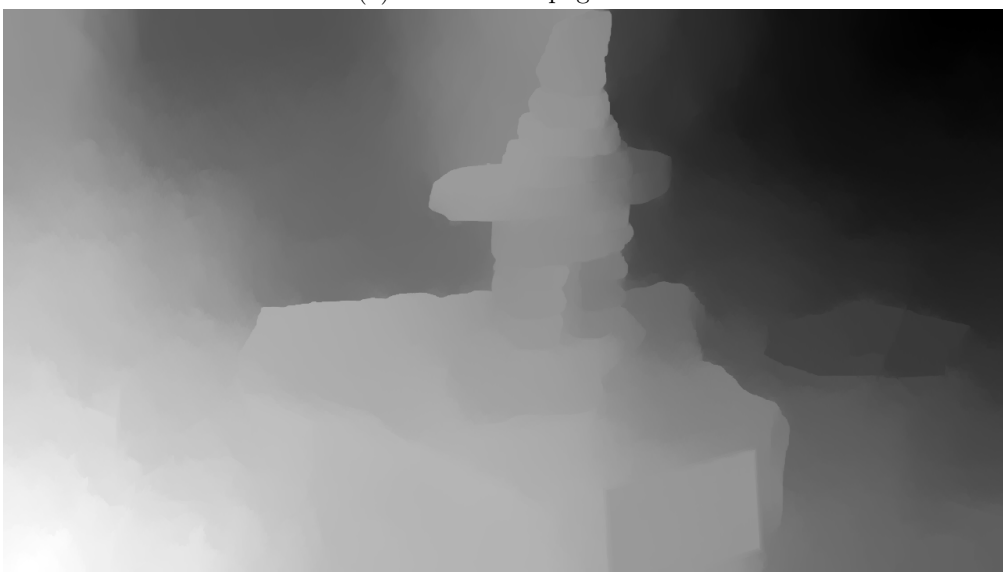
The choice between the iterative and resampling methods is not a binary one as both have their advantages and disadvantages. The iterative method is slower than resampling, taking $N_{iter} \times T$ time to generate a depth map (T is the time required to filter the image). The resampling method, however, takes constant time, $T + T/K^2$, as the DT filter processing time is linear in the number of pixels. It also produces cleaner results (object surfaces are smoother) than those from the iterative method but it has less small-scale detail.

A direct comparison between the two methods is shown in Figure 5.12. In this particular example, the resampling method does appear qualitatively better. The area around the inuksuk is clearer and the edges are more distinct. However, more detail is preserved in the ground for the iterative method.

For the remainder of this dissertation, a hybrid approach will be used as this tends to produce the best quality results. The image will first be subsampled by K and then the iterative method will be applied to obtain the map at the lower scale. The map will then be upsampled using the resampling procedure. This reduces the risk of a $0/0$ occurring at the lower scale while also not requiring such a large value of K.



(a) Iterative Propagation



(b) Propagation by Resampling

Figure 5.12: Comparison between the iterative and resampling propagation methods. The iterative example is from Figure 5.10b and the resampling example is from Figure 5.11a. While the DT-NC filter was used for both examples, the parameters are different (please see the respective figures).

Chapter 6

Post-processing

As presented, this dissertation is a purely online[1] system. A frame goes in and a depth map comes out. The system has an internal state (the tracking system from Section 4.1) but that is it. This means that it can be applied directly onto a video stream, e.g. TV broadcast, and it will continuously produce depth maps. But there are more than a few benefits for performing the processing **offline**.

Unlike online processing, offline processing, really post-processing, requires that all of the frames in the video be available. It does not necessarily have to be random access, i.e. being able to retrieve any frame at random, but it should be possible to run through the frames more than once. This type of access makes available certain types of processing that are not possible if a method is purely online. One particularly interesting ability is temporal filtering, and by extension temporal interpolation.

This chapter will cover a variety of techniques that can be used as a post-processing step for the output of either the depth estimation system (Chapter 4) or the maps produced by that system (Chapter 5). Namely it will discuss, among other things, how user-correction actually works, temporal interpolation and enforcing temporal-consistency as a post-processing step. The underlying assumption between all of these techniques is that the depth estimation has already been performed. All of the processing is then applied to those results and requires no modification as to how the depths are obtained.

6.1 User-guided Corrections and Adjustments

The output of the depth estimation front-end (Section 4.3) is as set of sparse depth estimates

$$\mathfrak{D}_k = \{d_k[n] : n \in [0, N_p)\}\,, \tag{6.1}$$

[1] Here "online" means that processing only requires access to the current frame.

where N_P is the number of features that are in frame k. Each of the individual $d_k[n]$ values are located at some position $\vec{p}_k[n]$. The values do not become a depth map until (5.1) has been applied to generate the initial sparse depth map.

Section 5.3 presented two methods for improving the quality of the initial map generation by performing multiple filtering passes. These two methods were designed to mitigate the effects of a low labelling density. However, these methods cannot add depths where none were found and will dutifully propagate those values into regions where they are inappropriate. An example of this problem is shown in Figure 6.1. Because of the poor contrast in the shadowed area, there were no features to track and so the best depth labels came from the monument and not the wall.

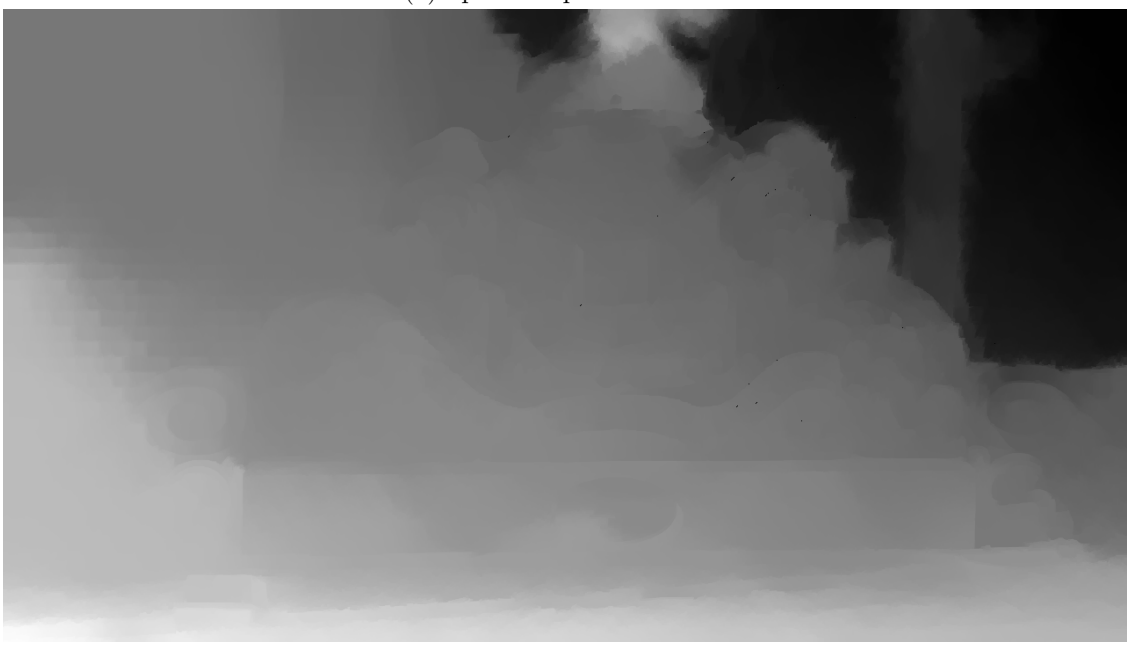
This type of error is due to two things. First, feature detectors, specifically corner-based detectors such as FAST, require regions with relatively high local contrast to detect features. Low contrast or homogeneous regions, by definition, do not have any features. This poses no problems with the depth estimation because there were enough features elsewhere to get a good estimate. This leads into the second factor: sparse interpolation. Because that region is unlabelled, depth information from nearby regions are used instead. The end result is that the left-half of the depth map is incorrect.

One way to avoid this is to instead use dense disparity methods. Instead of operating on point sets, frames along a track could be buffered, epipolar rectification applied to the images as per [30], and a multi-view method used to obtain the final disparity maps. This circumvents the lack of labels because the depth is estimated on a per-pixel basis. However, using dense methods is actually *not* a good solution in this case, as will be demonstrated in Chapter 7. Namely, if there are independently moving objects then sparse propagation produces less artifacts.

But because generating the depth maps is defined to be a label propagation problem it is straightforward to modify that labelling with corrections. Figure 6.2 shows how the depth map in Figure 6.1 can be corrected with some user-provided strokes. The resulting depth map in Figure 6.2b is now more consistent with the expected depth (the monument is perceptually "half-way" between the camera and the wall).

Choosing the appropriate depth values can be somewhat tricky. The depths have been estimated automatically and so their values may not match up with what the user expects them to be. This is really a user interface problem and is only briefly considered here. One possible solution is to provide a "depth picker" tool so that the user can select correct depths and use those elsewhere. This insures the values are both numerically and perceptually consistent without the user having to guess what the depth values should be.

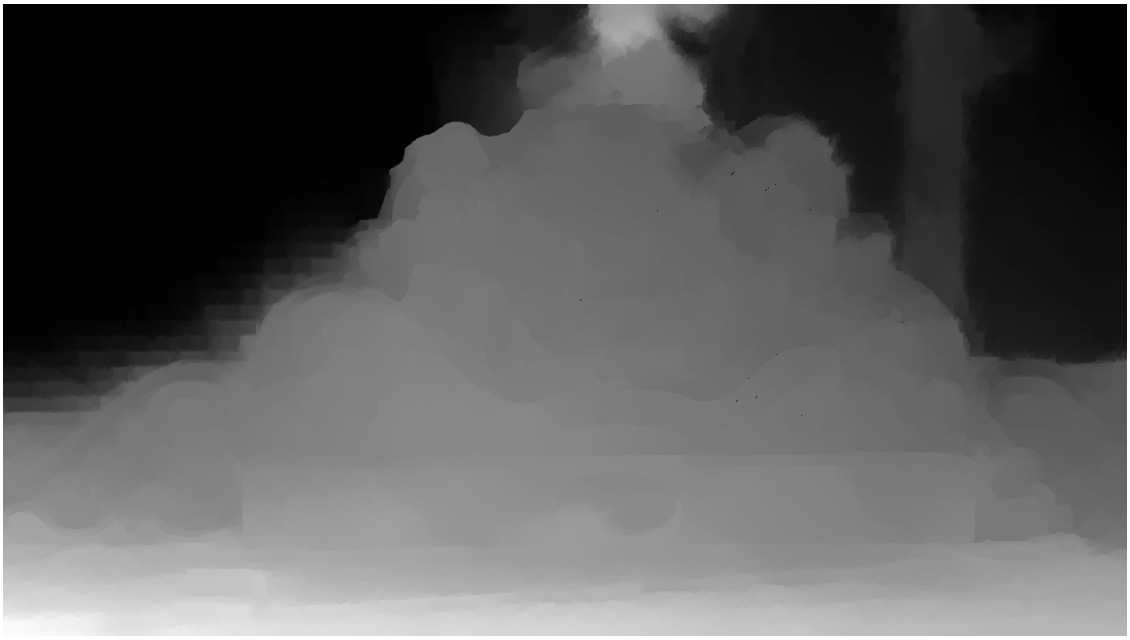
(a) Sparse Depth Estimates



(b) Generated Depth Map

Figure 6.1: Depth map obtained without any user input. Because there are no depth estimates on the left side of the image, the depth in that region is a mixture of whatever the depth was on the "boundary" of the region. The iterative method from Section 5.3.1 was used to generate the depth map in (b). Brighter colours in (a) indicate larger, i.e. closer, depth values. The sparse depths were obtained using the method of Chapter 4.

(a) User-corrected Sparse Depth



(b) Generated Depth Maps

Figure 6.2: Depth map example from Figure 6.1 with user correction. The two strokes correct the depth of the wall behind the monument and the depth on the left side of the monument itself.

6.2 Enforcing Temporal Consistency

Neither the depth estimator or map generator attempt to enforce temporal consistency. Each track is processed independently and so the depth estimates for one frame have nothing to do with those at another frame. If the camera is moving consistently (same direction and no sudden changes in velocity) then the results themselves be mostly consistent from frame to frame. This is just a consequence of footage itself. However, should the camera's motion change then it's possible for depth values to vary wildly from one frame to another. Similarly, small changes in the image content can cause the generated maps, even if the depths are correct, to appear slightly different from frame to frame. Individually the depth maps may appear to be "good" but taken together the result is a noticeable flickering.

6.2.1 Long-range Optical Flow

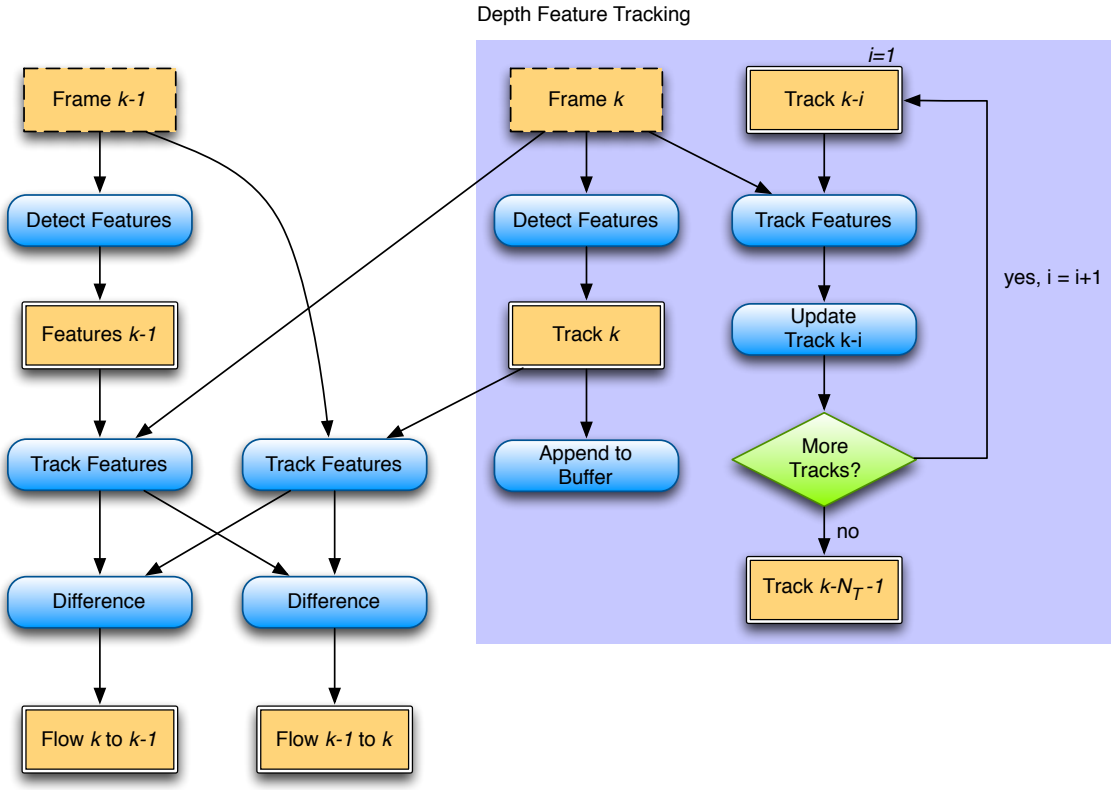


Figure 6.3: Augmented tracker for generating per-frame sparse optical flow.

There are two ways to enforce temporal consistency and they both rely on the long-range optical flow method proposed by Lang et al [26]. This method relies on sparse feature interpolation using the domain transform filter and so is a natural fit for the proposed framework. Their long-range temporal filtering method applies the DTNC filter both spatially and temporally with the

slight modification that the temporal filtering is done *along* optical flow vectors. This ensures that filter's temporal paths are consistent with how objects in the scene move instead of indiscriminately filtering all pixels equally.

Applying their method is relatively straightforward because this is the type of application that they had in mind when they developed their method. The first step is to generate the optical flow vectors for the image sequence. This can be done by augmenting the tracking system (Section 4.1) in the manner shown in Figure 6.3.

The tracker is modified so that it also buffers the previous frame, $I_{k-1}(x, y)$, and not just the feature points. Two sets of flow vectors,

$$\mathfrak{U}_k^f = \{\vec{u}_f[n] : n \in [0, N_P)\} \tag{6.2a}$$

$$\mathfrak{U}_k^r = \{\vec{u}_r[n] : n \in [0, N_P)\}, \tag{6.2b}$$

are generated by using the existing FAST+KLT tracking framework. Here $\vec{u} = (u, v)$ is an optical flow vector with u and v being the horizontal and vertical offsets, respectively.

The tracking is bidirectional and so this is why the current and previous frame need to be buffered. The sparse forward flow, mapping features in $(k-1)$ to features in k, is stored $\mathfrak{U}_k^f$. Similarly, $\mathfrak{U}_k^r$ stores the sparse reverse flow that maps $k \to (k-1)$. This occurs each time a new frame is presented and occurs in parallel with the feature tracking and depth estimation. The maps themselves are then generated using the interpolation methods presented in Chapter 5.

This approach is different from how Lang et al performed their tracking. Their approach was to extract SIFT [13] descriptors from regions surrounding features detected using a Harris corner detector[2] and then perform brute-force feature matching between frames. The matching score would then be used for the confidence map in the interpolation stage. While this is effective, nearly identical results are achieved using the much faster KLT tracker.

This setup allows optical flow estimation to be independent from the depth estimation and feature tracking. In fact, it can be enabled and disabled as needed without effecting how the depths themselves are obtained. However, because the flow sequence has been generated from sparse estimates it suffers from the same problems as the depth maps. Therefore, as suggested by Lang et al, the flow sequence itself should be filtered temporally. This ensures that the optical flow is stable over long periods of time and allows for better filtering later on.

[2]Lang et al call these "Lucas-Kanade" features but they are the same thing.

The actual filtering is an iterative process with a single iteration described as follows. As before, the superscript (i) indicates the value for the particular iteration.

1. Compute a confidence map (occlusion penalty) $O(x, y)$ from $\vec{u}_f^{(i-1)}(x, y)$ and $\vec{u}_r^{(i-1)}(x, y)$.

2. Generate a five-channel image sequence such that each frame k is

$$\vec{V}_k(x, y) = O(x, y)\left(\vec{u}_f^{(i-1)}(x, y), \vec{u}_r^{(i-1)}(x, y), O(x, y)\right).$$

   The confidence term is $O(x, y) \in [0, 1]$ and so it scales, i.e. weighs, each channel based how likely it is to be accurate. It is chosen so that occluded regions are considered to be very inaccurate and non-occluded regions as very accurate.

3. Perform a temporal filtering pass by filtering along the flow vectors in $\vec{u}_f(x, y)$.

4. Perform the horizontal and vertical spatial passes for each frame in the sequence.

5. Obtain the iteration result by

$$\vec{V}_k{}'(x, y) = \frac{\vec{V}_k(x, y)}{O(x, y)}.$$

   The values of $\vec{u}_f^{(i)}(x, y)$ and $\vec{u}_r^{(i)}(x, y)$ are extracted from $\vec{V}_k{}'(x, y)$. Because $O(x, y)$ is recalculated at the start of each iteration that channel can be discarded.

The confidence term $O(x, y)$ measures the degree of mismatch between $\vec{u}_f(x, y)$ and $\vec{u}_r(x, y)$. Theoretically, $\|\vec{u}_f(x, y) + \vec{u}_r(x, y)\|$ should be zero because the forward flow is just the reversal of the reverse flow and vice-versa. In practice, however, there will be significant disagreement where pixels are occluded by a foreground object. These pixels can then considered to be "unreliable" and so they can be assigned a very low weight in the interpolation equation (5.15). In fact, the temporal filter is just an extension of the propagation process described in Section 5.2.

Occlusion Penalty

The occlusion penalty used by Lang et al [26] is

$$O(x, y) = \left(1 - \|\vec{u}_f(x, y) + \vec{u}_r(x, y)\|\right)^{\theta}, \tag{6.3}$$

where θ controls the shape of the penalty curve (in [26] $\theta = 5$). While this does properly impose an occlusion penalty it suffers from a serious numerical problem. Namely, how do you handle the case when $\|\vec{u}_f(x, y) + \vec{u}_r(x, y)\| > 1$? Sparse estimates are not always well behaved and so the magnitude of the summed vectors in those regions can become quite large. One option is to scale the magnitudes so that

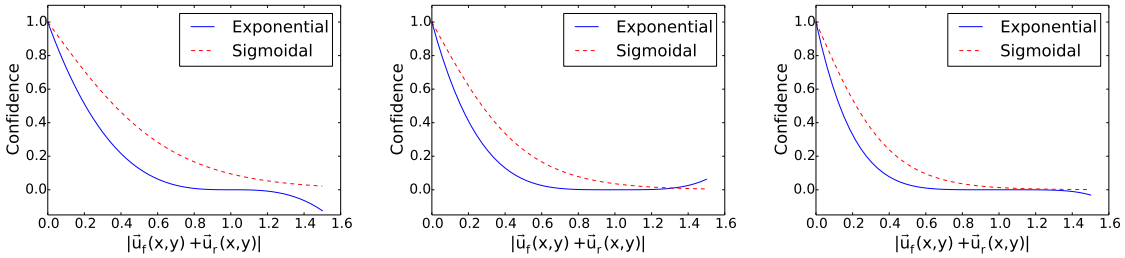
$$0 \leq \|\vec{u}_f(x, y) + \vec{u}_r(x, y)\| \leq 1 \tag{6.4}$$

though now this becomes very sensitive to outliers as not enough of a penalty will be applied to "moderate" errors. Alternatively, it can be clamped so that it never exceeds 1 and applies constant penalty after a certain point. Unfortunately this still does not address the one very serious numerical issue with this occlusion penalty: it creates a condition where the temporal filtering is now prone to the 0/0 condition described in Section 5.1.2.

The best way to address this problem is to use an alternate occlusion penalty that is more tuneable and will not experience the 0/0 condition. One such option, as was used by Phan et al [98], is to use a sigmoidal function rather than an exponential one. The specific occlusion penalty is

$$O(x,y) = \frac{2}{1+\exp\left(\theta\|\vec{u}_f(x,y)+\vec{u}_r(x,y)\|\right)}. \tag{6.5}$$

Unlike (6.3), the sigmoidal function does not have this ambiguity with what to do if the magnitude of the vector sum is greater than one. Furthermore, the value of θ can be chosen so that the magnitude will need to be very large before $c(x,y)$ is even close to zero. This effectively mitigates many of the numerical issues. A comparison between the two confidence measures is shown in Figure 6.4.



(a) $\theta = 3$ (b) $\theta = 4$ (c) $\theta = 5$

Figure 6.4: Comparison between exponential and sigmoidal confidence terms. The sigmoidal curve (dashed line) decays more slowly than the exponential term (solid line) but always remains positive.

In summary, the sigmoidal penalty has a number of desirable properties. First and foremost, it is a monotonically decreasing function. This is unlike the polynomial cost function, which will be non-monotonic if θ is even. Second, the sharpness of the curve can be better controlled. Finally, it will always be a positive value, which is possible for the polynomial cost if θ is odd.

Path-based Filtering

The rules used for the path-based filtering are the same as the ones used by Lang et al. The following summary is provided for completeness, but for more information the reader is asked to refer to [26]. The process finds a space-time path through the sequence using the forward optical flow.

All paths are initialized so that each pixel in the first frame $k = 0$ is the start of a path $\mathfrak{p}_i = \{\vec{p}[0]\}$, where $\vec{p}[0] = (x, y)$ is the 2D coordinate of the i-th pixel in the image using row-major indexing. The path position in frame $k + 1$ is found according to the rule

$$\vec{p}[k+1] = \vec{p}[k] + \vec{u}_f\left(\vec{p}[k]\right), \tag{6.6}$$

where $\vec{u}_f(\cdot)$ is the optical flow vector field that maps frame $k \to k + 1$.

The value of $\vec{p}[k + 1]$ is not immediately appended to $\mathfrak{p}_i$, though. Instead it is necessary to verify that the path can in fact be propagated to that position. The process for determining this is as follows:

1. If $\vec{p}[k + 1]$ is outside of the frame boundaries then $\mathfrak{p}_i$ will be considered as "terminated" and no longer updated.

2. If $\vec{p}[k + 1]$ had no another pixel from k mapped to it then it is appended to $\mathfrak{p}_i$ and extends the path into frame $k + 1$.

3. If another pixel in k had *already* been mapped to that position in $k + 1$ then the path is terminated and no further mapping will occur for that path.

4. If the pixel position $\vec{p}[k] = (x, y)$ did not have any points from $k - 1$ mapped to it (previous steps) then a new path $\mathfrak{p}_j = \{\vec{p}[k]\}$ is started for that pixel.

Rules 1 through 3 are straightforward: only propagate the path if the new position is inside of the frame boundaries and that the particular pixel is not already on an existing path. Handling the case of unmapped pixels (rule 4) is more complicated as there is valid image information in the preceding frames. It is important that the filtering utilizes this information so for any path that does not start at the first frame the path is tracked backwards through the sequence. Here the reverse flow is quite useful because it provides this mapping. These new positions, found by $\vec{p}[k - 1] = \vec{p}[k] + \vec{u}_r(\vec{p}[k])$, are *prepended* to $\mathfrak{p}_i$. Therefore, any path that does not start at $k = 0$ will be

$$\mathfrak{p}_i = \{\ldots, \vec{p}[k-2], \vec{p}[k-1], \vec{p}[k]\} \tag{6.7}$$

before the paths are propagated to the next frame.

The purpose of the backtracking is that it allows the filter to be "primed", i.e. the filter is setup correctly within the transformed domain, prior to actually filtering the pixel where path actually begins. While Lang et al do not specify how far to backtrack, they instead keep tracking until the filter is the proper width in the transformed domain, in practice limiting the backtracking to ten frames appears to be sufficient. The fact is that while this may cause the filter to be too narrow, it is directly related to how much variation there is along the path. If there is a lot of motion then the width will be relatively narrow and conversely it will be very wide if there is not. Finally, because

the filtering is iterative, and the paths change upon each iteration, the effect of the multiple filter passes will dwarf the small effect caused by the truncated backtracking.

An example of how the temporal filtering can improve and stabilize the optical flow is shown in Figure 6.5. The frame-to-frame optical flow can be quite erratic and it is due to a variety of different factors; in this case the footage was taken with a hand held camera and is quite unsteady. However, after filtering, the flow becomes much more consistent over the course of the sequence. Please keep in mind that this is somewhat hard to demonstrate with static images and so the unfiltered flow may appear better in this particular instance. Furthermore, this flow is used for the processing done in the next section.

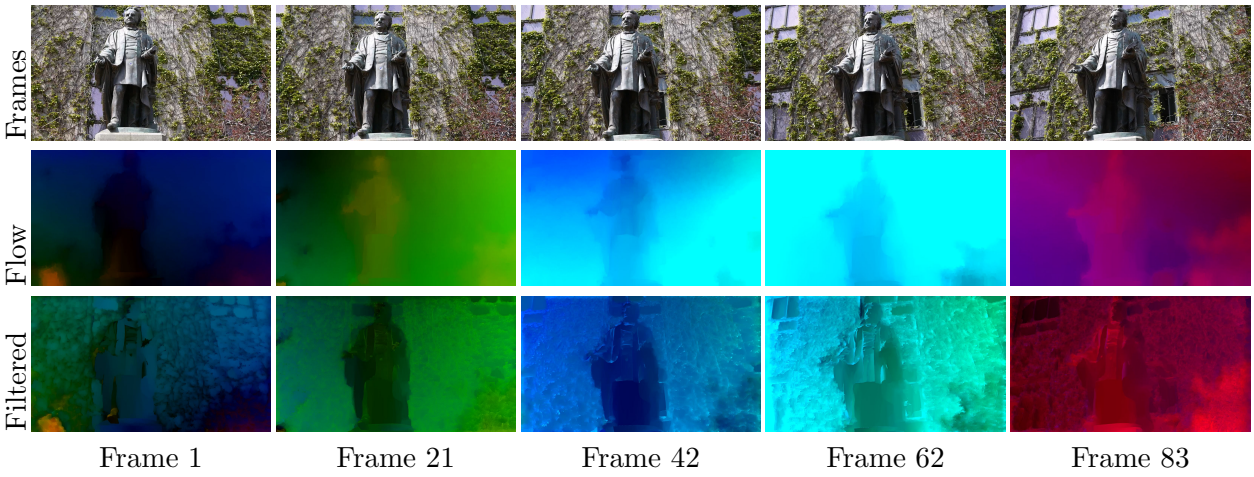


Figure 6.5: An example of temporally filtered optical flow. The top contains the original frames, the middle row contains the unfiltered forward flow and the bottom row contains filtered forward flow. The optical flow is visualized using an HSV colour scheme. The hue represents the vector's angle and the value (brightness) represents its magnitude with dark being equal to zero magnitude.

6.2.2 Temporal Feature Propagation

Temporal consistency is enforced by making the propagation itself temporal. This is simple to do for both the iterative (Section 5.3.1) and resampling (Section 5.3.2) methods. In fact, no changes whatsoever have to be made for the iterative method; just use a "temporally-aware" version of the DT-NC filter instead of the standard spatial version. For the resampling method, the only consideration that has to be made is to ensure that the resampling is only performed along the spatial dimensions. The temporal dimension should remain untouched. This avoids any ambiguities that could arise from downsampling and upsampling in time. Furthermore, the temporal filtering propagates information very effectively between frames and so is able to fill in missing depth values from frames with those values.

In both cases, the temporal filtering is quite simple to do. First, apply the DT-NC filter along

the temporal direction by following paths obtained from the filtered optical flow (Section 6.2.1). After that, perform the normal horizontal and vertical spatial passes. Unlike the optical flow filtering, the paths will never change and so they will not have to be recomputed each filtering iteration. This makes the actual filtering processing very fast and very easy to parallelize.

An example of the temporal propagation's ability to fill in missing information is shown in Figure 6.6. With online-only processing, the first several frames of a sequence have no depth information. This also occurs when there is insufficient motion and the model verification step (Section 4.2.3) fails. This particular example was generated using the resampling-based propagation method for both the online and temporal results.

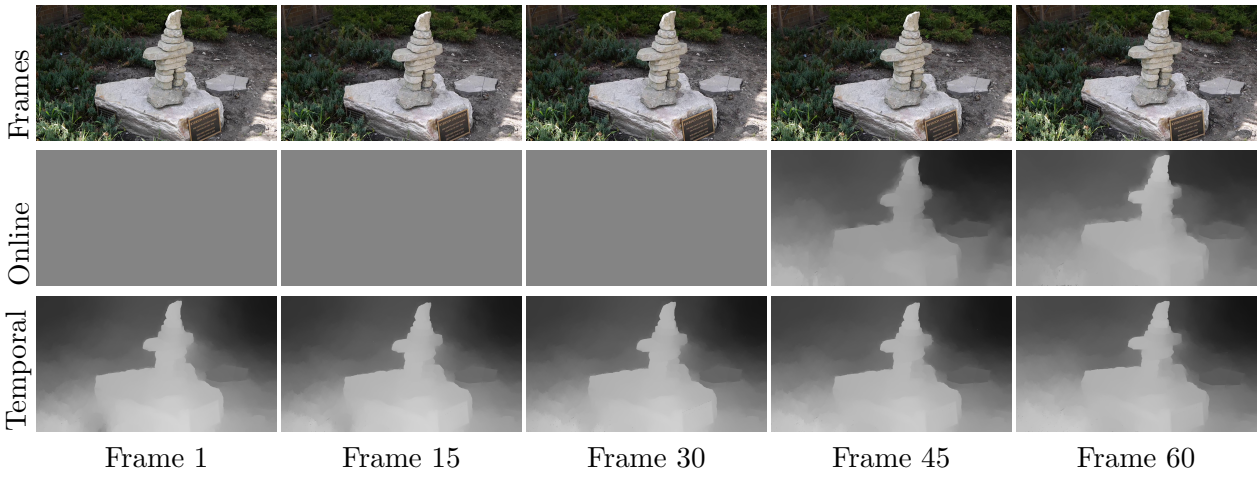


Figure 6.6: An example of temporal propagation for "Inuksuk". While no depth information is available for first dozen frames, temporal propagation is able to fill in the missing frames because it can propagate them using the optical flow sequence.

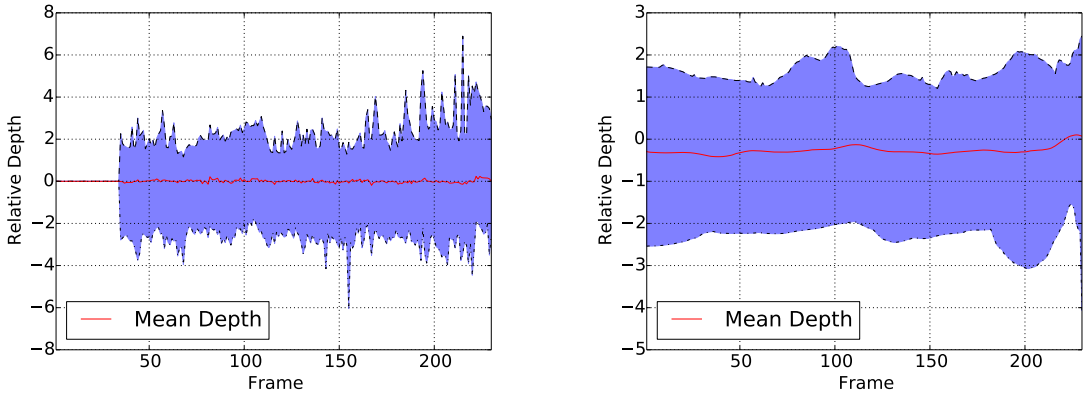
One final thing to mention: it is possible to also enforce temporal consistency on a set of *dense* maps. This is just a matter of applying the temporal DT-NC filter onto the dense sequence using the original frames as the guidance sequence. However, this does not possess the ability to propagate information; it can only smooth out differences between frames. It is analogous to joint filtering in images. Because post-filtering an existing depth map sequence requires *extra* work (the maps have to be generated in the first place), this will not be considered as temporal propagation is more flexible.

6.3 Depth Range Scaling

On their own, the obtained depth values are not particularly useful. They are unitless values based on the statistical distribution of the original disparity estimates. However, as they represent relative depth, it is convenient to rescale them so $D(x, y) \in [0, 1]$ where zero is considered far and one is

considered close. It is first useful to consider how the depth values are distributed.

In Section 4.3, the final depth estimates were standardized prior to aggregation. However, maximum and minimum values can vary quite a bit. In fact, the values are scaled relative to the magnitude of the measured disparity. Essentially, features that had very large disparities (i.e. they were close to the camera) would have large depth values relative to other frames. However, if the scene depth stayed the same over the course of the sequence then the range will stay relatively consistent. This is demonstrated in Figure 6.7



(a) Sparse Estimates

(b) Post-interpolation

Figure 6.7: Depth ranges pre and post-propagation. Temporal propagation was used to illustrate how the propagation process can effect the depth ranges.

For the "Inuksuk" footage, the camera stays at roughly the same distance from the inuksuk. So, overall, the maximum and minimum depth values stay more or less the same (Figure 6.7a). However, because each frame is handled independently, there is *some* variation with small outliers causing noticeable "spikes" in the plot. Temporal propagation (Figure 6.7b) is able to correct this, removing the large differences so that depth varies smoothly over time. It also keeps the depth fixed for those first thirty frames where the camera is stationary and the average depth, while not zero anymore, is still constant[3] over the course of the sequence.

In general, the scaling can be done by

$$D_{sc}(x,y) = \text{clamp}\left(\frac{D(x,y) - r_{min}}{r_{max} - r_{min}}\right), \tag{6.8}$$

[3]The small up-tick in the average depth at the end of the sequence, which occurs as the camera is moving *away* and so the mean should be *decreasing*, is caused by the depths becoming inverted. This will be discussed in more detail in Chapter 7.

where r_{max} and r_{min} control the scaling and clamp$(\cdot)$ is

$$\text{clamp}(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x \leq 1 \\ 1 & x > 1 \end{cases} . \tag{6.9}$$

This just ensures that the scaled values are in $[0, 1]$. The choice of r_{max} and r_{min} controls how the sequence is scaled. For online processing, the only possible choice is $r_{max} = \max\big(D(x, y)\big)$ and $r_{min} = \min\big(D(x, y)\big)$. This causes the nearest object to have a depth of '1' and the farthest to have a depth of '0'. But because this is per-frame, it can cause flickering because the average depth (brightness) depends on the range of depth values. That means that over the course of a sequence the relative position of objects *over time* is not respected.

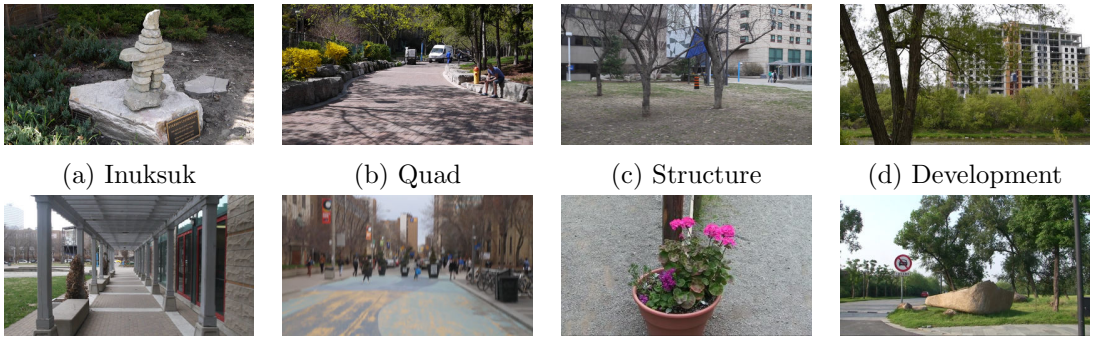
Alternatively, and this is only possible with offline processing, is to let r_{min} and r_{max} be the minimum and maximum depths over the course of the *entire* sequence. This shifts the average brightness (depth) of all frames by the same amount and so the depth of objects between frames is respected. The effect is that the closest *scene* object has a depth of '1' and the farthest a depth of '0'. Sometimes this may not be desirable because the depths are too noisy, such as if online interpolation will be used. Here, r_{min} and r_{max} can be set to something that will ignore outliers. For example, by visually inspecting Figure 6.7b, it appears that $r_{min} = -3$ and $r_{max} = 3$ would be good scaling parameters. This can be confirmed by inspecting Figure 6.7b where $r_{min} \approx -3$ and $r_{max} \approx 2.5$.

Chapter 7

Results and Discussion

The proposed framework is intended to act as a drop-in replacement for SfM in 2D-to-3D conversion methods that use SfM to obtain their initial depth estimates, such as Depth Director [76]. On its own it is *not* a 2D-to-3D conversion method. It accepts in video and produces depth maps. Therefore it is necessary to see that the framework is able to produce similar depth maps to those obtained using more sophisticated techniques (SfM/MVS).

7.1 Test Sets

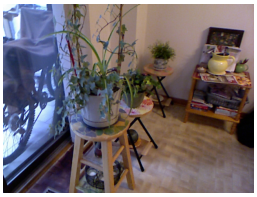


(a) Inuksuk (b) Quad (c) Structure (d) Development

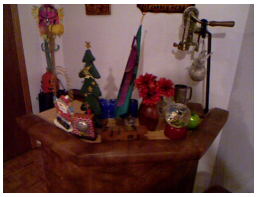
(e) Residence (f) Street (g) Flowers (h) Road

Figure 7.1: Test sequences used for the comparison between the proposed method and ACTS.

Figure 7.1 shows frames from the set of primary test samples. With the exception of "Flowers" and "Road", which are from [71], these samples were all captured with a consumer-grade DSLR camera at 24 frames-per-second. The resolution ranges from 480p SD to 720p HD, with the sequence length and resolution presented in Table 7.1. These samples were all compared against the depth recovery method of Zhang et al [71]. Visual results are provided at the end of this chapter.



(a) Plant (Kinect)



(b) Bar (Kinect)

Figure 7.2: Test sequences captured using a Kinect depth camera.

Also provided is a secondary test set, the samples having been obtained using a first-generation Microsoft Kinect depth camera[1]. Figure 7.2 shows a frame from each of the secondary samples. This allows for a "closer to ground truth" evaluation than against [71]. However, there are some limitations to the Kinect device that will be discuss in Section 7.2.1 in order to better understand those results.

Table 7.1: Resolution and duration of all of the test sequences. The "Flowers" and "Road" sequences are from [71]. Result samples are available at the end of this chapter.

Sequence	Resolution	Frames	Figure
Inuksuk	1280×720	230	7.10
Quad	1280×720	315	7.11
Structure	1280×720	368	7.12
Development	852×480	200	7.13
Residence	852×480	215	7.14
Street	852×480	240	7.15
Flowers	960×540	101	7.16
Road	960×540	120	7.17
Plant (Kinect)	640×480	81	7.18
Bar (Kinect)	640×480	36	7.19

All of the primary sequences were processed by the proposed framework and by the ACTS software package[2]. ACTS implements Zhang et al's SfM [51] and depth recovery [71] methods and so makes it a useful tool for benchmarking the performance of the proposed framework. Specifically, it makes it possible to see how close the depth estimates produced by the front-end (Chapter 4) are to those obtained through SfM. There will be some crucial differences as ACTS uses dense disparity estimation to generate depth maps while the back-end (Chapter 5) uses sparse propagation. The secondary Kinect sequences were only processed using the temporal version of the proposed method.

Processing parameters for the different sequences are given in Table 7.2. Both online and temporal propagation were used in order to provide a comparison between the two approaches. For

[1]This is officially known as the "XBox 360 Kinect".

[2]Available at `www.zjucvg.net/acts/acts.html`; accessed May 12, 2014.

Table 7.2: Processing settings used for the test sequences in Table 7.1. The number of DT filter iterations is $N = 3$ and the maximum number of iterations for iterative propagation (Section 5.3.1) is $N_{iter} = 10$. The σ_{st} and σ_{rt} control the DT filter during temporal filtering.

Sequence	σ_s	σ_r	σ_{st}	σ_{rt}	θ	K	K_T
Inuksuk	1000	0.7	8	0.9	3	16	4
Quad	1000	0.7	8	0.9	3	16	4
Structure	1000	0.7	8	0.9	3	16	4
Development	500	0.9	8	0.9	3	4	2
Residence	500	0.9	8	0.9	3	4	2
Street	500	0.9	8	0.9	3	4	2
Flowers	800	0.9	8	0.9	3	4	2
Road	800	0.9	8	0.9	3	4	2
Plant (Kinect)	500	0.7	5	0.9	3	–	1
Bar (Kinect)	500	0.7	5	0.9	3	–	1

online propagation, the map was generated as soon as an estimate was available while the estimates were stored and propagation performed after a first pass through the footage. In both cases, the propagation method was done as mixture of the iterative and resampling approaches:

1. Downsample the frame and depth estimates by a factor K (or the sequence by a factor K_T).
2. Apply iterative propagation at the lower scale.
3. Upsample to obtain depth maps back at the original resolution.

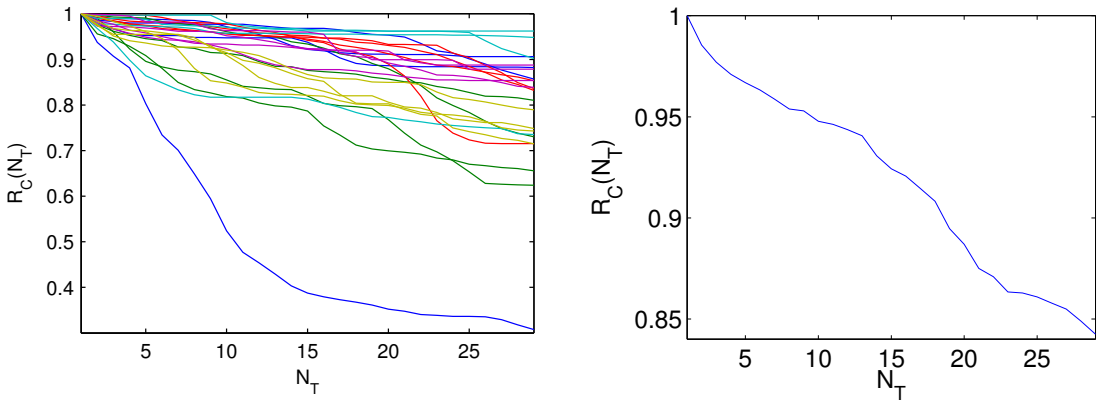
Different values of K were used for temporal propagation because it is *far* more resilient to missing data. In fact, it was found that it performed *better* when $\mathbf{F}_{k,l}$-outlier rejection was turned on. The online method required that outlier rejection be disabled otherwise the loss of so many features tended to produce maps of very poor quality.

7.1.1 Buffering Track Length Selection

The main parameter that controls the tracking scheme is N_T. While the KLT tracker and the corner detector have their own parameters, for the most part they can be left alone. This is mainly a consequence of the default parameters used in off-the-shelf implementations and the literature being appropriate for most cases. It is also often the case that there is not a simple relationship between the obtained results and the parameters, a situation very similar to hyperparameter tuning in machine learning [99, 100].

In the ideal case, $N_T = 2$ since two frames should be sufficient enough to establish a proper baseline. In practice, however, unsteady camera handling, insufficient motion and other factors make this unlikely to produce good results. One option would be to set N_T to be very large to

ensure a sufficient baseline but this in fact creates its own problems. First, it would make the buffering period extremely long. Second, the number of features still remaining at the end of the track may be very low, negating the benefit of the long duration tracking. Therefore there is a trade-off between these two factors when choosing N_T.



(a) Individual Feature Retention

(b) Median Retention

Figure 7.3: The relative feature count $R_C(N_T)$ with respect to track length. Lines with the same colour come from the same test sequence. $R_C(N_T)$ decreases monotonically with N_T and the median rate is approximately linear.

To find an "optimal" N_T, it is useful to see how features are lost over time, i.e. the feature retention rate. Figure 7.3 shows how *relative* number of features decreases as the track length N_T increases. These figures show the ratio of features after i frames to the number of features that were originally detected, i.e.

$$R_C(i) = \frac{N_P(i)}{N_P(0)}. \tag{7.1}$$

$N_P(i)$ is simply the number of features still being tracked after i frames. Figure 7.3a was obtained by starting the tracking at four, equally-space positions from the different test footage (same colour means the tracks came from the same footage). Figure 7.3b shows the *median* count of the different plots in Figure 7.3a.

There are two key observations: how fast features are lost varies wildly, even within the same video, and the overall loss rate is linear. Ideally $N_T = 2$ but because nothing is known about the footage, this will unlikely be long enough to provide enough of a baseline for the depth estimation. A good trade-off will be an N_T that tracks long enough for a good baseline while not losing too many features. From Figure 7.3, $N_T = 10$ is a good choice because in most cases only about 5% of the features will be lost while in the worst case half will still be available. It also corresponds to an initial buffering period of about one-third of a second for 24 fps footage.

It should be stated that this choice of N_T is best-suited for footage similar to the test footage in this chapter. For slow-moving cameras, this value may be too small and so the tracking length

would be insufficient. The choice of N_T will depend on the footage itself but given the intuitive nature of the parameter, it is relatively easy to pick an appropriate value.

7.1.2 Filtering Parameters

Unlike the feature tracking, the propagation stage is fairly robust against changes to the filtering parameters. That is, the filter parameters can be changed by a large degree without producing noticeable changes in the output. This has already been addressed in Chapter 5. Of the different parameters, only σ_r, σ_{rt} and K will produce any noticeable effects. The parameters in Table 7.2 were determined by manual tuning. The relationship between these parameters is described below.

The domain transform that the DT filter receives its name from is derived from the L_1 norm between RGB colour vectors. The range parameter σ_r controls the total amount of edge smoothing. When $\sigma_r < 1$ the filter shows better edge discrimination while when $\sigma_r > 1$ it tends to blur edges. Similarly, σ_{rt} controls the temporal smoothing so that when $\sigma_{rt} \ll 1$ there will be almost no temporal smoothing and vice versa. And, as discussed previously, the spatial term σ_s has little effect provided that $\sigma_s \gg 1$. Generally speaking, σ_r was chosen so that strong edges were not blurred and σ_{rt} chosen so that $\sigma_{rt} \approx 1$, with slight adjustments to avoid too much temporal smoothing. For both parameters, so long as they are not too small or too large, they can be set to any value that provides an adequate result (this will depend on the application).

The value of K has a much greater effect than σ_r and σ_{rt}. Because this is the amount by which the image is downsampled, it cannot be so large that the image is reduced to a handful of pixels. This in turn means that an appropriate value of K depends on the resolution. That is why the lower-resolution sequences use a smaller value of K. If temporal filtering is used then K can also be made small as the temporal filter can propagate information over multiple frames. Without temporal filtering, a larger value needs to be used as the 0/0 condition becomes more likely (less information) but at the expense of image detail.

7.2 Discussion

Evaluating the results will be done using both objective and subjective methods. Because no ground truth depth information exists for these sequences, it is not possible to say that the maps being produced are completely accurate and in fact it has been stated multiple times in this dissertation that this is not the case. What these different evaluations will do is determine how effective the proposed method is at depth recovery and how effective it is at being able to handle difficult situations, such as multiple independently moving objects.

Table 7.3: Mean, absolute cross-correlation scores for each test sequence. The standard deviation of the absolute cross-correlation scores is shown inside the brackets.

Sequence	Online	Temporal
Inuksuk	0.75 (0.32)	0.87 (0.16)
Quad	0.82 (0.16)	0.89 (0.02)
Structure	0.60 (0.20)	0.68 (0.16)
Development	0.70 (0.17)	0.78 (0.13)
Residence	0.70 (0.27)	0.78 (0.12)
Street	0.56 (0.21)	0.69 (0.12)
Flowers	0.85 (0.03)	0.94 (0.02)
Road	0.84 (0.24)	0.92 (0.02)
Plant (Kinect)	–	0.78 (0.03)
Bar (Kinect)	–	0.56 (0.02)

7.2.1 Correlation Scores

The mean, absolute cross-correlation between the ACTS/ground truth depth maps was used to validate that the proposed method is in fact obtaining realistic relative depth estimates. This strategy has been used before [67] because a measure like PSNR or RMSE will not adequately describe the similarity between the two depth maps. One is not the noisy version of the other but rather they are two different images representing the same underlying data: scene depth. The sign of the correlation is ignored as it implies that the signal is simply the negative[3] of the other. Because of the depth-reversal phenomenon (Section 7.2.2), taking the absolute value allows a "reversed" depth map to be compared with one that is not.

Comparing against ACTS does not necessarily indicate that the depth information is representative of the real world. Therefore, it is also instructive to compare the Kinect depth maps to those produced by the temporal version of the proposed method. The data has to first be processed because the depth-sensing camera is physically offset from the RGB camera and so must be post-processed so that the RGB and depth images are properly aligned [101]. The Kinect flags unknown depths and all unknown pixels in the Kinect map are ignored while computing the correlation. This is necessary as the unknown pixels cannot be compared to anything. While the Kinect depth data can be post-processed even further [26], this can potentially add another source of error.

Table 7.3 shows the correlation scores for the different samples. Both the online and temporal versions were used when comparing against ACTS. Ideally all matches should be close to 1 but due to the inherent differences in sparse propagation versus dense estimation, this is not the case. However, the following can be said from these results. First, the output of the proposed method strongly correlates with the output of ACTS. Therefore the depth being obtained is very similar

[3]Given two signals $x(t)$ and $y(t)$, a correlation close to 1 implies that $x(t) \approx y(t)$ while a correlation close to -1 implies that $x(t) \approx -y(t)$.

to what an SfM-based method would obtain. Second, temporal propagation greatly improves the matching score because it reduces the variations between frames, something that can be seen from the scores' standard deviations. The results from the Kinect data back up the assertion that the depth is in fact representative of the actual scene depth. The low matching score for the "Bar" sample is due to the lack of good features for tracking, particularly on the lower portion of the frame.

7.2.2 Depth Reversal

Depth reversal occurs when the camera moved opposite to how it is implicitly assumed to be moving: forwards or to the right. An example of this can be seen in the "Road" sample in Figure 7.17 where the camera moves to the left. Fixing this is trivial for the user and produces the expected results, as shown in Figure 7.4. This can become problematic for purely automated methods as it can negatively impact the temporal propagation. If the camera reverses direction through the scene then even though the actual depths do not change, the reversal will cause the overall depth to go towards zero.

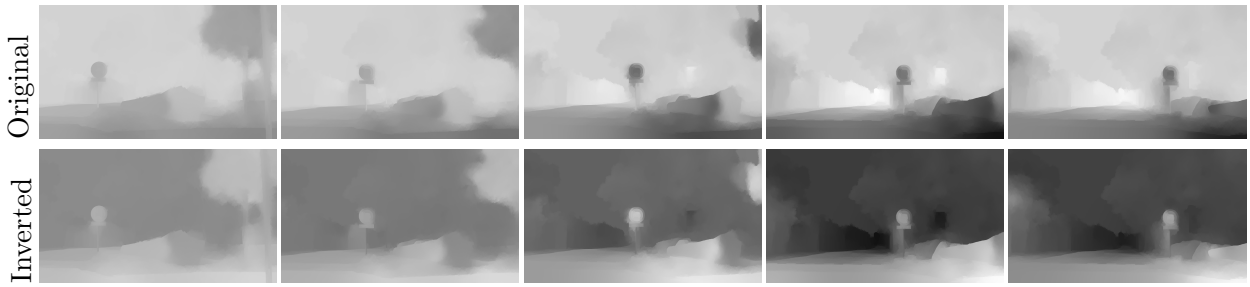


Figure 7.4: An example of depth reversal and its correction. The results are shown for the same frames as in Figure 7.17.

The depth reversal phenomenon is an artifact of using disparity as a proxy for depth. At its most fundamental, disparity is simply

$$d = x_{tgt} - x_{ref} \tag{7.2}$$

where x_{ref} and x_{tgt} are the horizontal positions of a feature in the reference and target images. So if the feature moved left then $x_{tgt} > x_{ref}$ and $d > 0$. However, if it moved right then $x_{tgt} < x_{ref}$ and $d < 0$.

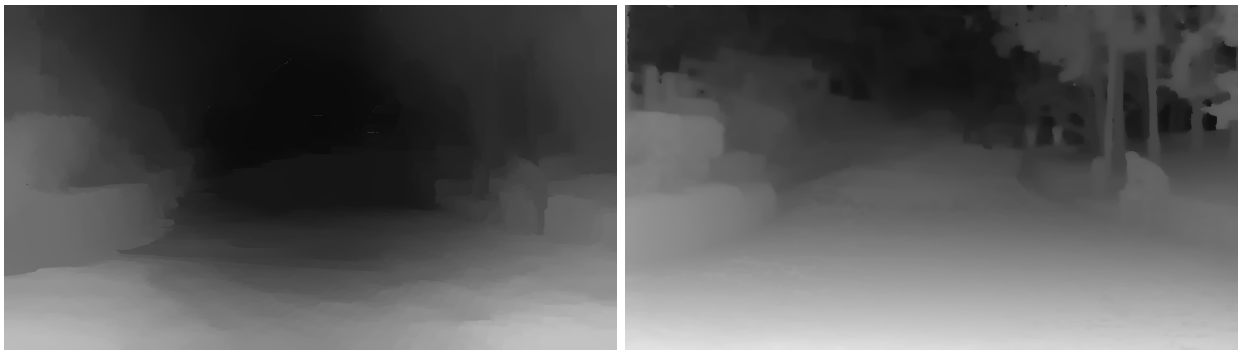
For rectified cameras, the sign will always be the same and detecting this condition is straightforward: most of all of the disparity values will be negative. But this is much more difficult for general motion due to the effect of the infinite homography (Section 3.4). In that case disparity values can be both positive *and* negative and the sign depends on which side of the convergence

surface they are on. Therefore a large number of disparities may be negative even if no depth reversal has occurred.

As it is designed right now, the framework requires that the user recognize that a depth reversal has occurred and fix it. Because the reversal either happens or it does not then flagging the event is easy for a user to do. However, it would be beneficial to detect or correct automatically and this will be the focus of future work.

7.2.3 Robustness and User Correction

There are a few places where the proposed framework can do better than ACTS and this is due to how propagation is done. ACTS is a purely automated approach with the depth maps generated through a form of multiview stereo. MVS generally uses dense (per-pixel) estimation, using camera poses to determine how to search along epipolar lines. This is a very accurate technique, as shown in Figure 7.5.



(a) Proposed (b) ACTS

Figure 7.5: Comparison between dense and sparse propagation. The samples are both from "Quad". Dense propagation can better extract details on the the right side of the image as it is not reliant on just whatever was tracked by the feature tracker.

However, what is shown in Figure 7.5 is actually a fairly "easy" sequence to process. The scene is completely static, there are many good features to track (helpful with SfM) and there are few motion artifacts such as motion blur. The moment this is *not* the case, dense estimation performs quite poorly. That is because a dense method cannot ignore any data; an estimate is found for each pixel. An example of this is Figure 7.6.

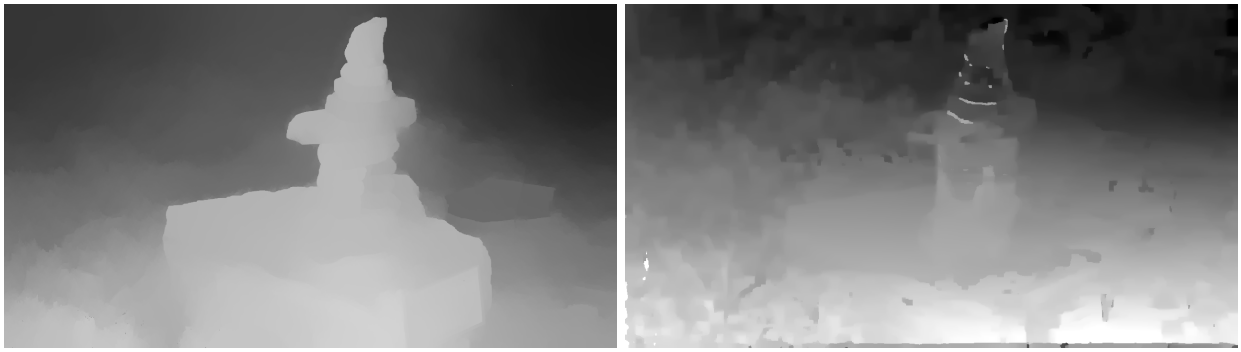
While the sparse propagation is not as detailed as what ACTS produces, it is noticeably more consistent with what one would expect of the scene depth. ACTS' dense estimation method produces noticeable artifacts where the pedestrians are. The proposed method, because any features tracked on the pedestrians were flagged as outliers or too few were tracked to effect the propagation, are nearly invisible. The street depth is somewhat inconsistent but that is mainly due to an insufficient number of features being tracked. This is on top of the fact that the fast camera motion

(a) Proposed (b) ACTS

Figure 7.6: Comparison between dense and sparse estimation for "Street". Dense propagation has noticeable artifacts caused by the moving pedestrians.

and strong motion blur corrupt the overall results obtained by ACTS. Even the online propagation is able to better handle this.

Because of how dense estimation is performed, it is possible for ACTS to experience some rather strange artifacts if the scene is not "perfect". For example, the camera in the "Inuksuk" sequence is stationary for about thirty frames and there is no usable depth information during that time. With temporal propagation this is not a problem and it produces perfectly useful depth maps. As evident in Figure 7.7, this is in stark contrast with what ACTS produces.



(a) Proposed (b) ACTS

Figure 7.7: Depth map anomalies that occur with insufficient camera motion.

Some of the anomalies are due to how ACTS performs dense matching. Mean shift [102] is applied to each frame and the segments are treated as 3D planes. For thin regions that are traditionally hard to segment, such as the tree branches in the "Condo Development" sequence, the foreground and background can get grouped together. The result is that the thin structures are "thickened". This can be best seen in Figure 7.13. Therefore, methods not reliant on segmentation may handle this better.

One other type of anomaly is caused by reflective or shiny[4] surfaces. These surfaces can cause inconsistencies when constructing a cost volume such as those used by dense disparity methods. For example, a reflection will move *with* the camera even if the scene is completely static. This is something that breaks the assumption of how features are supposed to be moving and is an inherent problem with dense estimation.

Figure 7.8: An example of how dense estimation has difficulty with reflections. Frame 60 from the "Residence" sequence was used for this example.

Figure 7.8 shows how sparse propagation can do a better job at handling these situations. Because the reflections end up being part of the cost volume, they produce depths inconsistent with the scene geometry. The sparse propagation is able to better match the scene since it is simply less likely to track these features or consider them inliers. In the rare case that it does, it is possible to remove them manually.

The main advantage to the proposed approach is the ability to adjust the generated depth estimates at some later time. An example of this is shown in Figure 7.9 that provides some frames the "Condo Development" sequence. During tracking, features on the pedestrians were not flagged as outliers and so became part of the depth estimates. Because the pedestrians are moving as fast as the camera, the features have zero disparity. This is seen by both the proposed approach and ACTS. However, with the proposed approach these erroneous features can be removed by the user and new "correct" maps can be generated from the modified labels.

7.2.4 Processing Times

Processing times for all of the different samples are shown in Table 7.4. All of the processing was performed on a desktop PC with a Intel Core i7-4771 3.5 GHz quad-core processor with 16 GB of RAM and an Nvidia Geforce GTX 660 Ti GPU with 2 GB of on-board memory. The tracking and depth estimation code is implemented in Matlab using functionality provided by the Computer Vision toolbox. The optical flow and propagation (both versions) were implemented as

[4]These are technically known as "non-Lambertian" surfaces.

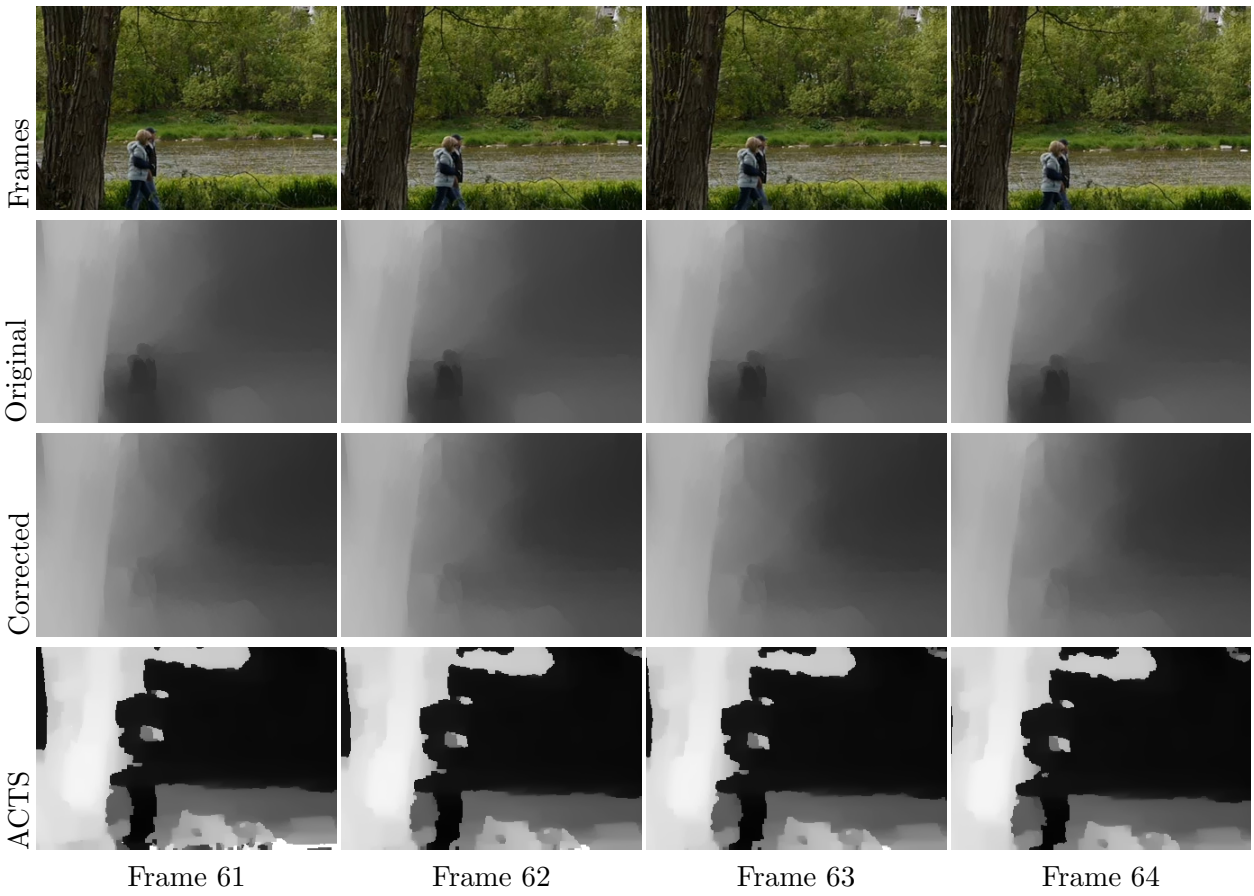


Figure 7.9: An example of correcting a video sequence. In the original sequence, features were tracked on the pedestrians and so have incorrect depth estimates. After having the features manually removed, the result is noticeably more consistent with the scene depth. The ACTS results have been provided for comparison.

Table 7.4: Running times for the primary test sequences. All times are in minutes. The ACTS runtime has been included as well, however it is just for the depth recovery (map generation) portion as the SfM times are not reported by the software. ACTS runtimes for the samples from [71] are not provided as those depth maps are available from the authors.

		Temporal			
Sequence	Online	Total	Estimation	Interpolation	ACTS
Inuksuk	21.1	84.1	81.3	2.8	110.8
Quad	23.1	107.5	103.9	3.6	158.1
Structure	30.6	72.8	68.4	4.4	168.7
Development	9.9	29.8	27.5	2.3	39.4
Residence	11.6	24	21.5	2.5	51.9
Street	10.7	18.5	16	2.5	51.9
Flowers	8.6	15.1	14.4	0.7	-
Road	4.7	13.2	11.8	1.4	-

OpenMP-accelerated MEX functions.

It should be pointed out that the implementation is *not* optimized for speed. The tracking front-end has a number of different operations occurring in parallel[5] and at the moment those are being done one at a time. The filtering functions, used for optical flow and propagation use OpenMP to run faster however they could very easily be implemented to run on a GPU using CUDA[6] or OpenCL.

Even with these caveats, the proposed framework is still faster. ACTS, which is GPU-accelerated, reports the time for obtaining the depth maps. In terms of the framework, this is analogous to propagation for the temporal method. Even including the tracking time, the proposed method is much faster and could be made to be even faster.

[5] In that the optical flow and depth estimation can both be done at the same time.
[6] Gastal and Oliviera reported a 23× speed up in [57] when using CUDA.

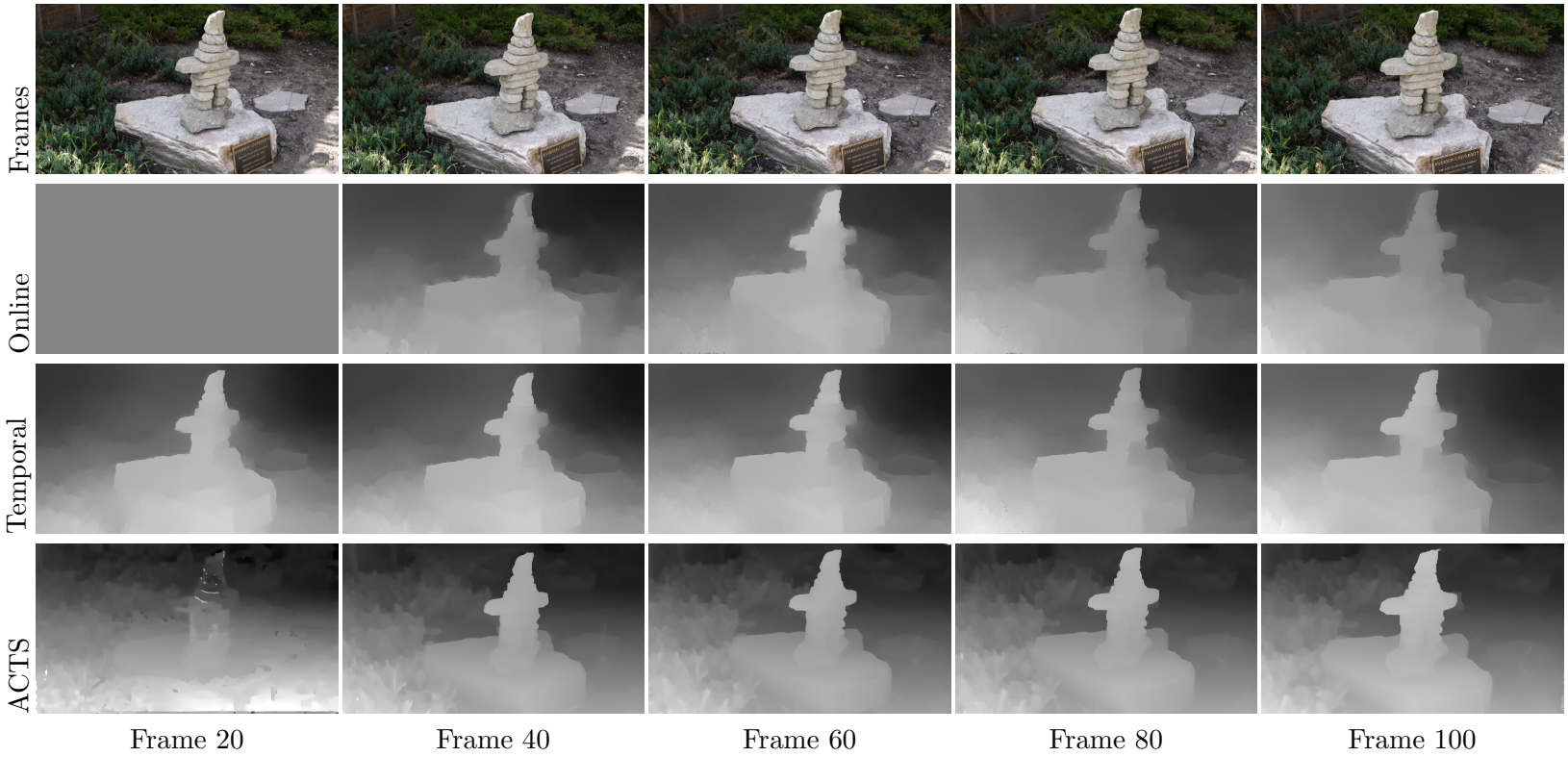


Figure 7.10: Sample frames for the “Inuksuk” sequence.

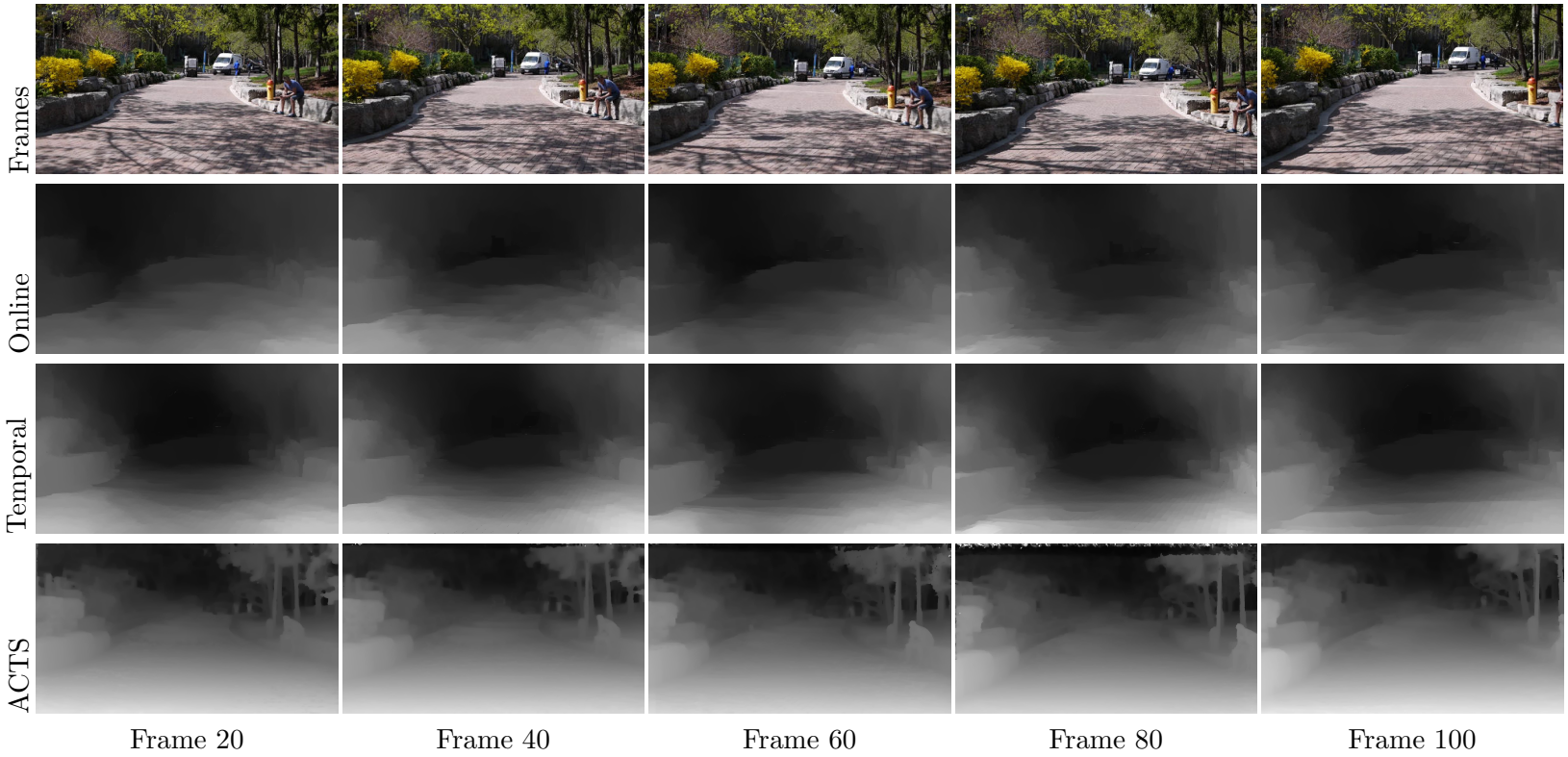


Figure 7.11: Sample frames for the “Quad” sequence.

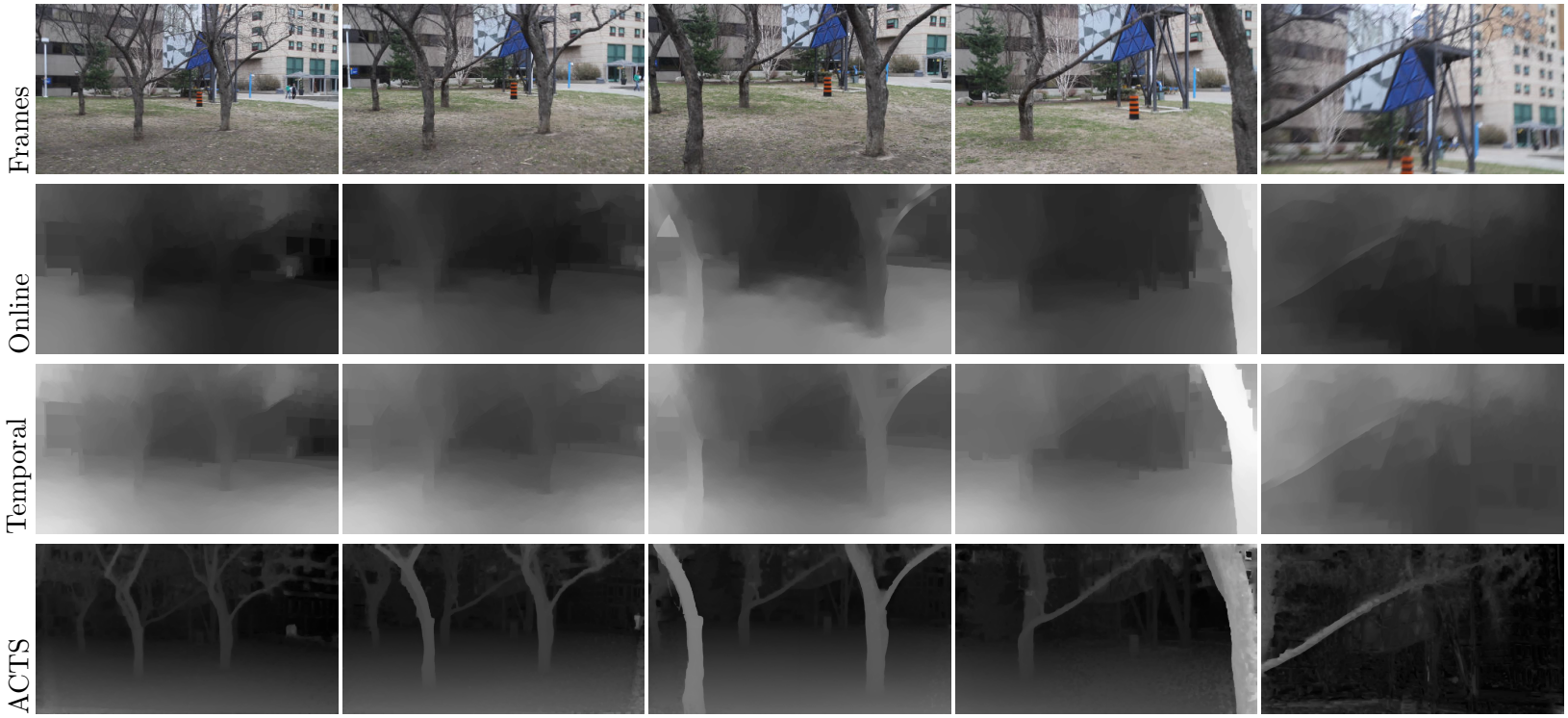


Figure 7.12: Sample frames for the “Ryerson Structure” sequence.

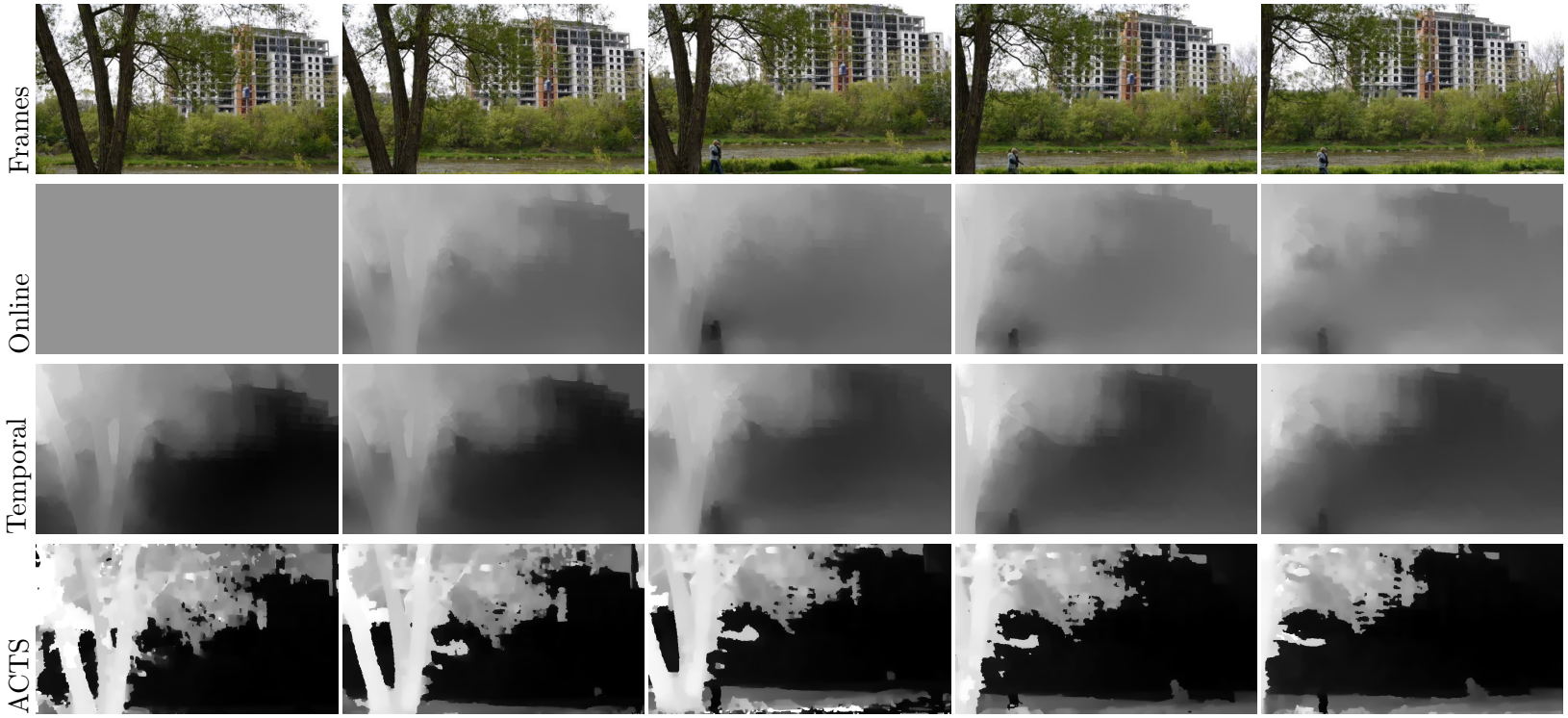


Figure 7.13: Sample frames for the "Condo Development" sequence.

Figure 7.14: Sample frames for the "Residence" sequence.

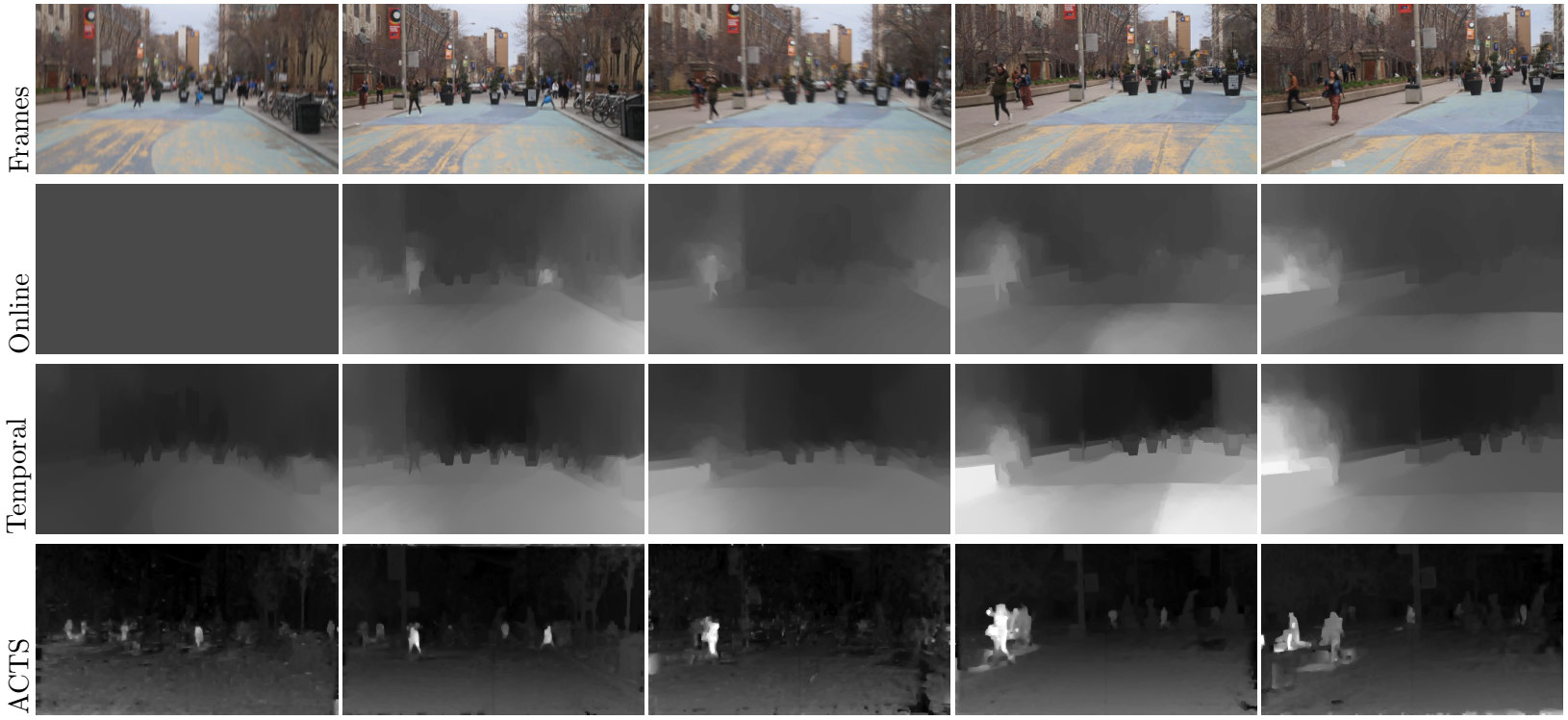


Figure 7.15: Sample frames for the “Street” sequence.

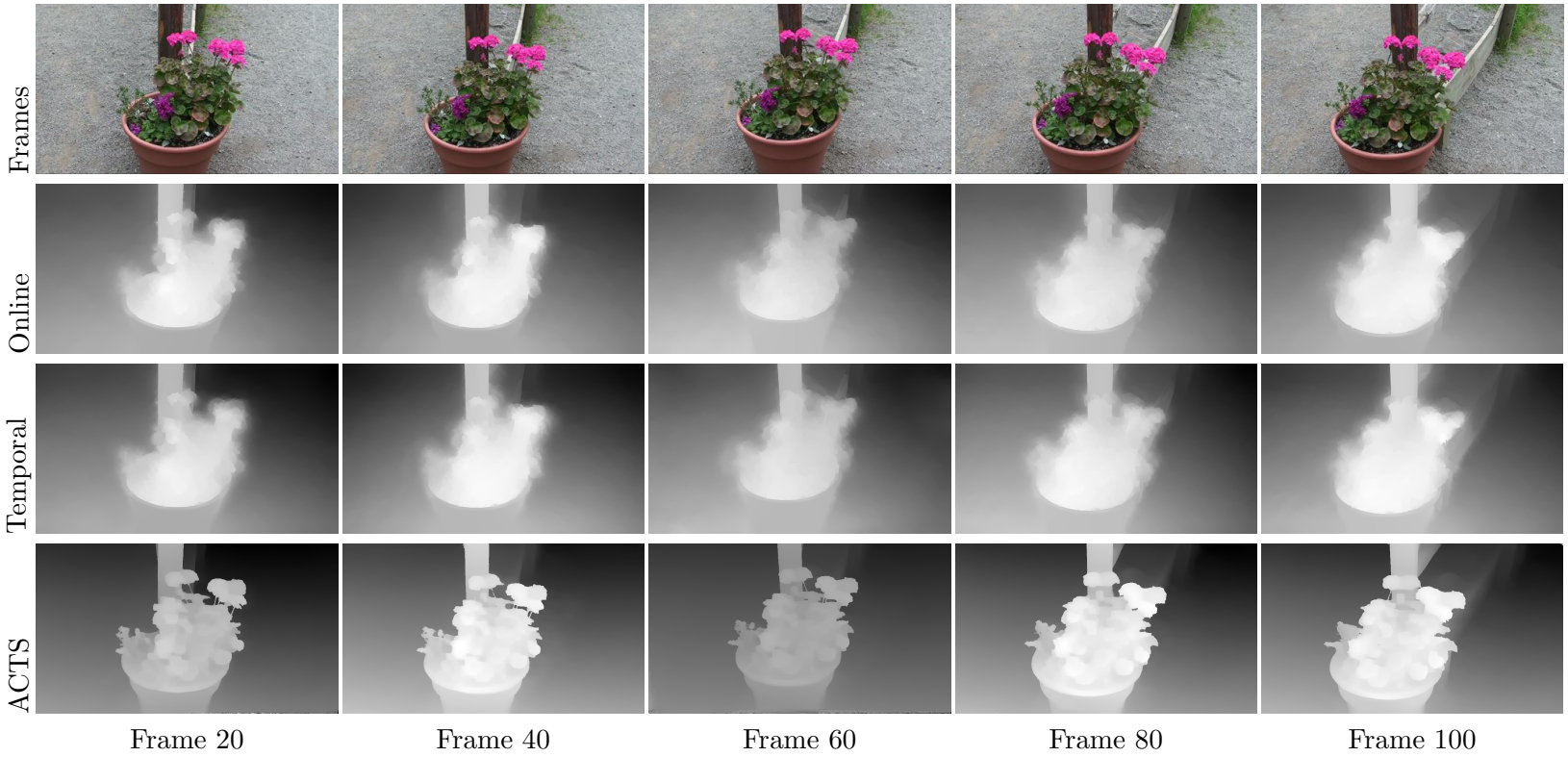


Figure 7.16: Sample frames for the "Flowers" sequence.

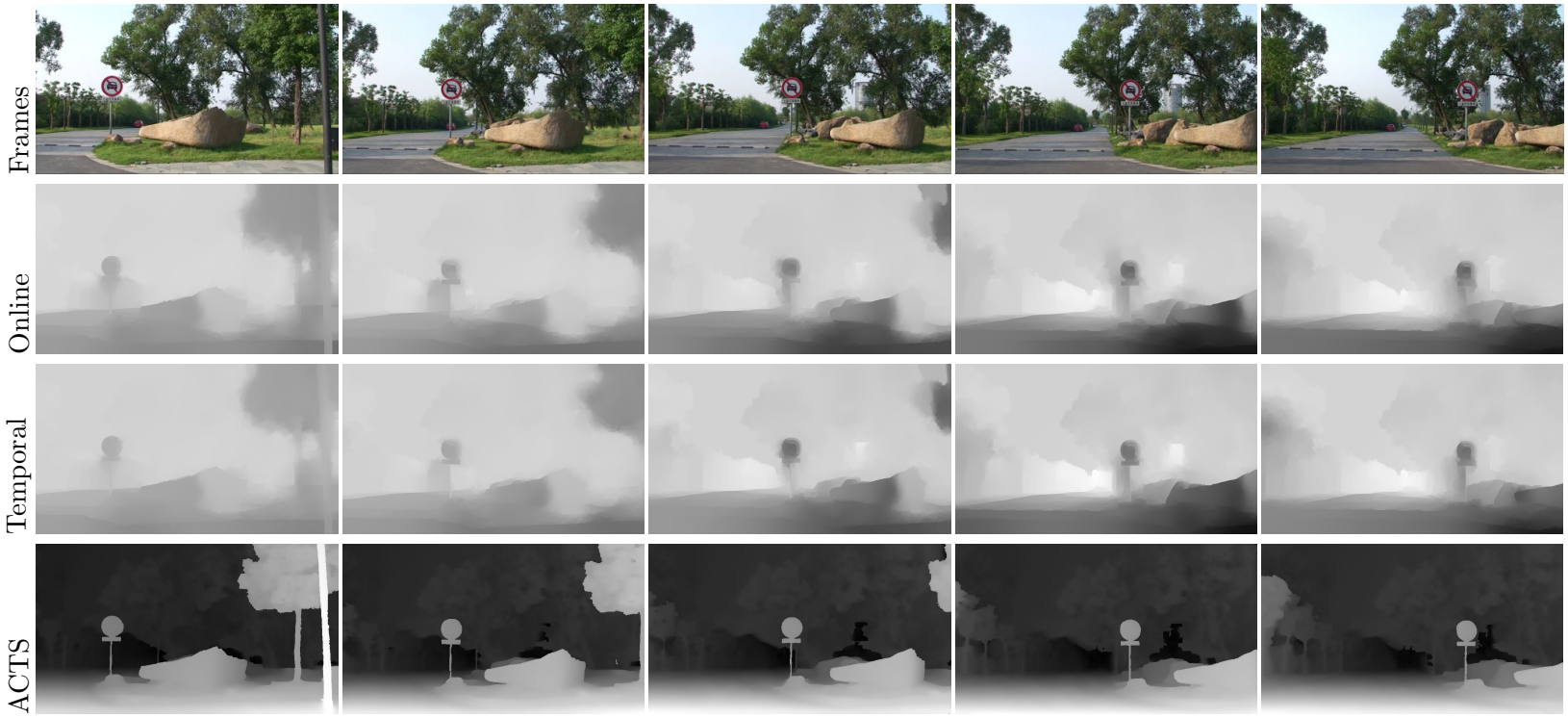


Figure 7.17: Sample frames for the “Road” sequence.

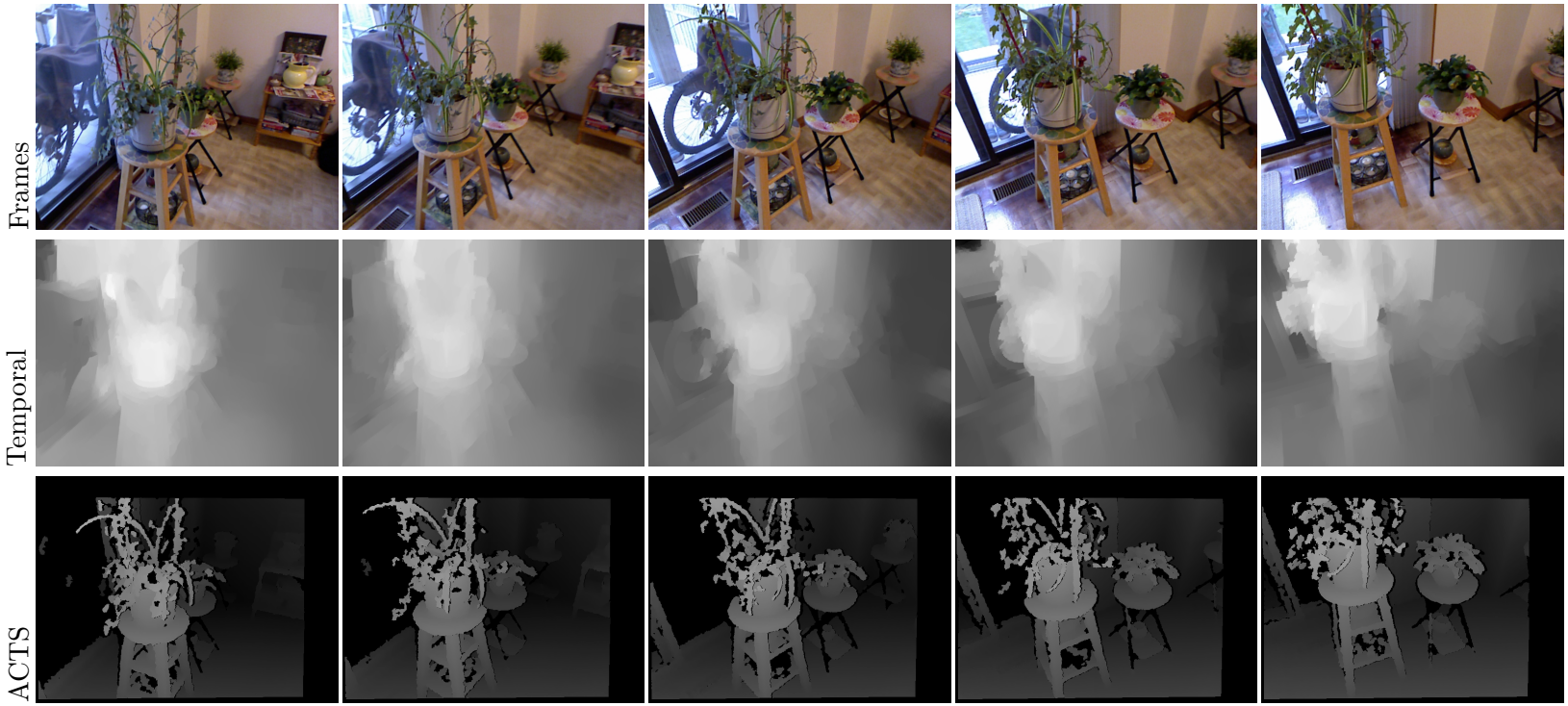


Figure 7.18: Sample frames for the “Plant (Kinect)” sequence.

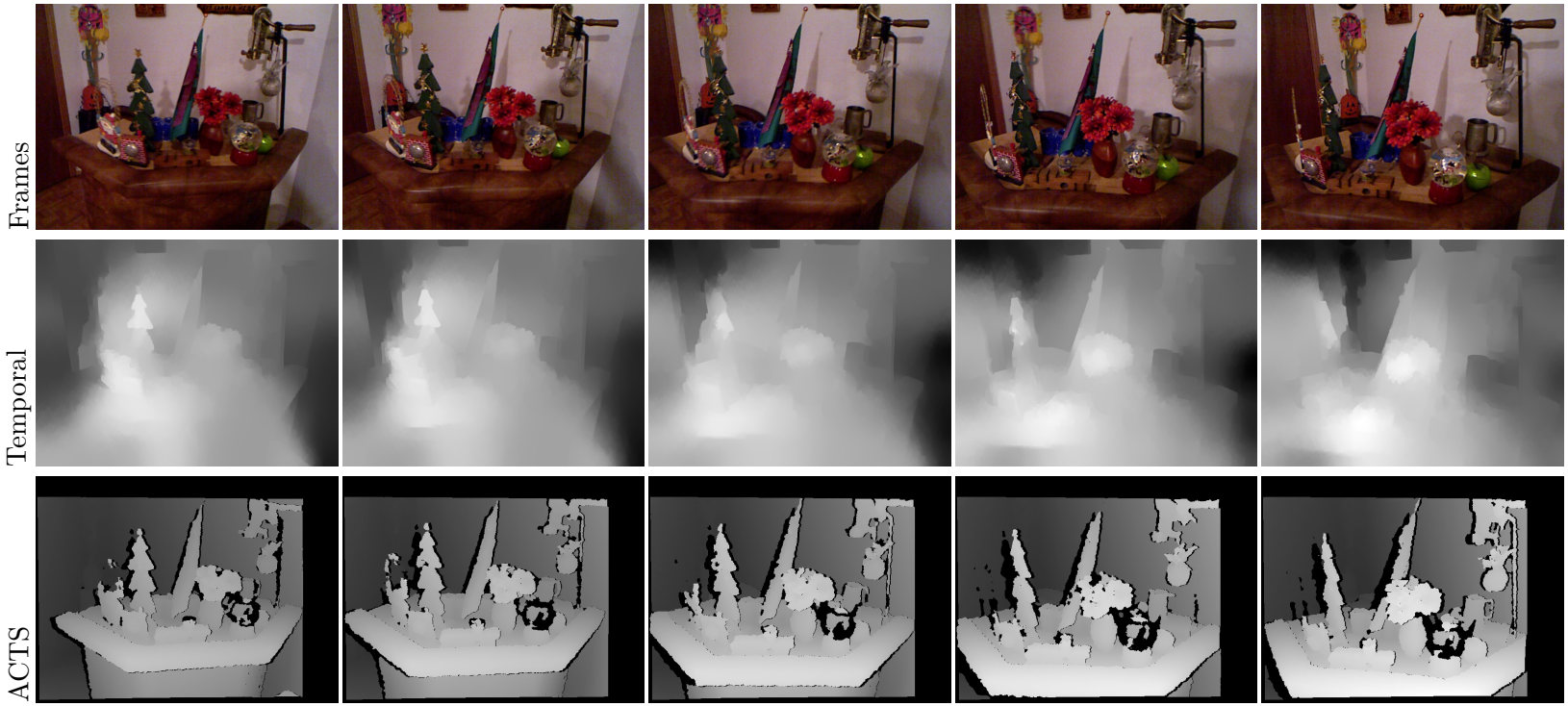


Figure 7.19: Sample frames for the “Bar (Kinect)” sequence.

Chapter 8

Conclusion

This dissertation has presented a method for automatically extracting relative depth from a video sequence. By applying well-established computer vision methods, the method can be applied to a wide variety of footage with only one restriction: the camera must have some translational motion. By using sparse interpolation it is able to handle complex scenes (i.e. independently moving objects) better then with dense estimation. While this may result in a loss of detail in certain cases, its flexibility outweighs this downside.

The main contribution of this dissertation is a novel method for estimating depth in a relatively simple fashion. It produces depth estimates in agreement with those produced using more complex methods, namely SfM. As with some recent work [76, 77, 78], the method seeks to decouple depth estimation from the map generation. However, as they are complete 2D-to-3D pipelines, [76] and [77] specifically, there is a fair degree of interdependence between the different stages.

The proposed framework completely separates the depth estimation from map generation and so is quite flexible. In fact the quality can be changed at will, as was demonstrated in Chapter 7. The advantages to the proposed framework are that it is

- computationally efficient and simple to implement;
- amenable to user corrections;
- can be easily reconfigured to change the results quality or even use different methods for depth estimation or map generation;
- comparable to SfM/MVS-based techniques when the scene is static and well-captured;
- can tolerate stationary or rotating cameras;
- and produces subjectively better results than a method such as ACTS when the scene is dynamic or poorly captured.

For the "ideal" case, where nothing moves and the scene has been perfectly captured, SfM and MVS will produce superior depth maps. They are mature techniques specifically designed to recover high-quality depth information. However, the moment this assumption breaks, which is quite often, the results are less than stellar. Furthermore, if something does break, it is very difficult to correct it. The proposed framework is able to generate depth maps when the scene has been poorly captured and can be corrected very easily.

Future work can be divided into three sections: tracking, algorithm and evaluation. These reflect the major areas where the proposed method can be improved upon and extended.

Tracking

The actual tracking framework is quite simple and relies on basic feature detection and tracking using FAST features and KLT. As such, it will work quite poorly if the scene has few "good" features to track since both RANSAC (for estimating the fundamental matrix) and the feature interpolator works best if more features are available. This can be somewhat alleviated through user input but it is not an adequate solution.

One potential alternative is to use dense optical flow as the basis for the feature tracker, potentially something similar to Particle Video [103]. In fact, this was done in the earliest form of this dissertation [104] but abandoned due to the overhead imposed by having to pre-compute the optical flow. But recent research into fast dense optical flow [105] could help replace the KLT feature tracker with something that could better handle scenes that are difficult to track.

It should be pointed out that this would not require any major changes to the tracking framework from Chapter 4.1. The only difference would be how the features are generated and propagated. The buffering strategy (Chapter 4.1.2) would not need to change at all and it would still be amenable to online processing. Because this could potentially generate and retain more features, it could also improve the quality of the depth maps generated in "online" mode.

Algorithm

The algorithm that obtains the actual sparse depth estimates has one major problem: depth reversal. This is trivial for a user to fix but it is problematic for automated processing. Due to the infinite homography (Chapter 3.4.1) there is no easy way to determine how to properly interpret the magnitudes of the disparity values, i.e. if large values are close and small values are far or vice-versa.

There are two possible ways to approach this. One would be to perform some sort of limited reconstruction to "guess" the structure. This would not need to be a full SfM-based approach; just something capable of discerning overall motion in order to decide how the magnitude should be interpreted. However, because the optical flow information, in one form or another,

is available, then it could be possible to use that as a hint for correcting for a depth reversal. The method used by Liao et al [77] to correct for perspective motion could potentially be used here.

Being able to correct for motion would actually enable the framework to do something new: fast depth estimation for unstructured image collections. Due to the depth reversal, the framework cannot work well with unstructured image collections because the depth values for one image pair will be the inverse of those for another image pair. When this happens, the aggregated depth effectively becomes zero and so is not particularly useful. Detecting the depth reversal would allow all of the depth magnitudes to have the same meaning and allow the aggregation stage to work properly.

Evalution

The proposed framework was mainly compared against Zhang et al's [71] video depth recovery. This comparison was designed to determing if the novel depth estimation algorithm from Chapter 4 could provide depth values similar to ones obtained using SfM. All of the primary test footage was captured using a hand-held camera with constant camera parameters. This type of footage is well-suited for SfM and is why it was used for the comparison. This made it possible to claim that the depths obtained by the proposed method were similar to SfM.

Ideally the evaluations should be made against ground truth data for a variety of different footage. Unfortunately, obtaining ground truth depth data is notoriously difficult and often requires specialized equipment. For example, the Make3D project [64] used laser range finders to obtain accurate depth information but the captured resolution was quite low.

A Microsoft Kinect is another option (the method used to obtain the secondary samples) but the processing required to make a valid comparison can impact the final results. The RGB camera quality is poor and the depth and RGB image are not registered, with large regions of the image having no depth information. Furthermore, the device does not work well outdoors.

An better solution is to use purely synthetic data, i.e. computer generated imagery or CGI. The quality of modern CGI is such that it can provide a more than sufficient surrogate for "real" imagery. In fact, this has been used to create an optical flow benchmarking suite [106]. The main issue for creating such a test set is more practical than theoretical. For instance, the optical flow test suite was created using the sources for "Sintel", an open-source CGI movie where the source files (art assets, 3D models, scene files, etc) were all made publicly available and released under a Creative Commons licence[1]. It would be beneficial to create a test suite from a variety of films to provide a wide sampling of different types of footage. While the licencing issues will need to be sorted out, creating such a test suite would be quite useful for future researchers.

[1] `http://www.sintel.org/sharing/`; Accessed July 2, 2014.

Appendix A

Geometrically Robust Information Criterion

A particular problem with estimating something such as the fundamental matrix is that there is no guarantee that it will properly describe the observed motion. Namely, the fundamental matrix is rank-deficient[1] and so has no inverse. Intuitively this can be understood by the fact that it maps points to lines. A feature in one image has (potentially) infinitely many correspondences in the other image. This makes it inherently less stable than a homography that maps points to points and is invertible.

This particular problem was studied by Torr et al [89,90,91] in order to determine if a particular estimate of $\mathbf{F}$ was reliable. The actual problem being solved is one of model selection: given a homography or a fundamental matrix, which one describes the observed motion *better*. This is done by scoring the two models such that the model with the lower score provides a better explanation of the observed data. The method used to assign this score is called the Geometrically Robust Information Criterion (GRIC).

GRIC is a scoring method that looks at how well the model itself fits the given data while applying a penalty relative to its degrees of freedom. This is necessary to prevent over-fitting since a model with a higher degree of freedom will *always* be able to better fit a model with a lower degree of freedom. The GRIC score is defined as

$$GRIC = \sum_{i=0}^{n-1} \rho(e_i^2) + \lambda_1 dn + \lambda_2 k, \tag{A.1}$$

where

$$\rho(e^2) = \min \left\{ \frac{e^2}{\sigma^2}, \lambda_3(r-d) \right\} \tag{A.2}$$

[1]Specifically, $\text{rank}\,(\mathbf{F}) = 2$. This is a result of $\left[\tilde{e}_b\right]_\times$ from (3.26) having $\det(\left[\tilde{e}_b\right]_\times) = 0$. Skew symmetric matrices have a determinant of zero and multiplication with another matrix causes the result to also have a zero determinant.

is a robust penalty function. The constants are defined [107](Sec. 7.3.2) as follows:

n the number of feature matches

e_i the error or residual between the measurement and the model

d dimensionality of the model (2 for homographies, 3 for fundamental matrices)

k number of model parameters (8 for homographies, 7 for fundamental matrices)

r dimensionality of the data (for 2D features, twice the number of views)

σ standard deviation of the measurement error

The λ_* terms, where $* \in \{1, 2, 3\}$, are defined as

$$\lambda_1 = \log(4), \tag{A.3a}$$

$$\lambda_2 = \log(4n), \tag{A.3b}$$

$$\lambda_3 = 2. \tag{A.3c}$$

While the σ term is in theory obtained from the measurement error, it can be difficult to obtain in practice. That is because it requires knowing how much error there is in the measurement, which is not actually possible short of creating a complete mathematical model of how the feature correspondences are generated. In the GRIC formulation, it is assumed that the models were found using a Maximum Likelihood Estimate (MLE) and so there is an underlying assumption on the distribution of the noise. As such it must be treated as a tuneable parameter. Therefore if $\mathbf{F}_{k,l}$ is generated using an MLE-based approach, that value of σ must be used. However if an MLE-based approach is *not* used[2] then σ controls the "sensitivity" of GRIC.

A very small value of σ implies that the data is nearly noise-free while a large value assumes a high-degree of corruption. It controls how likely any particular value of e_i will be declared an outlier, i.e. given a cost of $\lambda_3(r-d)$. In this case this simplest option is to let $\sigma = 1$ or not scale the residuals. Because the residuals, as will be defined shortly, are based on geometric measurements, this corresponds to a standard deviation of about ± 1 pixel.

The choice of model residual depends on the model being tested. For homographies the measure is straightforward: just use the sum-of-squares distance between the actual feature and the transformed feature from the other image. This is

$$D_{\mathbf{H}}(\vec{x}, \vec{y}) = \|\vec{x}' - \vec{y}\|^2, \tag{A.4}$$

where $\vec{x}'$ is $\mathbf{H}\tilde{x}$ after a division by w. Note that if $\mathbf{H}$ perfectly describes the mapping of $\vec{x} \to \vec{y}$ then $D_{\mathbf{H}}(\vec{x}, \vec{y}) = 0$. This definition is not symmetric since it only deals with the forward mapping.

[2]RANSAC is not based on MLE, though it does have an MLE variant [108].

This is corrected by

$$D'_{\mathbf{H}}(\vec{x},\vec{y}) = D_{\mathbf{H}}(\vec{x},\vec{y}) + D_{\mathbf{H}^{-1}}(\vec{y},\vec{x}) \tag{A.5}$$

so that the total error is the error from mapping $\vec{x} \to \vec{y}$ as well as $\vec{y} \to \vec{x}$.

The definition of the residual for the fundamental matrix is a bit different but the most straightforward approach[3] is the use the distance to the nearest epipolar line, which measures how well a feature matches the epipolar constraint. This is defined as

$$D_{\mathbf{F}}(\vec{x},\vec{y}) = \frac{(\tilde{y}^T\mathbf{F}\tilde{x})^2}{(\tilde{l}_{\vec{x}}(0))^2 + (\tilde{l}_{\vec{x}}(1))^2}, \tag{A.6}$$

where

$$\tilde{l}_{\mathbf{F}} = \mathbf{F}\tilde{x} = \begin{bmatrix}\tilde{l}_{\vec{x}}(0) & \tilde{l}_{\vec{x}}(1) & \tilde{l}_{\vec{x}}(2)\end{bmatrix}^T \tag{A.7}$$

This expression is just the point-line distance from $\vec{y}$ to the line generated by $\mathbf{F}\tilde{x}$. Again, like (A.5), this only finds the error from the $\vec{x} \to \vec{y}$ and not vice-versa. This error is made symmetric[4] by

$$\begin{aligned} D'_{\mathbf{F}}(\vec{x},\vec{y}) &= D_{\mathbf{F}}(\vec{x},\vec{y}) + D_{\mathbf{F}^T}(\vec{y},\vec{x}). \\ &= (\tilde{y}^T\mathbf{F}\tilde{x})^2 \left\{ \frac{1}{(\tilde{l}_{\vec{x}}(0))^2 + (\tilde{l}_{\vec{x}}(1))^2} + \frac{1}{(\tilde{l}_{\vec{y}}(0))^2 + (\tilde{l}_{\vec{y}}(1))^2} \right\}, \end{aligned} \tag{A.8}$$

where

$$\tilde{l}_{\mathbf{F}^T} = \mathbf{F}^T\tilde{y} = \begin{bmatrix}\tilde{l}_{\vec{y}}(0) & \tilde{l}_{\vec{y}}(1) & \tilde{l}_{\vec{y}}(2)\end{bmatrix}^T. \tag{A.9}$$

Once the error terms have been calculated, using GRIC is straightforward. Simply compare the scores and the model with the *lower* score can be assumed to better describe the observed motion. This does not guarantee that the model is in fact the true model, just that it is the better of the two being compared.

[3] There are alternate distance measures, such as Sampson distance/error, but for the purposes of model verification the epipolar distance is sufficient.

[4] Because $\vec{y}^T\mathbf{F}\vec{x} = \vec{x}^T\mathbf{F}^T\vec{y}$, as shown in (3.28), this term only needs to be computed once. This is often how the symmetric epipolar distance is presented in the literature.

Appendix B

Domain Transform Filtering

Non-linear filters, specifically edge-aware filters, are good candidates for problems such as sparse, non-linear interpolation. The most well-known of these is the bilateral filter [54], though it requires a fair degree of modification in order to run quickly [55]. The Domain Transform (DT) filter [57] described here is used in this dissertation because it provides a good trade-off between results quality and simplicity of the implementation.

To begin, the domain transform filter proposes that there is a transformation $T(\cdot)$ that will take some original one-dimensional input $x(u)$ and transform it into $y(v)$ such that the distance between samples in the transformed domain is proportional to $|x'(u)|$ where $x'(u)$ is the derivative of $x(u)$. In other words, it compresses $x(u)$ based on the L_1-norm. Specifically the transform is written as

$$T(v) = \int_0^v \left(1 + \frac{\sigma_s}{\sigma_r} \sum_{c=1}^{D} |\vec{x}_c'(u)|\right) du, \tag{B.1}$$

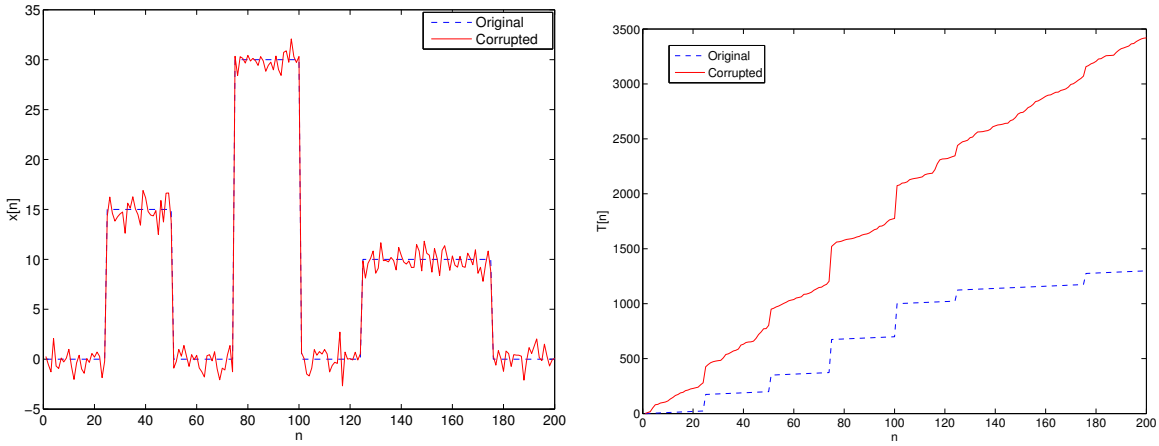
where σ_s and σ_r are the filter's spatial and range parameters, respectively. If the input is vector-valued, i.e. a colour image, then the individual D channels are summed together. The spatial parameter effectively controls the filters spatial extent while the range parameter controls the sensitivity of the filter to $x'(u)$. Please note that (B.1) is defined as a continuous function as this was how it is presented in [57]. A more appropriate form of the transform is

$$T[n] = \sum_{i=0}^{n} \left\{1 + \frac{\sigma_s}{\sigma_r} \sum_{c=1}^{D} \Big|\vec{x}_c[i] - \vec{x}_c[i-1]\Big|\right\} \tag{B.2}$$

as a digital image is by definition a discrete signal.

The transform $T[n]$ is unique to the particular signal. Again, that is because it depends on the L_1-norm (absolute value of the derivative) of $x[n]$. An example of the transform is shown in Figure B.1. The original signal, shown in Figure B.1a, is a series of pulses of differing amplitudes. A corrupted version, $x_n[n] = x[n] + N(0, 1)$, is also shown to demonstrate how the transform is

signal-dependant. The two transforms $x[n] \rightarrow T[n]$ and $x_n[n] \rightarrow T_n[n]$, are shown in Figure B.1b. There are clear "jumps" in the transform for $x[n]$ while they are less clear in the transform for $x_n[n]$. This is the effect of the noise on the derivative.



(a) Original Signal

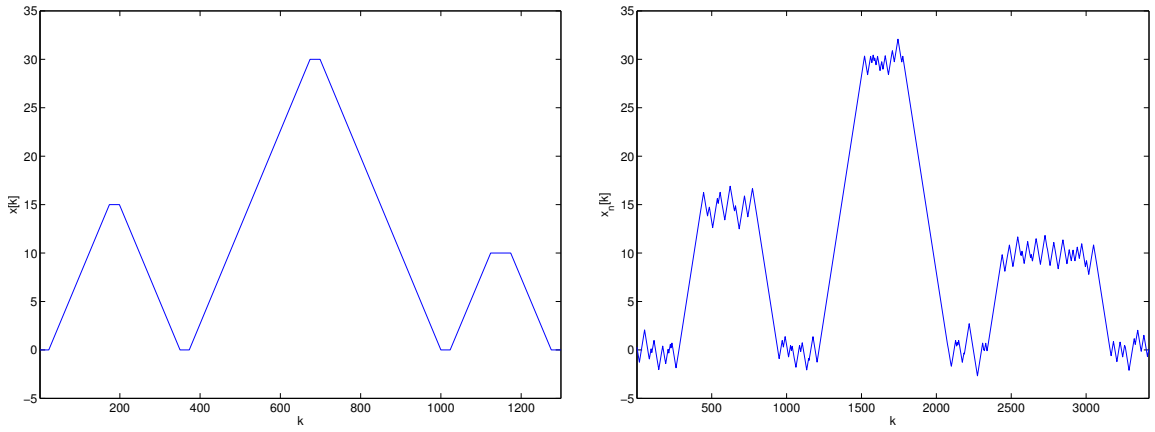
(b) Domain Transform

Figure B.1: An example of the domain transform. The transform is shown for two signals: a set of pulses of varying heights and the same signal but corrupted with zero-mean additive Gaussian noise with unit variance. The transform for the corrupted signal is noticeably different and this is due to the effect of the noise on the L_1-norm. The transform parameters were chosen so that $\sigma_s/\sigma_r = 10$.

The signals in the transformed domains are shown in Figure B.2. The transformation does not affect the signal amplitudes. Rather it transforms the *spacing* between samples by $T[n] \rightarrow k$ so that if there is little change in the signal then those samples will be close in the transformed domain $x[k]$. Conversely, if there is a large difference between samples then the samples will be far apart in the transformed domain. This is why the pulses are more "pinched" in the original signal (Figure B.2a) versus those in the noisy signal (Figure B.2b). It is also why the range of the domain is $0 \leq T[n] \leq 1300$ for the clean signal and $0 \leq T_n[n] \leq 3600$ for the noisy one.

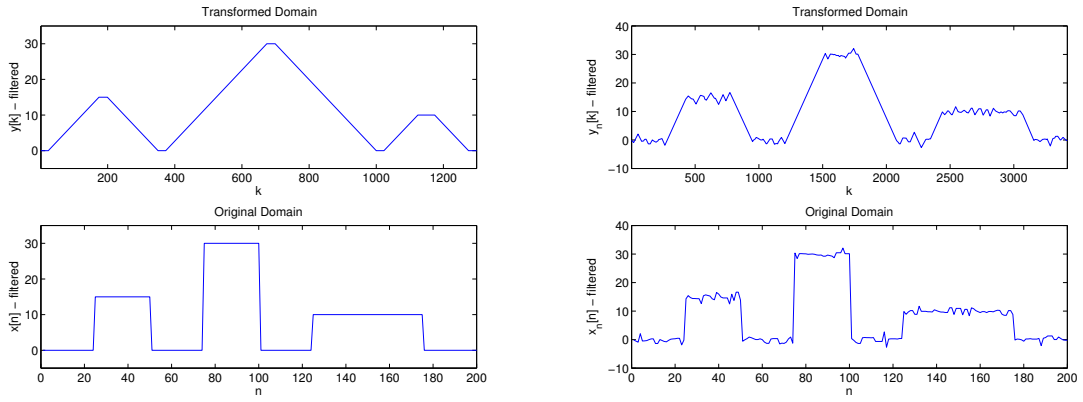
The purpose of the domain transform is so that a normal, *linear* filter can be applied in the transformed domain instead of the signal's original domain. Because edges are expanded in the transformed domain, a low-pass filter, for instance, will not affect the edges as much as in homogeneous regions. An example of this is shown in Figure B.3. The two signals were filtered in the transformed domain using a box filter with a radius $r = 20\sqrt{3}$. This value is obtained from the parameters used to generate the domain transform and will be elaborated upon shortly.

As defined, the DT filter can only be applied to one-dimensional signals. In fact, as noted by Gastal and Oliviera it is impossible to find this type of domain transform in higher dimensions (refer to [57] for more information). However, Gastal and Oliviera provide a solution to this: iteratively filter an image $I(x, y)$ while treating the DT filter itself as a standard separable filter. Therefore



(a) Original Signal (b) Corrupted Signal

Figure B.2: Signals in the transformed domain. The distance between samples in the transformed domain is proportional to their L_1-distance.



(a) Original Signal (b) Corrupted Signal

Figure B.3: Signals after filtering in the transformed domain. The edges in the signal are preserved but, as can be seen in the corrupted version, the noise is smoothed out.

filtering is done as follows:

1. Filter $I(x, y)$ horizontally to produce $I'(x, y)$.
2. Filter $I'(x, y)$ vertically to produce $I_{\text{filt}}(x, y)$.
3. Let $I(x, y) = I_{\text{filt}}(x, y)$ and return to step 1; repeat N times.

$I(x, y)$ is the image being filtered and $I'(x, y)$ is the intermediate filter result. Computationally this is very efficient because each row/column is processed independently and can be parallelized. Gastal and Oliviera showed that after five iterations there is no noticeable change in the results and that three iterations are often sufficient.

The actual filtering is not performed in the transformed domain; the example shown in Figure B.3 was simply meant to be illustrative. Rather, the general approach is to use the transform domain to control the size of the filter kernel, whatever it may be, back in the *original* image domain. The original paper proposed three different types of DT filters: Normalized Convolution (DT-NC), Recursive Form (DT-RF) and Interpolated Convolution (DT-IC).

The DT-NC filter is defined as a straightforward box (moving average) filter in the *image* domain such that

$$y[n] = \frac{1}{j-i} \sum_{m=i}^{j} x[m], \tag{B.3}$$

where i and j are determined by

$$\big(T[j] - T[i]\big) = 2r. \tag{B.4}$$

Note that this definition is symmetric in the transformed domain but *asymmetric* in the original image domain. This results from the fact that in order for (B.4) to be satisfied, $T[j] - T[n] = T[n] - T[i]$ for any value of r. Since $T[n]$ is monotonic, the only way for this to happen is if $T[n]$ is equidistant from $T[i]$ and $T[j]$. However, this condition does not apply in the original domain and so while $T[j] - T[i]$ is constant, $j - i$ is not. The DT-IC filter is similarly defined except that it is a continuous convolution of the form

$$y(u) = \frac{1}{2r} \int_{T(u)-r}^{T(u)+r} X(v)dv, \tag{B.5}$$

where $X(v)$ is the transformed *signal*. This definition somewhat complicates the implementation because it requires not only generating $X(v)$, which is done through interpolation, but also approximating the integral.

The DT-RF filter is a simple, first order IIR filter defined as

$$y[n] = (1 - a^d)x[n] - a^d y[n-1], \tag{B.6}$$

where $d = T[n] - T[n-1]$. This makes the filter's decay rate proportional to the L_1-distance so that more filtering is done when there is less of a change in the input signal. Unlike the DT-NC and DT-IC filters, the DT-RF filter is not symmetric and so has to be applied twice for each row/column: once in the forward direction (n increases) and once in the reverse direction (n decreases).

In the example presented earlier, the filter radius was chosen to be $r = 20\sqrt{3}$ and said to be related to the filter parameters that generated the domain transform. Because the filter is iterative, Gastal and Oliviera adjust the filter radius through

$$\sigma_{H_i} = \frac{\sigma_s}{\sigma_r} \sqrt{3} \frac{2^{N-i}}{\sqrt{4^N - 1}}, \tag{B.7}$$

where N is the number of iterations and $1 \leq i \leq N$ is the current iteration. This expression halves the variance of the filter in the transform domain each iteration so that as $i \to \infty$, the filter converges to a stable result instead of completely blurring the image. The DT-NC and DT-IC filters then define their radius' by

$$r = \sigma_{H_i}\sqrt{3}. \tag{B.8}$$

The DT-RF filter uses the σ_{H_i} to control its pole/zero locations in tandem with the derivative of $T[n]$. Specifically, the amount of feedback is defined as

$$a = \exp\left(-\frac{\sqrt{2}}{\sigma_{H_i}}\right). \tag{B.9}$$

Since the term inside of the exponential is negative, the filter is guaranteed to be stable.

While any of the DT filter variants would work, the DT-NC filter was chosen for this dissertation because of its strong edge response. Unlike the DT-IC and DT-RF variants, the DT-NC filter will respect strong edges to the point where there is a negligible amount of smoothing across the edge. To demonstrate this, Figure B.4 was filtered using the DT-NC and DT-RF filter variant using the exact same parameters. The results are shown in Figure B.5.

Figure B.4: A photo of Vancouver's historic Gastown district.

(a) DT-NC

(b) DT-RF

Figure B.5: A comparison between the DT-NC and DT-RF filters. Figure B.4 was filtered using the DT-NC and DT-RF filter variants, both with the exact same parameters ($\sigma_s = 100$, $\sigma_r = 1$ and $N = 3$).

What can be noticed when comparing the two versions is that the DT-RF image (Figure B.5b) appears to be slightly more blurry then the DT-NC image (Figure B.5a). While many of the smaller features are suppressed/filtered, the edges are less distinct in the DT-RF image. In [57] the authors argue that the DT-RF is best suited for propagating sparse information because it will spread the information through out the *entire* image as it is an IIR filter. While this is true, this property can arguable make the DT-RF filter *undesirable* for interpolation, particularly with depth maps. It is better for the depth maps to have strong edges as there is generally very little intra-object depth variation but large amounts of inter-object variation, particularly when it comes to the background.

Appendix C

List of Publications

The publications listed below were produced either directly in development of this dissertation or as work done in parallel with what has been presented here.

Conference Publications

- **R. Rzeszutek**, R. Phan, and D. Androutsos, "Depth estimation for semi-automatic 2d to 3d conversion," in Proceedings of the 20th ACM international conference on Multimedia, ser. MM '12. New York, NY, USA: ACM, 2012, pp. 817-820.
- **R. Rzeszutek** and D. Androutsos, "Efficient automatic depth estimation for video," Digital Signal Processing (DSP), 2013 18th International Conference on , vol., no., pp.1,6, 1-3 July 2013
- **R. Rzeszutek** and D. Androutsos, "Label propagation through edge-preserving filters," Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on, vol., no., pp.599,603, 4-9 May 2014.

Journal Publications

- **R. Rzeszutek** and D. Androutsos, "A Framework for Estimating Relative Depth in Video", *submitted to* Computer Vision and Image Understanding. Manuscript Number: CVIU-14-390.

Other Publications

- **R. Rzeszutek**, D. Tian and A. Vetro, "Disparity estimation of misaligned images in a scanline optimization framework," Acoustics, Speech and Signal Processing (ICASSP), 2013

IEEE International Conference on , vol., no., pp.1523,1527, 26-31 May 2013

- M. Fawaz, R. Phan, **R. Rzeszutek** and D. Androutsos, "Adaptive 2D to 3D image conversion using a hybrid Graph Cuts and Random Walks approach," Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on , vol., no., pp.1441,1444, 25-30 March 2012

- **R. Rzeszutek**, R. Phan, and D. Androutsos, "Semi-automatic synthetic depth map generation for video using random walks," in Multimedia and Expo (ICME), 2011 IEEE International Conference on, July 2011, pp. 16.

- R. Phan, **R. Rzeszutek**, and D. Androutsos, "Semi-automatic 2d to 3d image conversion using a hybrid random walks and graph cuts based approach," in Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, 2011, pp. 897900.

- R. Phan, **R. Rzeszutek**, and D. Androutsos, "Semi-automatic 2d to 3d image conversion using scale-space random walks and a graph cuts based depth prior," in Image Processing (ICIP), 2011 18th IEEE International Conference on, sept. 2011, pp. 865868.

- **R. Rzeszutek**, D. Androutsos, M. Kyan, "Self-Organizing Maps for Topic Trend Discovery", IEEE Signal Processing Letters, Vol. 17, No. 6, pp. 607-610. June 2010.

- **R. Rzeszutek**, T. El-Maraghi, D. Androutsos, S. Zhou, "An Advantageous Rotoscoping Method," Signal Processing Magazine, IEEE, vol.27, no.2, pp.34-39, March 2010.

Bibliography

[1] E. B. Goldstein, *Sensation and Perception*, 7th ed. Thomson Wadsworth, 2007, ch. 8, pp. 169–172.

[2] Y. Boykov and G. Funka-Lea, "Graph cuts and efficient n-d image segmentation," *International Journal of Computer Vision*, vol. 70, no. 2, pp. 109–131, 2006. [Online]. Available: http://dx.doi.org/10.1007/s11263-006-7934-5

[3] L. Grady, "Random walks for image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 11, pp. 1768–1783, 2006.

[4] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?" *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 2, pp. 147–159, 2004.

[5] R. Zabih and V. Kolmogorov, "Spatially coherent clustering using graph cuts," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, 2004, pp. II–437–II–444 Vol.2.

[6] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1, 2003, pp. I–195–I–202 vol.1.

[7] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, 2007, pp. 1–8.

[8] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 231–242. [Online]. Available: http://doi.acm.org/10.1145/280814.280882

[9] R. Rzeszutek, R. Phan, and D. Androutsos, "Semi-automatic synthetic depth map generation for video using random walks," in *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, july 2011, pp. 1–6.

[10] R. Phan, R. Rzeszutek, and D. Androutsos, "Semi-automatic 2d to 3d image conversion using a hybrid random walks and graph cuts based approach," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, 2011, pp. 897–900.

[11] ——, "Semi-automatic 2d to 3d image conversion using scale-space random walks and a graph cuts based depth prior," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*, sept. 2011, pp. 865 –868.

[12] J. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," Intel Corporation, Tech. Rep., 2001.

[13] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[14] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2007.09.014

[15] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 510–517.

[16] G. Zhang, Z. Dong, J. Jia, T.-T. Wong, and H. Bao, "Efficient non-consecutive feature tracking for structure-from-motion," in *Proceedings of the 11th European conference on Computer vision: Part V*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 422–435. [Online]. Available: http://dl.acm.org/citation.cfm?id=1888150.1888184

[17] K. Cordes, O. Müller, B. Rosenhahn, and J. Ostermann, "Feature trajectory retrieval with application to accurate structure and motion recovery," in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science, G. Bebis, R. Boyle, B. Parvin, D. Koracin, S. Wang, K. Kyungnam, B. Benes, K. Moreland, C. Borst, S. DiVerdi, C. Yi-Jen, and J. Ming, Eds. Springer Berlin Heidelberg, 2011, vol. 6938, pp. 156–167. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24028-7_15

[18] L. Torresani, A. Hertzmann, and C. Bregler, "Nonrigid structure-from-motion: Estimating shape and motion with hierarchical priors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 5, pp. 878–892, 2008.

[19] D. Sun, S. Roth, and M. Black, "Secrets of optical flow estimation and their principles," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, june 2010, pp. 2432 –2439.

[20] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002. [Online]. Available: http://dx.doi.org/10.1023/A%3A1014573219977

[21] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 2, pp. 504–511, 2013.

[22] M. Bleyer, C. Rhemann, and C. Rother, "Patchmatch stereo - stereo matching with slanted support windows," in *British Machine Vision Conference 2011*, 2011, pp. 1–11, vortrag: British Machine Vision Conference BMVC 2011, Dundee; 2011-08-29 – 2011-09-02. [Online]. Available: http://publik.tuwien.ac.at/files/PubDat_201949.pdf

[23] A. Klaus, M. Sormann, and K. Karner, "Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 3, 2006, pp. 15–18.

[24] J. Kim, V. Kolmogorov, and R. Zabih, "Visual correspondence using energy minimization and mutual information," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003, pp. 1033–1040 vol.2.

[25] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 2, pp. 328–341, 2008.

[26] M. Lang, O. Wang, T. Aydin, A. Smolic, and M. Gross, "Practical temporal consistency for image-based graphics applications," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 34:1–34:8, Jul. 2012. [Online]. Available: http://doi.acm.org/http://doi.acm.org/10.1145/2185520.2185530

[27] R. Hartley, "Theory and practice of projective rectification," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 115–127, 1999. [Online]. Available: http://dx.doi.org/10.1023/A%3A1008115206617

[28] A. Fusiello, E. Trucco, and A. Verri, "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, vol. 12, no. 1, pp. 16–22, 2000. [Online]. Available: http://dx.doi.org/10.1007/s001380050120

[29] A. Fusiello and L. Irsara, "Quasi-euclidean uncalibrated epipolar rectification," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, 2008, pp. 1–4.

[30] M. Pollefeys, R. Koch, and L. Van Gool, "A simple and efficient rectification method for general motion," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, 1999, pp. 496–501 vol.1.

[31] D. Oram, "Rectification for any epipolar geometry." in *BMVC*, vol. 1, 2001, pp. 653–662.

[32] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. Seitz, "Multi-view stereo for community photo collections," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, oct. 2007, pp. 1 –8.

[33] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multiview stereopsis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 8, pp. 1362–1376, Aug 2010.

[34] H. Jiang, H. Liu, P. Tan, G. Zhang, and H. Bao, "3d reconstruction of dynamic scenes with multiple handheld cameras," in *Computer Vision – ECCV 2012*, ser. Lecture Notes in Computer Science, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Springer Berlin Heidelberg, 2012, pp. 601–615. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33709-3_43

[35] M. Vergauwen and L. Van Gool, "Web-based 3d reconstruction service," *Mach. Vision Appl.*, vol. 17, no. 6, pp. 411–426, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1007/s00138-006-0027-1

[36] N. Snavely, I. Simon, M. Goesele, R. Szeliski, and S. Seitz, "Scene reconstruction and visualization from community photo collections," *Proceedings of the IEEE*, vol. 98, no. 8, pp. 1370 –1390, aug. 2010.

[37] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Generic and real-time structure from motion using local bundle adjustment," *Image Vision Comput.*, vol. 27, no. 8, pp. 1178–1193, Jul. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.imavis.2008.11.006

[38] S. A. Holmes and D. W. Murray, "Monocular slam with conditionally independent split mapping," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 6, pp. 1451–1463, 2013.

[39] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[40] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment — a modern synthesis," in *Vision Algorithms: Theory and Practice*, ser. Lecture Notes in Computer Science, B. Triggs, A. Zisserman, and R. Szeliski, Eds. Springer Berlin Heidelberg, 2000, vol. 1883, pp. 298–372. [Online]. Available: http://dx.doi.org/10.1007/3-540-44480-7_21

[41] C. Engels, H. Stewénius, and D. Nistér, "Bundle adjustment rules," in *In Photogrammetric Computer Vision*, 2006.

[42] B. Triggs, "Autocalibration and the absolute quadric," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, 1997, pp. 609–614.

[43] R. Gherardi and A. Fusiello, "Practical autocalibration," in *Proceedings of the 11th European conference on Computer vision: Part I*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 790–801. [Online]. Available: http://dl.acm.org/citation.cfm?id=1886063.1886123

[44] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch, "Visual modeling with a hand-held camera," *Int. J. Comput. Vision*, vol. 59, no. 3, pp. 207–232, Sep. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000025798.50602.3a

[45] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[46] ——, "Parallel tracking and mapping on a camera phone," in *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, Orlando, October 2009.

[47] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys, "Live metric 3d reconstruction on mobile phones," in *Computer Vision (ICCV), 2013 IEEE International Conference on*, Dec 2013, pp. 65–72.

[48] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3d," in *SIGGRAPH Conference Proceedings*. New York, NY, USA: ACM Press, 2006, pp. 835–846.

[49] ——, "Modeling the world from Internet photo collections," *International Journal of Computer Vision*, vol. 80, no. 2, pp. 189–210, November 2008. [Online]. Available: http://phototour.cs.washington.edu/

[50] S. Agarwal, N. Snavely, I. Simon, S. Seitz, and R. Szeliski, "Building rome in a day," in *Computer Vision, 2009 IEEE 12th International Conference on*, Sept 2009, pp. 72–79.

[51] G. Zhang, X. Qin, W. Hua, T.-T. Wong, P.-A. Heng, and H. Bao, "Robust metric reconstruction from challenging video sequences," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, 2007, pp. 1–8.

[52] P. F. Sturm and B. Triggs, "A factorization based algorithm for multi-image projective structure and motion," in *Proceedings of the 4th European Conference on Computer Vision-Volume II - Volume II*, ser. ECCV '96. London, UK, UK: Springer-Verlag, 1996, pp. 709–720. [Online]. Available: http://dl.acm.org/citation.cfm?id=645310.649025

[53] B. Triggs, "Factorization methods for projective structure and motion," in *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, jun 1996, pp. 845 –851.

[54] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*, 1998, pp. 839–846.

[55] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Int. J. Comput. Vision*, vol. 81, no. 1, pp. 24–52, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1007/s11263-007-0110-8

[56] R. Fattal, "Edge-avoiding wavelets and their applications," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–10, 2009.

[57] E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM TOG*, vol. 30, no. 4, pp. 69:1–69:12, 2011, proceedings of SIGGRAPH 2011.

[58] K. He, J. Sun, and X. Tang, "Guided image filtering," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 6, pp. 1397–1409, 2013.

[59] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004. [Online]. Available: http://dx.doi.org/10.1023/B%3AVISI.0000013087.49260.fb

[60] E. Bosc, R. Pepion, P. Le Callet, M. Koppel, P. Ndjiki-Nya, M. Pressigout, and L. Morin, "Towards a new quality metric for 3-d synthesized view assessment," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 5, no. 7, pp. 1332–1343, 2011.

[61] M. Guttmann, L. Wolf, and D. Cohen-Or, "Semi-automatic Stereo Extraction from Video Footage," *Proc. IEEE Intl. Conf. on Computer Vision (ICCV)*, 2009.

[62] W. Tam, C. Vázquez, and F. Speranza, "Three-dimensional TV: A novel method for generating surrogate depth maps using colour information," in *Proceedings of SPIE, the International Society for Optical Engineering*. Society of Photo-Optical Instrumentation Engineers, 2009.

[63] L. J. Angot, W.-J. Huang, and K.-C. Liu, "A 2D to 3D Video and Image Conversion Technique based on a Bilteral Filter," *Proc. SPIE Electronic Imaging - Three-Dimensional Image Processing (3DIP) and Applications*, 2010.

[64] A. Saxena, M. Sun, and A. Ng, "Make3d: Learning 3d scene structure from a single still image," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 5, pp. 824 –840, may 2009.

[65] Z. Zhang, Y. Wang, T. Jiang, and W. Gao, "Stereoscopic learning for disparity estimation," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, may 2011, pp. 365 –368.

[66] K. Karsch, C. Liu, and S. Kang, "Depthtransfer: Depth extraction from video using non-parametric sampling," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[67] J. Konrad, M. Wang, P. Ishwar, C. Wu, and D. Mukherjee, "Learning-based, automatic 2d-to-3d image and video conversion," *Image Processing, IEEE Transactions on*, vol. 22, no. 9, pp. 3485–3496, 2013.

[68] H. Tian, B. Zhuang, Y. Hua, Y. Zhao, and A. Cai, "Recovering depth of background and foreground from a monocular video with camera motion," in *Visual Communications and Image Processing (VCIP), 2013*, Nov 2013, pp. 1–6.

[69] S. Knorr, M. Kunter, and T. Sikora, "Stereoscopic 3d from 2d video with super-resolution capability," *Image Commun.*, vol. 23, no. 9, pp. 665–676, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.image.2008.07.004

[70] G. Zhang, W. Hua, X. Qin, T.-T. Wong, and H. Bao, "Stereoscopic video synthesis from a monocular video," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 4, pp. 686–696, 2007.

[71] G. Zhang, J. Jia, T.-T. Wong, and H. Bao, "Consistent Depth Maps Recovery from a Video Sequence," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 31, no. 6, pp. 974–988, June 2009.

[72] G. Zhang, Z. Dong, J. Jia, L. Wan, T.-T. Wong, and H. Bao, "Refilming with depth-inferred videos," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 5, pp. 828–840, 2009.

[73] O. Wang, M. Lang, M. Frei, A. Hornung, A. Smolic, and M. Gross, "Stereobrush: interactive 2d to 3d conversion using discontinuous warps," in *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, ser. SBIM '11. New York, NY, USA: ACM, 2011, pp. 47–54. [Online]. Available: http://doi.acm.org/10.1145/2021164.2021173

[74] R. Phan and D. Androutsos, "Robust semi-automatic depth map generation in unconstrained images and video sequences for 2d to stereoscopic 3d conversion," *Multimedia, IEEE Transactions on*, vol. 16, no. 1, pp. 122–136, 2014.

[75] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 689–694, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015780

[76] B. Ward, S. B. Kang, and E. Bennett, "Depth director: A system for adding depth to movies," *Computer Graphics and Applications, IEEE*, vol. 31, no. 1, pp. 36 –48, jan.-feb. 2011.

[77] M. Liao, J. Gao, R. Yang, and M. Gong, "Video stereolization: Combining motion analysis with user interaction," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 7, pp. 1079–1088, July 2012.

[78] M. Becker, M. Baron, D. Kondermann, M. Bussler, and V. Helzle, "Movie dimensionalization via sparse user annotations," in *3DTV-Conference: The True Vision-Capture, Transmission and Dispaly of 3D Video (3DTV-CON), 2013*, Oct 2013, pp. 1–4.

[79] R. Szeliski, *Computer vision: algorithms and applications*. Springer, 2010.

[80] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*. McGraw-Hill Inc., 1995.

[81] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, 1994, pp. 593–600.

[82] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*, vol. 1, May 2006, pp. 430–443. [Online]. Available: http://edwardrosten.com/work/rosten_2006_machine.pdf

[83] P. Chen and D. Suter, "Simultaneously estimating the fundamental matrix and homographies," *Robotics, IEEE Transactions on*, vol. 25, no. 6, pp. 1425–1431, 2009.

[84] R. Hartley and H. Li, "An efficient hidden variable approach to minimal-case camera motion estimation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 12, pp. 2303–2314, 2012.

[85] D. Nister, "An efficient solution to the five-point relative pose problem," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 6, pp. 756–770, 2004.

[86] H. Stewénius, C. Engels, and D. Nistér, "Recent developments on direct relative orientation," {*ISPRS*} *Journal of Photogrammetry and Remote Sensing*, vol. 60, no. 4, pp. 284 – 294, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092427160600030X

[87] P. Torr and D. Murray, "The development and comparison of robust methods for estimating the fundamental matrix," *International Journal of Computer Vision*, vol. 24, no. 3, pp. 271–300, 1997. [Online]. Available: http://dx.doi.org/10.1023/A%3A1007927408552

[88] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[89] P. Torr, "An assessment of information criteria for motion model selection," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, 1997, pp. 47–52.

[90] P. Torr, A. Fitzgibbon, and A. Zisserman, "Maintaining multiple motion model hypotheses over many views to recover matching and structure," in *Computer Vision, 1998. Sixth International Conference on*, 1998, pp. 485–491.

[91] ——, "The problem of degeneracy in structure and motion recovery from uncalibrated image sequences," Microsoft Research, Tech. Rep. MSR-TR-99-03, March 1999.

[92] J. Repko and M. Pollefeys, "3d models from extended uncalibrated video sequences: addressing key-frame selection and projective drift," in *3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on*, 2005, pp. 150–157.

[93] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 12, no. 7, pp. 629–639, Jul 1990.

[94] C. Couprie, L. Grady, L. Najman, and H. Talbot, "Power watershed: A unifying graph-based optimization framework," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 7, pp. 1384–1399, July 2011.

[95] D. Barash, "Fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 6, pp. 844–847, Jun 2002.

[96] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM National Conference*, ser. ACM '68. New York, NY, USA: ACM, 1968, pp. 517–524. [Online]. Available: http://doi.acm.org/10.1145/800186.810616

[97] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[98] R. Phan and D. Androutsos, "Edge-aware temporally consistent simpleflow: Optical flow without global optimization," in *Digital Signal Processing (DSP), 2013 18th International Conference on*, 2013, pp. 1–6.

[99] A. M. Oded Maron, "Hoeffding races: Accelerating model selection search for classification and function approximation," in *Advances in Neural Information Processing Systems*, G. T. . J. A. Jack D. Cowan, Ed., vol. 6. 340 Pine Street, 6th Fl., San Francisco, CA 94104: Morgan Kaufmann, April 1994, pp. 59–66.

[100] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, S. Dasgupta and D. Mcallester, Eds., vol. 28, no. 2. JMLR Workshop and Conference Proceedings, May 2013, pp. 199–207. [Online]. Available: http://jmlr.csail.mit.edu/proceedings/papers/v28/bardenet13.pdf

[101] D. Herrera C., J. Kannala, and J. Heikkilä, "Joint depth and color camera calibration with distortion correction," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 10, pp. 2058–2064, Oct 2012.

[102] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603–619, May 2002.

[103] P. Sand and S. Teller, "Particle video: Long-range motion estimation using point trajectories," *International Journal of Computer Vision*, vol. 80, pp. 72–91, 2008, 10.1007/s11263-008-0136-6. [Online]. Available: http://dx.doi.org/10.1007/s11263-008-0136-6

[104] R. Rzeszutek, R. Phan, and D. Androutsos, "Depth estimation for semi-automatic 2d to 3d conversion," in *Proceedings of the 20th ACM international conference on Multimedia*, ser. MM '12. New York, NY, USA: ACM, 2012, pp. 817–820. [Online]. Available: http://doi.acm.org/10.1145/2393347.2396320

[105] M. W. Tao, J. Bai, P. Kohli, and S. Paris, "Simpleflow: A non-iterative, sublinear optical flow algorithm," *Computer Graphics Forum (Eurographics 2012)*, vol. 31, no. 2, May 2012. [Online]. Available: http://graphics.berkeley.edu/papers/Tao-SAN-2012-05/

[106] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conf. on Computer Vision (ECCV)*, ser. Part IV, LNCS 7577, A. Fitzgibbon et al. (Eds.), Ed. Springer-Verlag, Oct. 2012, pp. 611–625.

[107] T. Moons, M. Vergauwen, and L. V. Gool, *3D reconstruction from multiple images*, International Computer Vision Summer School (ICVSS), July 2008, lecture.

[108] P. Torr and A. Zisserman, "MLESAC: A new robust estimator with application to estimating image geometry," *Computer Vision and Image Understanding*, vol. 78, pp. 138–156, 2000.

Acronyms

Notation	Description
DBIR	Depth-based Image Rendering. 16, 18
DT	Domain Transform. 16, 55, 56, 63, 80, 82, 106, 107, 110
DT-IC	Interpolated Convolution. 56, 109, 110
DT-NC	Normalized Convolution. ix, 55–57, 59, 60, 62–64, 74, 75, 109, 110, 112
DT-RF	Recursive Form. ix, 56, 57, 62, 63, 109, 110, 112
GIF	Guided Image Filter. 55
GRIC	Geometrically Robust Information Criterion. 42, 103–105
IDW	Inverse Distance Weighting. 48, 51
KLT	Kanade-Lucas-Tomasi. 14, 33–35, 70, 80, 101
MLE	Maximum Likelihood Estimate. 104
MRF	Markov Random Fields. 47
MVS	Multiview Stereo. 9, 12, 17, 78, 85, 100, 101
PTAM	Parallel Tracking and Mapping. 13, 14
RANSAC	Random Sample Consensus. 41, 42, 101, 104
SfM	Structure from Motion. 8, 9, 12–15, 17, 18, 33, 42, 43, 78, 79, 84, 85, 89, 100–102
SIFT	Scale-Invariant Feature Transform. 14, 70
SURF	Speeded-Up Robust Features. ix, 14, 38, 39