REAL-TIME 3D RECONSTRUCTION AND RENDERING FOR MINIMALLY INVASIVE SURGERY WITH MULTI CAMERAS ARRAY

by

Jianwei Ke

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2024

Date of final oral examination: 02/09/2024

The dissertation is approved by the following members of the Final Oral Committee:
Yu Hen Hu, Professor, Electrical and Computer Engineering
Hongrui Jiang, Professor, Electrical and Computer Engineering
John Gubner, Professor, Electrical and Computer Engineering
Yin Li, Assistant Professor, Biostatistics and Medical Informatics
William Sethares, Professor, Electrical and Computer Engineering

Acknowledgments

This endeavor would not have come to fruition without the invaluable assistance of numerous remarkable individuals over the years.

Foremost, I extend my deepest appreciation to my advisor, Professor Yu Hen Hu, for his unwavering support. It has been a sincere honor to be part of his lab and collaborate with him. His insightful guidance has played a pivotal role in shaping me as a student, researcher, and, above all, as a better human being. Gratitude is also extended to my co-advisor, Professor Hongrui Jiang, and his team, with whom working on this project has been a delightful and honorable experience.

I express my thanks to my committee members, Professor John Gubner, Professor William Sethares, and Professor Yin Li, for their valuable suggestions and contributions that have significantly influenced the course of this dissertation. Special acknowledgment is due to my numerous colleagues, whose names, regrettably, I cannot list here.

My heartfelt thanks go to my parents, Mingwei Huang and Jin Ke, for their unconditional love. I am equally indebted to my brother, Jianlong Ke, and sister, Yicui Ke, who have consistently stood by me, for their unwavering support, encouragement, and love, which have consistently accompanied me throughout this journey.

Contents

List of Tables

List of Figures

Abstract

Advancements in digital camera technologies have led to significantly smaller camera sensors without compromising imaging quality. This trend opens avenues for replacing single-camera systems with multi-camera arrays, offering benefits such as an expanded field-of-view and 3D information. However, camera arrays require additional processing to interpret the collected image data. This report addresses two significant algorithmic challenges in developing a camera array.

The first challenge involves virtual view generation, which is the task of estimating the image for a virtual camera using image data from multiview cameras. Ideally, if the geometry of the underlying scene is known, the virtual view can be generated by projecting the 3D geometry to the virtual camera. However, finding the 3D geometry of a scene from multiview cameras is an active research topic in Computer Vision. Considering the real-time processing requirement (<33 ms), we propose a real-time 3D visualization (RT3DV) system using a multiview RGB camera array that can process multiple synchronized video streams to produce a stereo video of a dynamic scene from a chosen view angle. We implemented a proof of concept RT3DV system tasked to process five synchronous video streams acquired by an RGB camera array. It achieves a processing speed of 44 milliseconds per frame and a peak signal-to-noise ratio (PSNR) of 15.9 dB from a viewpoint coinciding with a reference view. As a comparison, an image-based MVS algorithm will require 7 seconds to render a frame and yield a reference view PSNR of 16.3 dB.

The second challenge is to stabilize a video at a real-time rate. We propose LSstab, a novel algorithm that efficiently suppresses unwanted motion jitters in real-time. LSstab features a parallel realization of the *a-contrario* RANSAC (AC-RANSAC) algorithm to estimate the inter-frame camera motion parameters. A novel least-squares smoothing cost function is proposed to mitigate undesirable camera

jitters. A recursive least square solver is then derived to minimize the smoothing cost function with a linear computation complexity. Evaluation against state-of-the-art video stabilization methods using publicly available videos demonstrates that LSstab achieves comparable or superior performance, particularly attaining real-time processing speed with GPU utilization.

CHAPTER 1
Introduction

It was not that long ago owing a camera was a luxury. Nowadays, anyone with a cell phone can take pictures with ease, and millions, if not, more images are shared over the internet every day. Cameras have become ubiquitous, affecting almost every aspect of our daily lives. Unmanned vehicles employ cameras to permit us to explore hazardous areas and execute complicated tasks safely and efficiently. Surgical cameras allow surgery to be performed efficiently and safely while maintaining minimum invasive. In all these applications, cameras are used as an imaging device directly providing the 2D projection of the 3D environment. However, a pair of dissimilar images of the 3D scene must be created for each of our eyes to facilitate the perception of depth, which is a mission impossible for a single-camera system. With two or more cameras viewing the same scene, we begin to be capable of reconstructing the depth information that is critical for depth perception and was lost during the projection process.

As digital camera sensors develop rapidly, the miniaturization of camera sensors has made replacing single-camera systems with camera arrays possible. A camera array consists of multiple cameras mounted on a rig, simultaneously capturing videos of the same scene from different perspectives. This multiview property of a camera array has enabled the possibility of reconstructing the depth information of a scene, which is the key to solving challenging tasks that could not be accomplished easily with a single camera system. For example, once the 3D model has been computed, the corresponding image can be created by projecting the 3D model to the desired virtual view.

However, depth reconstruction from multiview cameras requires the relative camera pose to be known. If the individual cameras are not firmly mounted on

the rig, we will have to consistently calibrate the cameras. Hence, the individual cameras should be stable relative to each other. Moreover, a majority of videos are created by hand-held cameras. As such, many video clips suffer from motion jitters due to inevitable handshaking, and the visual quality can be tremendously enhanced if videos are stabilized.

Many applications such as drone surveillance and laparoscopic surgery have strictly real-time requirements. Switching to camera arrays, the real-time requirement should also be satisfied, but it is more difficult because more images are needed to process within the real-time rate, i.e., 33 ms.

1.1 Contributions

There are several fundamental research challenges to the problem identified above: (1) generating virtual views from camera arrays in real-time, (2) accurately reconstructing the surgical tools maneuvered by surgeons, and (3) stabilizing video frames in real-time. In my thesis, I will present new algorithms and techniques in computer vision that address these challenges.

1.1.1 Real-time 3D visualization

A multiview camera array contains multiple cameras mounted on a rigid rig, capturing videos in a synchronous manner. By properly arranging camera orientations, one may estimate a dynamic 3D model of the foreground objects from the component videos and synthesize a new video of stereo vision of these objects from a desired view angle. A real-time 3D visualization (RT3DV) system consisting of a camera array and a processing platform will synthesize the stereo video when the camera array is capturing the dynamic scene.

An RT3DV system is very similar to the multi-view stereo (MVS) algorithm [3, 4, 5, 1, 6] in that a 3D surface model will be estimated based on multiple images (video frames). However, the primary design objective of current MVS systems is to accurately reconstruct a 3D surface model of a foreground object. As such,

a dense point cloud 3D model [3, 4, 5, 1, 6] often needs to be estimated based on time-consuming optimization procedures. Hence, many of the traditional MVS pipeline modules may not be cost-effective. RT3DV, on the other hand, is developed to output a stereo video from a given viewing direction in real-time. A 3D model is needed here only to provide depth sensation of a stereo vision from a given view orientation.

In developing the RT3DV system, we focus on developing low complexity 3D surface model estimation and update methods such that the resulting algorithm may be executed on a commodity computing platform at a video frame rate (real-time). The low-complexity 3D surface model contains very few triangular meshes, formed with a sparse set of 3D key points. Periodically, the coordinates of these 3D key points are estimated from matching 2D key points in temporally synchronized video frames of multiple video streams via epipolar geometry and trifocal tensor. Between successive periodic refreshing of 2D key points detection and matching, their 2D positions are updated in intermediate frames using visual feature tracking. These innovative 3D modeling approaches reduce computation time per video frame by orders of magnitudes. Given the low-complexity 3D surface model and a desired view point, a stereo view may be rendered by mapping the texture of each triangular surface from the corresponding triangle in the closest view. Since the objective of RT3DV is 3D visualization, the image quality of the rendered video frame will be evaluated rather than the accuracy of the 3D coordinates of the 3D surface model.

We built a proof-of-concept camera array [7, 8, 9] and developed the RT3DV algorithm on this platform. In Fig. 1.1(a), five miniature cameras form a camera array acquiring videos synchronously. In Fig. 1.1(b), three objects used in the experiment are displayed. This camera array is mounted on the top of a surgical training box, as shown from the side in Fig. 1.1(c). The electronics for transmitting the multiview video streams to a close-by laptop are shown in Fig. 1.1(d). For this kind of embedded system application, trade-offs between the image quality of rendered 3D stereo video and corresponding computation costs become very important.

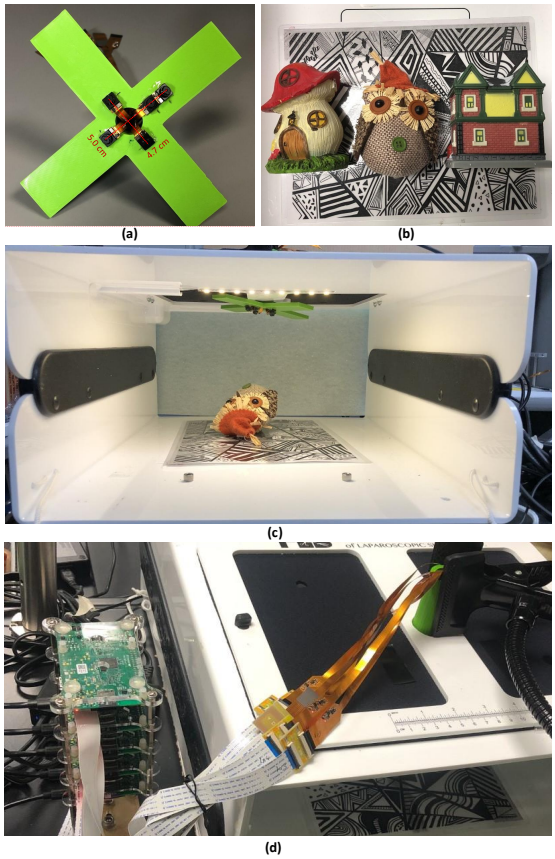


Figure 1.1: (a)Micro-camera Array used to collect data (b)(c) Objects to reconstruct are placed in a Fundamentals of Laparoscopic Surgery (FLS) laparoscope trainer box while the camera array is recording. (d) Each Pi camera is connected to a Raspberry Pi.

The technical contribution of the RT3DV system is the development of a low-complexity 3D reconstruction algorithm for 3D visualization of dynamic scenes. It does not require an expensive depth sensor (e.g., Kinect) and can be port to internet of things (IoT) devices.

1.1.2 Real-time Moving Surgical Tool Reconstruction

In a typical laparoscopic surgery setting, the surgical tools, specifically graspers, interact against a background objects that remains relatively stable between suc-

cessive frames. These graspers play a pivotal role in surgical procedures, enabling precise maneuvers that result in deliberate and controlled movements of tissues and organs within human interior body. The accurate perception of the 3D states of these surgical tools during operation is of utmost importance. Hence, the graspers demand special attention due to their central role in the surgeon's focus. A laparoscopic grasper, depicted in Fig. 4.1, is characterized by a semi-rigid structure consisting of a rod, two finger tips, and a joint point.

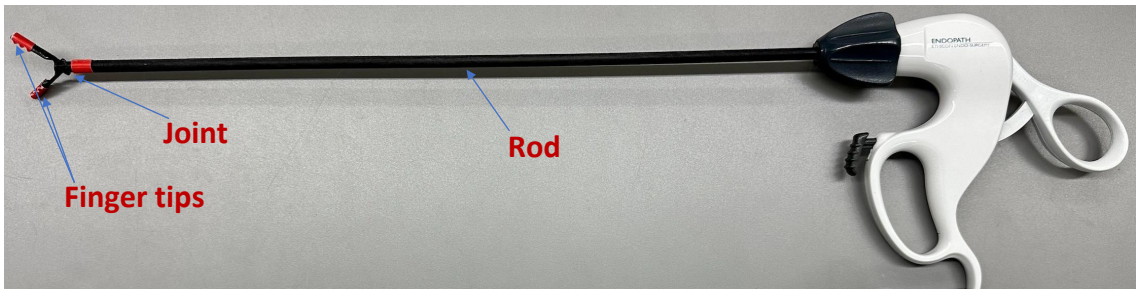


Figure 1.2: The surgical tool for a laparoscopic surgery, e.g. a grasper, consist of a rod, two finger tips and a joint point.

The primary objective is to efficiently and precisely reconstruct the 3D pose of the laparoscopic grasper through the utilization of synchronized multiview video streams. Our assumption is that the laparoscopic environment maintains a static background, with two graspers serving as the surgical tools tasked with manipulating beans within a laparoscopic surgery training box. The foreground objects encompass rigid beans and the two graspers, simulating the tools in action, while the background scene incorporates one or more cups to introduce depth variations within the interior body, as shown in Fig. 4.2. To enhance the reconstruction and rendering of these textureless foreground objects, we introduce color markers for identifying key components and pre-established 3D models for generating virtual views.

The proposed algorithm framework initiates with an offline calibration stage, where we determine camera poses, obtain dense 3D models for background objects, and establish a pre-built 3D model for the foreground object. In acquiring camera

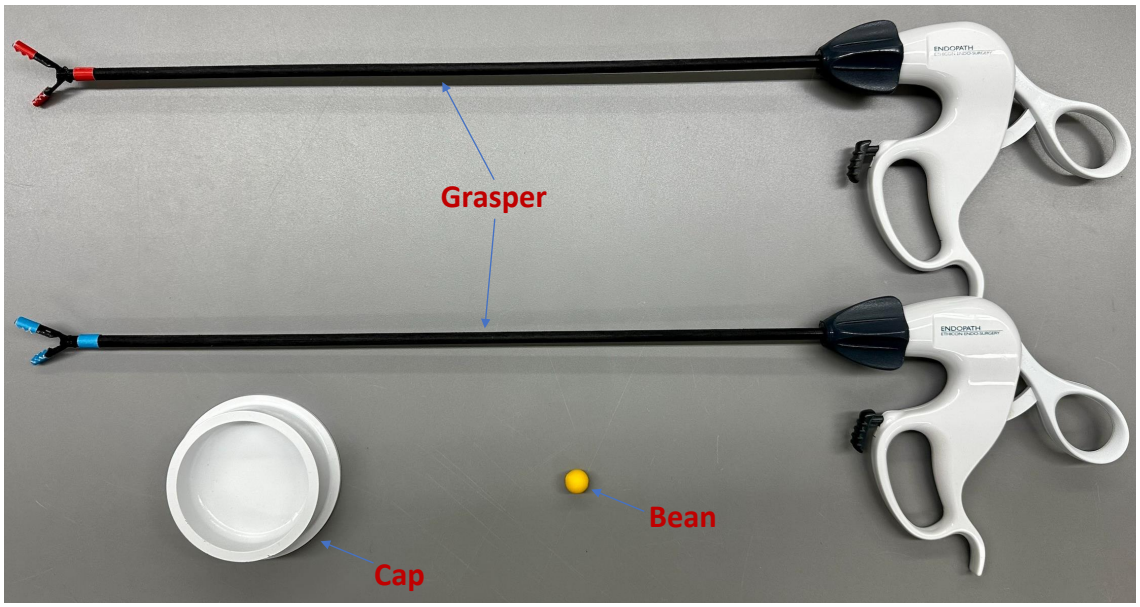


Figure 1.3: Foreground and background objects

poses and dense 3D models for background objects, a structured light with a random stride pattern is shined to backgrounds. Subsequently, Structure from Motion (SfM) [10] is applied to derive camera poses. Next, Multi-View Stereo (MVS)[1] is utilized to generate a dense 3D model for background objects. We employ SolidWorks to acquire the 3D models of the grasper and beans.

In the reconstruction of the 3D pose for foreground objects, specifically the graspers and beans, our approach involves initial tracking of the two finger tips, the joint point, and the direction of the rod in each 2D view. We use the Kalman filter to mitigate random jitters in 2D tracking. Subsequently, we establish correspondences for the finger tips, joint point, and rod directions. Once these correspondences are established, real-time calculation of the corresponding 3D poses is performed using a combination of Direct Linear Transformation (DLT), as outlined in [11], and the Kalman filter.

1.1.3 Real-time online video stabilization

A video stabilization algorithm can be helpful in many applications, such as visual tracking [12, 13], video surveillance [14], and wearable cameras [15]. If the video stabilization algorithm is online and in real-time, it is also beneficial to minimally invasive surgery [16, 17, 18], unmanned aerial vehicles [19, 20], etc.

Video stabilization is to reduce annoying jitter in the captured video due to unwanted or uncontrolled camera shake during video capturing [21, 22, 23]. Hardware video stabilizers have been developed to mitigate the physical shaking of the camera [22, 24] while shooting the video. Video stabilization algorithms [21, 25, 26, 24, 27, 22, 28] may also be applied to post-process a captured video to produce a stabilized video that exhibits smoother global camera motion. Both the hardware and software solutions can be combined to ensure desired video quality.

Algorithmically the process of video stabilization consists of the following steps: (a) Estimate global camera motion trajectory in a given video clip; (b) Choose a targeted (usually smoothed) camera motion trajectory; (c) Modify individual video frames according to the targeted camera motion trajectory; (d) apply additional post-processing steps to mitigate potential motion blurs due to (original) camera jitters and irregular frame boundaries due to geometrical transformation applied to realize the desired camera motion trajectory.

Depending on the context of the video, the global camera motion may not be easily defined. If the video consists of a rapidly moving foreground object, one may want to track the foreground object to maintain its position at the center of the video frame. The camera trajectory should be estimated from the tracked foreground object in this case. On the other hand, for surveillance purposes, the desired camera motion may be stationary or smooth panning of the camera. In this case, the background may be used to determine the camera's global motion trajectory. Thus, an ideal video stabilization algorithm must allow human input to estimate the global camera motion and appropriately determine the targeted global camera motion trajectory. Currently, almost all video stabilization algorithms directly use static background objects to estimate the desired camera's global motion.

[25, 21] and [19, 26, 20, 24, 22, 28] compute optical flow and feature points and then use RANSAC [29] to find a subset of feature points belonging to the background. Liu et al. [22, 28] use Structure-from-Motion (SfM) to reconstruct the 3D camera trajectory and a sparse 3D point cloud, where RANSAC is used to select a subset of background feature points to derive the camera motions.

The targeted global motion trajectory is also affected by the length of the video clips. Video processing may be applied in a batch mode, where the entire video is to be processed at once, or in an online real-time mode, where the stabilization may be applied incrementally for each additional short video clip which may be a single frame. The targeted motion trajectory can be regarded as a smoothed trajectory of the original trajectory. How "smooth" the targeted trajectory is relative to the original motion trajectory is yet another hyper-parameter that may need to be fine-tuned based on the outcome of a chosen view quality metric. Grundmann et al. [24] and Liu et al. [22] state that a desirable stable camera path should follow the cinematography principles. In other words, the desired motion path should be composed of constant, linear, and parabolic segments as if the video is taken with professional stabilization tools.

Motion smoothing is particularly challenging in the online real-time mode if the unseen future motion trajectory cannot be reliably predicted based on prior knowledge about the global camera motion. Successful video stabilization algorithms [21, 25, 26, 24, 27, 22, 28] first compute the entire motion trajectory either in 2D or 3D then smooth it at once. Several real-time approaches [25, 30] adapt Kalman filters for smoothing, but their performance is limited.

Given the current and desired global motion trajectories, a 2D or 3D geometrical transformation will be applied to each video frame to obtain the final stabilized frame. During this process, the quality of the output video frame may be impacted, and additional mitigating measures may be applied. These may include the correction of motion blurs due to rapid camera jitters and undefined frame boundaries due to the geometrical transformation of the frame images. Several post-processing techniques such as cropping [24, 22, 27] mosaicing [31], and inpainting [21] are proposed to mitigate this issue, which can be considered as a supplement to the

standard video stabilization procedure.

This paper proposes an online real-time geometry transformation-based algorithm for video stabilization. Unlike other geometry transformation-based algorithms, we adapt and parallelize a new technique called *a-contrario* RANSAC (AC-RANSAC), which does not require any hard thresholds for inlier/outlier discrimination appearing in RANSAC. Hence, it can compute inter-frame global motions more robustly.

The proposed algorithm starts by estimating the inter-frame global motion between two consecutive frames except for the first one. First, features points for the current frame are extracted and matched over the previous frame. We choose SURF [32] and FLANN [33] as our feature extraction and matching algorithm because of their delicate balance of robustness and efficiency [34, 35]. Then, the parallel AC-RANSAC is used to estimate the inter-frame geometry transformation, from which the motion parameters (translations, rotations, and scales) are derived. The inter-frame global motion estimation is performed for each incoming frame.

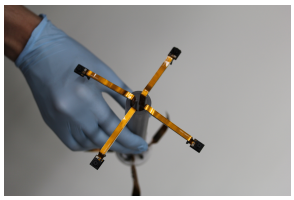
Then the algorithm enters the motion smoothing phase, which requires a user-specified parameter N. If the current frame number is less than N, the camera motion for the current frame will be smoothed by our cinematography principles guided modified recursive least squares algorithm (C-MRLS). After the N^{th} frame, we stabilize the current camera motion with our modified sliding window least squares algorithm (MSWLS). Finally, the current frame is warped to the previous stable space in the motion compensation stage to create a stabilized video sequence. The whole algorithm pipeline is shown in Fig. 5.1

The key contribution of the proposed algorithm is a novel least-squares-based smoothing cost for estimating the intentional motion and its associating solver that minimizes the cost in linear time.

1.2 Motivation

Minimally invasive surgery, in particular laparoscopic surgery, has become the standard for many general surgeries. It is now the gold standard approach for many

(**a**) Laparoscopic surgery



(**b**) Trocar Camera Array

Figure 1.4: (a)One single camera (laparoscope) and surgical instruments are inserted through multiple ports. An operator holds and navigates the laparoscopic surgery during the entire procedure. (b)A trocar camera array (TCA) consists of several cameras built into a surgical trocar.

abdominal operations, as it can offer significantly less recovery time for patients and fewer long-term consequences for the surgery. One single camera (laparoscope) and surgical tools are inserted through multiple ports in laparoscopic surgery, as shown in Figure 1.4 (a). An assistant is dedicated to navigating the camera through the surgical scene during the entire procedure.

Despite the progress in the techniques and instruments of minimally invasive surgery, the visualization system used by these procedures provides no depth perception and is subject to unwanted motion jitters due to inevitable handshaking. These deficiencies tend to increase operating time and the possibility of accidental injuries. Surgeons and assistants must rely on extensive training to gain sufficient spatial awareness of the surgical area and to maintain smooth communication.

We have built a prototype of a trocar camera array consisting of several cameras with different viewing positions and angles, as shown in Figure 1.4 (b). With such a multiview camera array, a fine 3D model of the surgical scene can be reconstructed and used to generate a pair of stereo images for a wide range of view angles. If stereo image pairs are generated in real-time, then an immersive and panoramic 3D view of the surgical area can be provided during the surgery in real-time. It is shown that panoramic 3D views reduce operating time and blood loss, especially for novice surgeons.

During surgery, the movement of surgical tools through the ports can cause unwanted camera jitters on the camera array, giving rise to video blur and limiting

the camera array's utility. Moreover, a shaky near-field video can easily cause dizziness and discomfort. An immediate and effective solution is to develop a real-time scene stabilization algorithm that removes the camera jitter and deblur the video.

Our goal is to develop a real-time 3D visualization system that behaves the same as an open window into the surgical cavity. The ideal result is to augment the generated 3D result on top of the patient's body so that the maneuver of surgical instruments behave the same as in open cavity surgery. Delivering such a visualization system is a difficult task. However, it is essential to bear the ideal in mind while we are approaching the end goal.

CHAPTER 2
Related Work

2.1 Image-based 3D Reconstruction

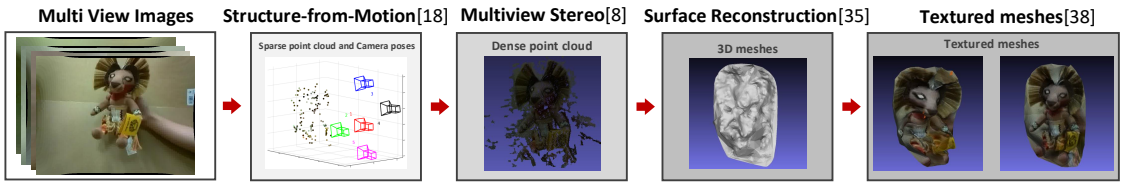


Figure 2.1: Baseline image-based 3D reconstruction pipeline

A typical image-based 3D reconstruction is the process of rebuilding the 3D shape of the original scene captured by multiview images. Existing 3D reconstruction algorithms aim to estimate a dense point cloud 3D representation of the scene. The objective is to enhance details of the estimated 3D surface model while conforming to visibility evidence. Computation complexity and computing time are of lesser concern. In this work, a typical image-based 3D reconstruction pipeline will be implemented as a baseline algorithm, of which the performance will be compared against our proposed algorithm.

The typical image-based 3D reconstruction pipeline consists of 3 steps: (a) 3D dense point cloud estimation, (b) 3D surface reconstruction, and (c) texture mapping. Before applying these three steps, a set of feature points (keypoints) will be detected at each view (image) using a feature detection algorithm such as SIFT [36], FAST[37], or SURF[32], etc. Then, a feature matching algorithm and RANSAC will be applied to jointly calibrate the camera intrinsic parameters as

well as camera extrinsic parameters relative to a reference world coordinate. The baseline image-based 3D reconstruction pipeline is summarized in Fig. 2.1.

2.1.1 Multiview Stereo (3D Dense Point Cloud Reconstruction)

Furukawa et al. [3] proposed a point-growing multiview stereo algorithm (PMVS) that iteratively grows the point cloud by adding new feature points according to the epipolar geometry while not violating visibility constraints. Bleyer et al.[38] introduced the PatchMatch algorithm[4, 39] for stereo matching. Initially, each pixel is assigned to a 3D plane randomly. A good plane that reduces a cost function will be propagated diagonally to neighboring pixels in an iterative manner. This PatchMatch algorithm has been adopted and extended in other works [4, 39, 5, 1, 6]. Shen[4] employs PatchMatch stereo[38] for multiview reconstruction to generate a depth map for each image and imposes depth consistency over neighboring images. Based on the PatchMatch propagation scheme, Zheng et al. [5] propose a probabilistic graphical model for jointly view selection and depth estimation for each pixel without considering slanted 3D planes. Galliani et al. develop *Gipuma*[1] in which they use a diffusion-like propagation scheme to efficiently propagate good planes to half the amount of pixels at once, which utilizes the parallel computation of Graphic Process Unit (GPU). Xu et al. [6] adopt an asymmetric checkerboard propagation scheme based on the confidence of current neighbor hypotheses and jointly selects a subset of views for cost aggregation.

2.1.2 3D Surface Reconstruction from Dense Point Clouds

When the dense point cloud is estimated using the PatchMatch method [40], the surface reconstruction problem may be posed as an energy minimization problem using the Delaunay triangulation. The energy cost function measures the agreement of inside/outside labeling of Delaunay tetrahedra based on the visibility constraints. A globally optimal tetrahedra labeling can be obtained by solving a graph S-T minimum cut problem. The method described in [40], however, assumes a strong geometrical prior and may fail for weakly-supported surfaces well.

Improvements were proposed in [41] and [41] which yield a more complete 3D surface at additional computation cost.

2.1.3 Texture Mapping

Texture mapping [42, 43, 44] is the process of painting the triangular surface mesh with realistic color, texture, and shade using images acquired from one or more appropriated cameras. The selection of camera(s) for texture mapping is formulated as a multi-label Markov random filed energy optimization problem. Each 3D triangular mesh will be assigned to a close-by view so that its appearance can be warped from a triangular area in the video frame with matching vertices. In [42], the selection criterion is to align the surface normal of the triangular mesh to the optical axis of the view. A global color adjustment and a local Poisson editing are applied to minimize the seam line along the boundary of the triangular mesh. In [43], instead of one view (camera), the corresponding 2D triangular regions in multiple views (cameras) are collected and blended to yield the final texture of the mesh. It reduces the blurring and ghosting artifacts due to blending but cannot mitigate texture bleeding due to geometric registration error and camera calibration error. In [44], post-processing efforts are introduced to ensure color consistency and geometry consistency of textures in adjacent surface meshes.

2.2 Free-Viewpoint Video

Free-viewpoint video (FVV), a.k.a 4D video[45, 46] refers to a 3D video service that allows viewers to choose their preferred viewing angles freely. A 4D video, represented by a 3D surface model, associated texture maps, and the evolution of this 3D model over time (hence 4D), is generated to achieve this goal.

The MVS algorithm is the basis of FVV for developing and updating the 3D surface model. In [47], an initial dense correspondence is established to compute depth for each pixel. The estimations of depths are then filtered and used to refine the correspondence estimation in turn. Rendering from a given view angle is

performed using both refined depths and updated correspondence. In [48] and [49], the shapes and the segmentation of dynamic objects are jointly computed and optimized. Many of the existing efforts focus on encoding and transmitting FVV streams, assuming the models have been obtained offline. The online acquisition of FVV has yet to be explored in depth. RT3DV developed in this work is perhaps the first effort to generate free-viewpoint video in real-time.

2.3 Real-time 3D reconstruction with RGB-D camera

A real-time template-based reconstruction method of dynamic scenes is demonstrated in [50], in which an online template was deformed to fit the depth data from an RGB-D camera. The template is non-rigidly tracked to provide a detail layer to account for high-frequency details. However, a rigid template must be captured at the beginning[50]. DynamicFusion[51] is the pioneering work for real-time and template-free 3D reconstruction of dynamic scenes using RGB-D cameras, where a canonical reference model is updated incrementally by unwarping depth measurements returned with a single RGB-D camera at a real-time rate. Several follow-ups improved the quality of reconstruction via additional constraints. For example, VolumeDeform[52] combines depth correspondences with robust sparse correspondences (SIFT) to avoid drift. Fusion4D [53] extend [51, 52] to a multi-view scenario in which 8 RGB-D cameras capture depth data simultaneously, and multiple GPUs are used to compute the deformation field and the fusion of all data frame. However, the examples shown in [51, 52, 53] are limited to the reconstructed scene only undergoing slow motion and minor topological changes.

KillingFusion[54] estimates a dense deformation field in the TSDF space constrained by a damped Killing motion via a variational formulation, capturing more free movements. SobolevFusion[55] proposes to use Sobolev gradient flow to compute the deformation field and determine the voxel correspondences by matching the low-dimensional signatures of their Laplacian eigenfunctions, allowing large motion and topological changes of the scene. The recent work [56] uses the dual back RGB cameras of a VR device to achieve real-time 3D rendering. In [57], a

video encoder is used to find a sparse 70×70 depthmap by block matching over a pair of rectified images, and then a fast Laplacian solver is used to smooth the depthmap. These methods all take in as input the depthmaps return by RGB-D cameras at a real-time rate. In our work, we tackle the problem of real-time 3D rendering using multiview RGB cameras, where depth information is derived from pure RGB images.

2.4 Neural Rendering

Neural rendering techniques leverage neural networks to generate virtual view. Early approaches in novel view rendering employed deep neural networks to explicitly learn the 3D model, based on either a single view or multiple views. For instance, Dupont *et al.*[58] used a network to learn the 3D scene representation from different views, rotated it into the desired view, and then transferred it back to a 2D image. Transformable Bottleneck Networks [59] learned the 3D volumetric representation from multiview images which then are warped to the desired output view and finally generated the virtual 2D images using a network.

Implicit model representation methods, as opposed to explicit approaches, are commonly employed in neural rendering. In the NeRF framework[60], a scene is represented as a continuous 5D vector-valued function, taking 3D coordinates and a 2D viewing direction as inputs and generating emitted color and volume density as outputs. This 5D function is approximated using a multilayer perceptron (MLP). Once the network is trained, a novel view can be synthesized using traditional volume rendering techniques, projecting output colors and densities into an image. The significant contributions of this work lie in both the novel problem formulation and the associated optimization techniques.

Building upon NeRF[60], NeRV[61] includes the simulation of light transport, enabling rendering under arbitrary illumination conditions. In contrast to NeRF, NeRV represents a scene that absorbs and reflects light emitted by external sources. NeRF utilizes two MLPs: a "shape" MLP estimating volume density δ and a "reflectance" MLP computing BRDF parameters for input 3D points. Drawbacks of

both methods include the need for a dozen multiview images for training, a lengthy training duration of approximately 1-2 days, and the restriction on static scenes.

To address processing time concerns, IBRNet[62] synthesizes novel views by interpolating nearby source views. Density features are derived from multiview-aware features using an MLP fed with 2D features from nearby multiview images. A ray transformer module aggregates density features to predict the final density, while multiview-aware features concatenated with view direction yield blending weights using an MLP for color prediction.

Pixelnerf[63] aims to generalize NeRF[60] using very sparse input views. It extracts a feature volume with a CNN encoder and generates image features for each point on a camera ray by projecting the feature volume to the image plane. NeRF[60] is then used to predicts density and color for the query point using these image features. While Pixelnerf[63] can synthesize novel views with sparse multiview images, there is room for improvement in network efficiency for real-time applications.

To expand NeRF[60] to dynamic scenes, [64] introduced a deep learning model to implicitly encode the scene and generate novel views at any given time. The architecture proposed in D-NeRF[64] incorporates time as an additional input and divides the learning process into two distinct fully connected networks. The first, a deformation network, is responsible for mapping all scene deformations to a shared canonical representation. The second, a canonical network, then takes this canonical representation and maps it to the deformed scene for a specific time instance. In a multiview setup, DyNeRF[65] is efficiently trained to learn a time-conditioned neural radiance field, capturing dynamic scene variations through a learned compact and expressive time-variant latent code. Despite the effectiveness of both methods in handling general dynamic scenes, they share a common limitation of relying on dense training data and have slower inference speeds. Furthermore, these methods are naturally offline, whereas videos must be processed in an online fashion.

2.5 2D and 3D Video Stabilization Methods

Video stabilization algorithms can generally be divided into three categories: (1) 2D methods [25, 21, 19, 26, 20, 24, 22, 28], (2) 3D methods [66, 67, 68, 69, 70], and (3) learning-based methods [71, 72, 73, 74, 75, 76]. The 2D and 3D methods are considered conventional but differ in the assumed global camera motion model. A 2D method assumes that the global camera motion between consecutive frames is an affine or homography transformation, whereas 3D methods try to reconstruct the relative 3D camera poses for each video frame.

2D algorithms start by estimating 2D global camera motion, such as affine transformation or Homography, between consecutive frames. Optical flow [25, 21] and geometry transformation [19, 26, 20, 24, 22, 28] are two conventional methods. Feature points are first extracted from both video frames. Then RANSAC [29] is used to select a subset of matched features to estimate the transformation parameters.

The next step is to derive a smooth motion path. The processed motion path should be sufficiently smooth so as not to cause discomfort during viewing. Several motion-smoothing methods have been proposed in the literature, including low pass filtering [22], Kalman filtering [77, 31, 78], Gaussian Filtering [21], Spline Smoothing [20], Motion Vector Integration[26], etc. Grundmann et al. [24] and Liu et al. [22] state that a desirable stable camera path should follow the cinematography principles. Grundmann et al. [24] also propose an offline algorithm based on Linear Program optimization with L1-smoothness constraints to find a camera path obeying such principles.

The 2D method achieves a great balance of robustness and efficiency and thus is a great choice for real-time development. Optical flow [25, 21] and geometry transformation [19, 26, 20, 24, 22, 28] are two conventional ways to estimate the 2D inter-frame motion. The latter is becoming more popular because of its efficiency and robustness. Geometry transformation-based methods directly estimate 2D transformation due to camera jitters between adjacent video frames. Feature points are first extracted from both video frames. Then, a subset of matched feature points is selected to estimate the transformation parameters using the RANSAC

[29] algorithm, whose performance heavily depends on a user-specified parameter.

3D methods stabilize videos by reconstructing the camera poses in 3D space. Liu et al. [22] proposed to use Structure-from-Motion(SfM) to compute the relative camera and sparse feature trajectory in 3D space. Then each frame is wrapped to a user-specified path with the "content-preserving" principles to generate a stable virtual output. To avoid reconstructing long camera and feature trajectory, Liu et al. [28] smooth the basis trajectories of the subspace formed by the 3D feature track. Goldstein et al. [66] used epipolar constraints to estimate the fundamental matrices accounting for the stabilized 3D camera motion, reducing the dependency on long feature tracks. Methods [67, 68] using gyroscope are also proposed to estimate and 3D rotation. Additional hardware, such as depth sensor [69] and light field cameras [70], are also used to estimate the 3D camera motion and synthesize the stable virtual video.

Generally speaking, 2D methods are more robust and faster than 3D methods, but the 2D motion is insufficient to deal with complex scenes with significant depth variations and severe parallax. On the other hand, 3D methods can handle depth variation and generate great stabilized results in principle. However, the 3D motion model estimation is fragile to various degeneration, such as feature tracking failure, motion blue, etc. Besides, 3D methods often have expensive computational costs or require additional hardware devices. Hence, it is much slower and less robust than 2D methods, which limits its usage in real-time applications.

2.6 Learning-based Video Stabilization Methods and Real-time Methods

Recently DNN based video stabilization has attracted more and more attention. According to [23], StabNet [71, 23] is the first deep-learning approach for video stabilization, where an encoder and multigrain transformation regressor are trained under a Siamese network. The authors of StabNet [71] also collect 61 pairs of training videos. Xu et al. [72] train a GAN network to extract the affine transformation

for warping unsteady frames.

Instead of predicting transformations between images or in coarse grid level, PWStableNet [73] learns a pixel-wise warping map through a cascade encoder-decoder based on the siamese network. Yu et al. [74] first run a 2D method to stabilize the video. Then the optical flows are computed and fed into an encoder-decoder network to train a pixel-wise warp map. Choi et al. [75] propose an unsupervised deep approach that iteratively interpolates the input video to a stable video without cropping. Another unsupervised method is suggested by Shi et al. [76], where gyroscopes provide the actual camera poses.

It is noted that the performances of these learning-based methods highly depend on the training data [23] and can suffer from large motions [76]. Due to the lack of publicly available data sets, the conventional methods are more robust and perform better in a general setting than the learning-based methods [23]. Learning-based methods are typically computationally demanding and also unsuitable for real-time applications.

Although most techniques are offline for post-processing, several real-time approaches [25, 19, 20, 30, 78, 79] have been studied for video stabilization. Ratakonda et al. [79] uses Integration Projection Matching, which is very computationally efficient for computing the translation of consecutive frames. However, the method is limited as they assume camera motions are always translational. Most of the current real-time methods [19, 30, 78] are geometry-transformation-based, in which RANSAC is used to estimate the inter-frame transformation, and Modified Kalman filter [25, 30, 78], Spline smoothing [20], and low-pass filter [19] is employed to smooth the motion parameters on-the-fly.

CHAPTER 3
Towards Real-Time 3D Visualization with Multiview RGB Camera Array

3.1 Overview

The inputs to the RT3DV algorithm are video streams acquired synchronously by cameras on a camera array. The outputs are a 3D surface model consisting of connected triangular meshes and a texture map (color, texture, and shade) for each triangular mesh. These outputs will be forwarded to a 3D rendering engine (Unity[80] in this work) to display a stereopsis video from given viewpoints. The RT3DV algorithm performs the following tasks for each video frame: (a) identifying 2D distinct feature points at each view (camera), (b) establishing correspondence of 2D feature points across all pairs of views, (c) estimating the 3D world coordinate of corresponding 2D feature points, (d) applying the Delaunay graph cut algorithm [40] to reconstruct the 3D triangular mesh surface model using the estimated 3D feature points as its vertices, and (e) estimating corresponding appearance map (texture, color, shade) for each triangle surface pigment.

When the algorithm is initiated (initiation mode), the cameras need to be calibrated to estimate their intrinsic parameters (focal length, pixel scaling, etc.) and extrinsic parameters (positions and poses). If the camera array remains stationary throughout the video, the camera parameters will be assumed available, and the initiation mode will not be executed anymore. Once the cameras are calibrated, the RT3DV algorithm will be executed in either a feature detection mode or a feature tracking mode. In the feature detection mode, 2D feature points will be detected at each camera's current frame. In the feature tracking mode, existing 2D features

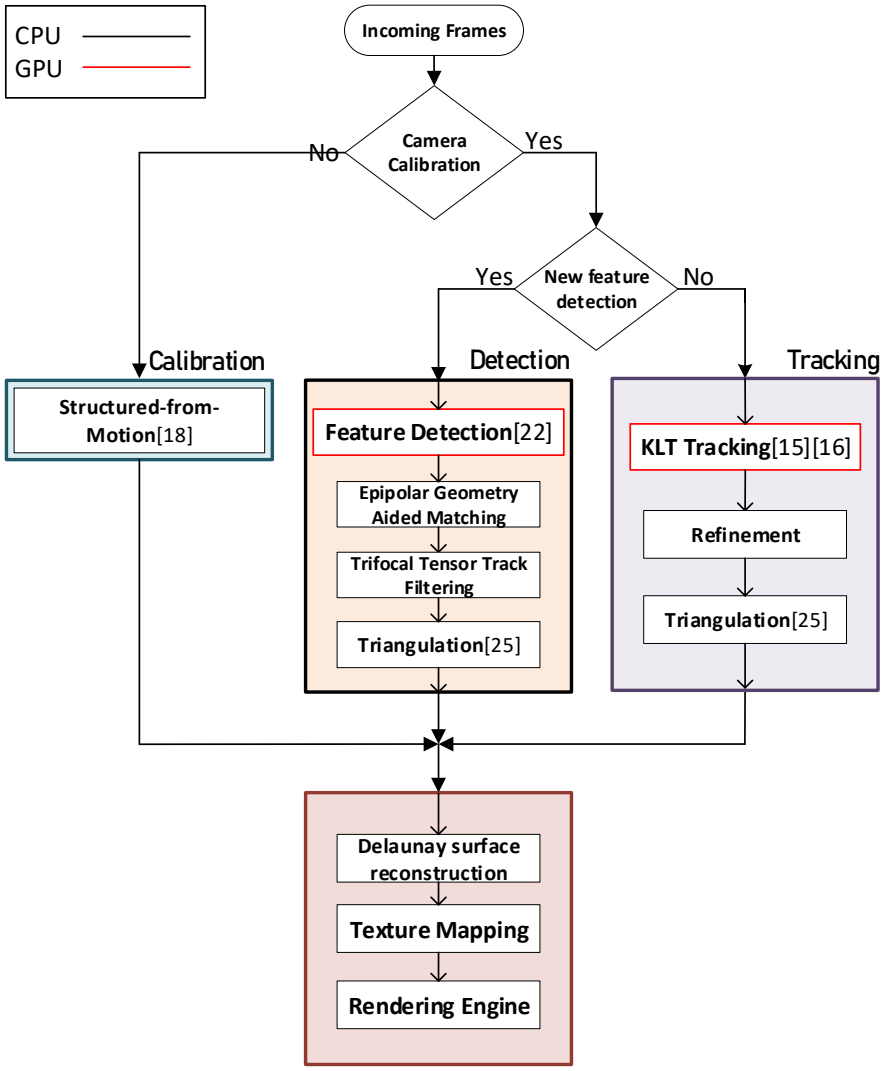


Figure 3.1: Block diagram of the proposed pipeline RT3DV

from the previous frame will be tracked. Leveraging the temporal correlations between successive video frames, the feature detection mode will be executed periodically with the feature tracking mode executed in between. The pipeline and block diagram of RT3DV are shown in Fig. 3.1.

3.2 Towards real time 3D visualization

3.2.1 Initiation

To initiate the RT3DV algorithm, camera calibration will be performed. An incremental structure from motion (iSfM) algorithm [81, 10, 82, 83] will be applied to jointly estimate the camera intrinsic and extrinsic parameters and 3D coordinates of feature points.

First, the speeded-up robust feature (SURF)[32] detection algorithm is applied to the first video frames of all cameras to detect local features. Each detected local feature is represented by its 2D image coordinate within the video frame and a feature descriptor characterizing its local appearance. A fast matching algorithm FLANN[33] will then be applied to find figures across neighboring views having similar feature descriptors (appearance consistency). This appearance-based matching results will be verified using the epipolar geometry constraints. The RANSAC[29] algorithm will be applied to select a subset of matching 2D feature points of two views to estimate the corresponding fundamental matrix[11]. If the majority of remaining 2D feature points of both views also meet the epipolar constraints with the estimated fundamental matrix, the relative positions (extrinsic parameters) between this pair of cameras then may be determined. Matching 2D feature points that fail this geometric consistency check will be deemed as outliers and discarded. Based on the estimated positions and poses of cameras, 3D coordinates of the matching 2D feature points may be determined. Given these estimated 3D coordinates, one may proceed to refine the camera calibration. And then, the 3D coordinates will be refined further. This iterative Bundle Adjustment [84] process will converge as no further changes are observed. The iSfM repeats the above steps for one camera at a time until all cameras are processed. On completion of iSfM, the calibrated camera parameters and estimated 3D coordinates of feature points will be made available for subsequent frames.

3.2.2 Fast Reconstruction with Feature Tracking

Given the calibrated camera parameters, the set of matching 2D feature points, and corresponding 3D coordinates, one may leverage the temporal correlation of videos to update the 2D feature positions using feature tracking instead of the time-consuming feature detection.

The Kanade–Lucas–Tomasi (KLT) feature tracker [85] will be used to track local movement of an existing 2D feature available from the previous frame. The outcome will further be refined by applying a block matching algorithm using the Sum of Absolute Differences (SAD) similarity metric.

$$\mathrm{SAD}(k,l) = \sum_{(i,j)\in N} |E_p(i,j) - E_c(i+k, j+l)| \tag{3.1}$$

where E_p, E_c denote the previous and current frame and N denotes a template window in a feature point's local neighborhood.

Since only existing 2D feature points from the previous frame are tracked, the feature correspondence relationship will remain unchanged unless a 2D feature disappears (cannot be tracked) due to dynamic scene change, in which case the track will be discarded. If a matching 2D feature point changes its position after tracking, its corresponding 3D coordinates will also be updated by triangulation using the stander DLT method for [11]. Otherwise, the previously estimated 3D coordinates will remain unchanged. This on-demand update strategy saves lots of computation when only a small fraction of feature points move between successive frames.

3.2.3 Fast Reconstruction based on Feature Detection

Feature tracking will capture movements of existing features in a dynamic scene. It does not, however, detects the presence of new features. Thus feature detection will be performed periodically in the RT3DV algorithm. The period between two feature detection frames depends on prior knowledge of the dynamics of the scene and

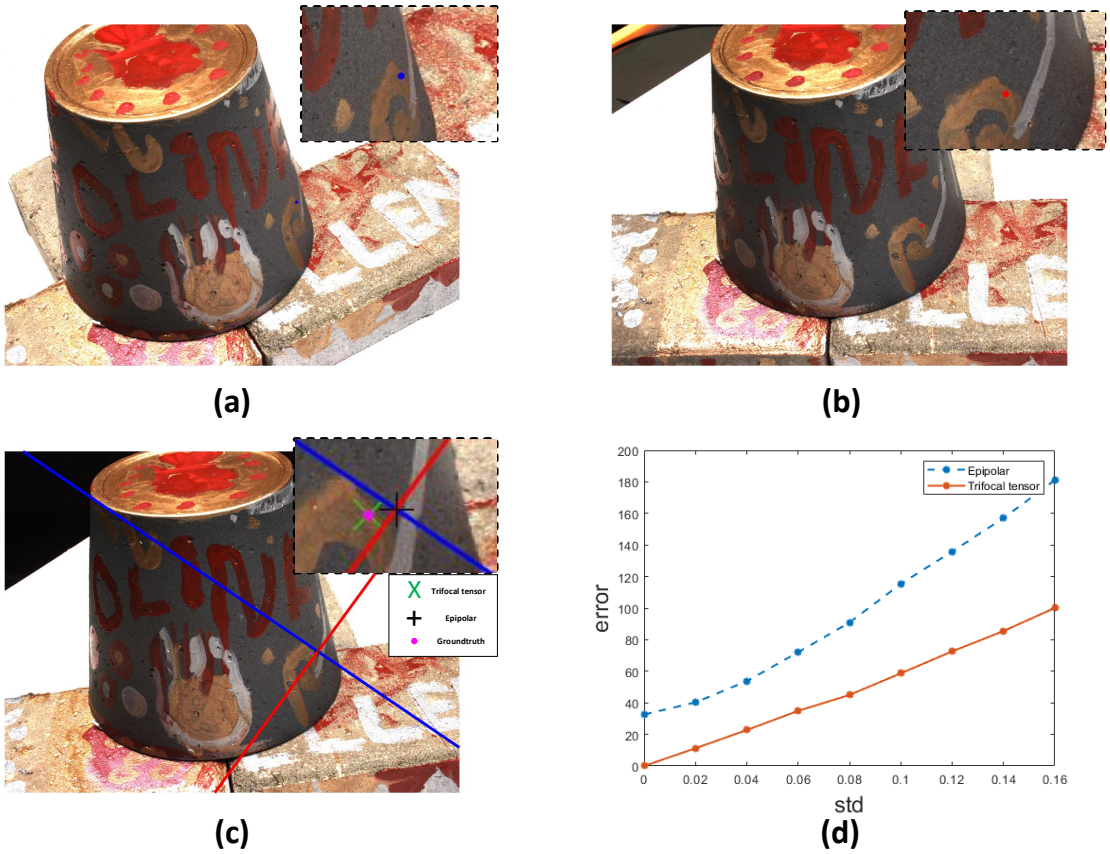


Figure 3.2: (a)A feature point is shown in the first view. (b)The corresponding point in the second view. (c) The corresponding point in the third view can be found by intersecting two epipolar lines or trifocal tensor. The epipolar method is prone to error while trifocal tensor method is more robust. (d) Average transfer errors for the epipolar method and trifocal tensor method over DTU dataset. Gaussian noise with different standard deviation values are added to the corresponding image points.

can be adjusted. We assume that the camera calibration parameters are available. Therefore, after new feature detection is performed, the robust feature matching process can be accelerated by enforcing the epipolar geometry[11] and aided by trifocal tensor [11]. Specifically, with calibrated cameras, the essential matrix $\mathbf{E}$ between any two cameras in the camera array is available. If a 2D feature point $\mathbf{x}$ in the video frame of one camera and another 2D feature point $\mathbf{x}'$ in the video frame

of another camera correspond to the same 3D point, then

$$\mathbf{x}'^{\mathsf{T}}\mathbf{E}\mathbf{x} = \mathbf{x}'^{\mathsf{T}}\boldsymbol{\ell} = 0 \tag{3.2}$$

where $\boldsymbol{\ell} = \mathbf{E}\mathbf{x}$ is the *epipolar line*.

Instead of using equation (3.2) to verify the matches, we only retain the matches if the corresponding point is within 3 pixels from the epipolar line. This verification procedure only takes constant operations for each match, and thus the complexity is $O(N_m)$, where N_m is the number of initial matches returned by FLANN. Then, we build feature tracks (2D feature correspondences across all views) from the remaining matches. The above procedure is called epipolar-geometry-aided matching. These tracks are then further refined by trifocal tensors.

Since all camera poses are available, we can quickly calculate the epipolar line for each 2D feature. A simple extension of epipolar geometry can help us find outliers: given a pair of matched points $(\mathbf{x}_1, \mathbf{x}_2)$, the third corresponding point $\mathbf{x}_3$ must pass both epipolar lines $\boldsymbol{\ell}_{13}$ and $\boldsymbol{\ell}_{23}$, where

$$\boldsymbol{\ell}_{13} = \mathbf{K}_3^{-\mathsf{T}}\hat{\mathbf{T}}_{13}\mathbf{R}_{13}\mathbf{K}^{-1}\,\mathbf{x}_1 \tag{3.3}$$

$$\boldsymbol{\ell}_{23} = \mathbf{K}_3^{-\mathsf{T}}\hat{\mathbf{T}}_{23}\mathbf{R}_{23}\mathbf{K}_2^{-1}\,\mathbf{x}_3 \tag{3.4}$$

In principle, we can determine $\mathbf{x}_3$ by intersecting $\boldsymbol{\ell}_{13}$ and $\boldsymbol{\ell}_{23}$:

$$\mathbf{x}_3 = \boldsymbol{\ell}_{13} \times \boldsymbol{\ell}_{23} \tag{3.5}$$

However, this approach fails when $\boldsymbol{\ell}_{13}$ and $\boldsymbol{\ell}_{23}$ are parallel and will be inaccurate if they are nearly colinear. This happens if the 3D point $\mathbf{X}$ lies on or near the trifocal plane defined by the three camera centers.

The degeneracy of the epipolar method can be avoided by using the trifocal tensor in three views which is analogous to fundamental matrix in two [11] views. The idea is to construct a homograpy by finding a plane defined by the back-projection of a line in the second view using the trifocal tensor. The homography

and $\mathbf{x}_3$ are [11],

$$h_i^k = l_2^j \mathcal{T}_i^{jk} \tag{3.6}$$

$$x_3^k = \sum_{i=1}^{3} \sum_{j=1}^{3} h_i^k x_1^i \tag{3.7}$$

where $\mathbf{x}_1 = (x_1^1, x_1^2, x_1^3)$, $\mathbf{x}_3 = (x_3^1, x_3^2, x_3^3)$, the line in the second view is $\mathbf{l}_2 = (l_2^1, l_2^2, l_2^3)$, and $\mathcal{T}_i^{jk} = a_i^j b_4^k - a_4^j b_i^k$ is the trifocal tensor, and a_i^j, b_i^j are the (i, j) element of the camera projection matrices $\mathbf{P}_2$ and $\mathbf{P}_3$ for the second and third view. A good chioce for $\mathbf{l}_2$ is the line that is through $\mathbf{x}_2$ and perpendiculat to the epipolar line. A comparison of accuracy for the epopolar method and the trifocal tensor method is shown in Fig. 3.2.

We couple the above procedure with RANSAC to filter outliers in a track and find the largest support set of corresponding points. Because each view has at most five elements (thus ten pairs possible), we can quickly iterate the ten possible pairs for each track. Once the outliers are filtered out, the stander DLT method[11] is used to triangulate for the 3D position for each track. As shown in Fig. 3.3, a better rendering result is achieved with the proposed trifocal tensor filtering procedure. More discussion about Fig. 3.3 can be found in Sec. 5.5

The Trifocal-Tensor-based Track Filtering is summarized in Algorithm 1.

3.2.4 Surface Reconstruction and Texture Mapping

Once we have an accurate point cloud, the next step is to reconstruct a 3D surface model out of it. The desired surface reconstruction algorithm should not only work well with the sparse nature of our reconstructed point cloud but also be very computationally efficient to satisfy the real-time requirement. We use the Delaunay graph cut algorithm by Labatut et al. [40] because it is fast and robust to changes in point density, which helps to reconstruct difficult surface parts. In their work, they also showed that their approach is very robust against outliers.

Once we obtain the 3D surfaces, each surface is projected to all views that observe it. If the 3D surface is viewable by multiple views, we retrieve the texture

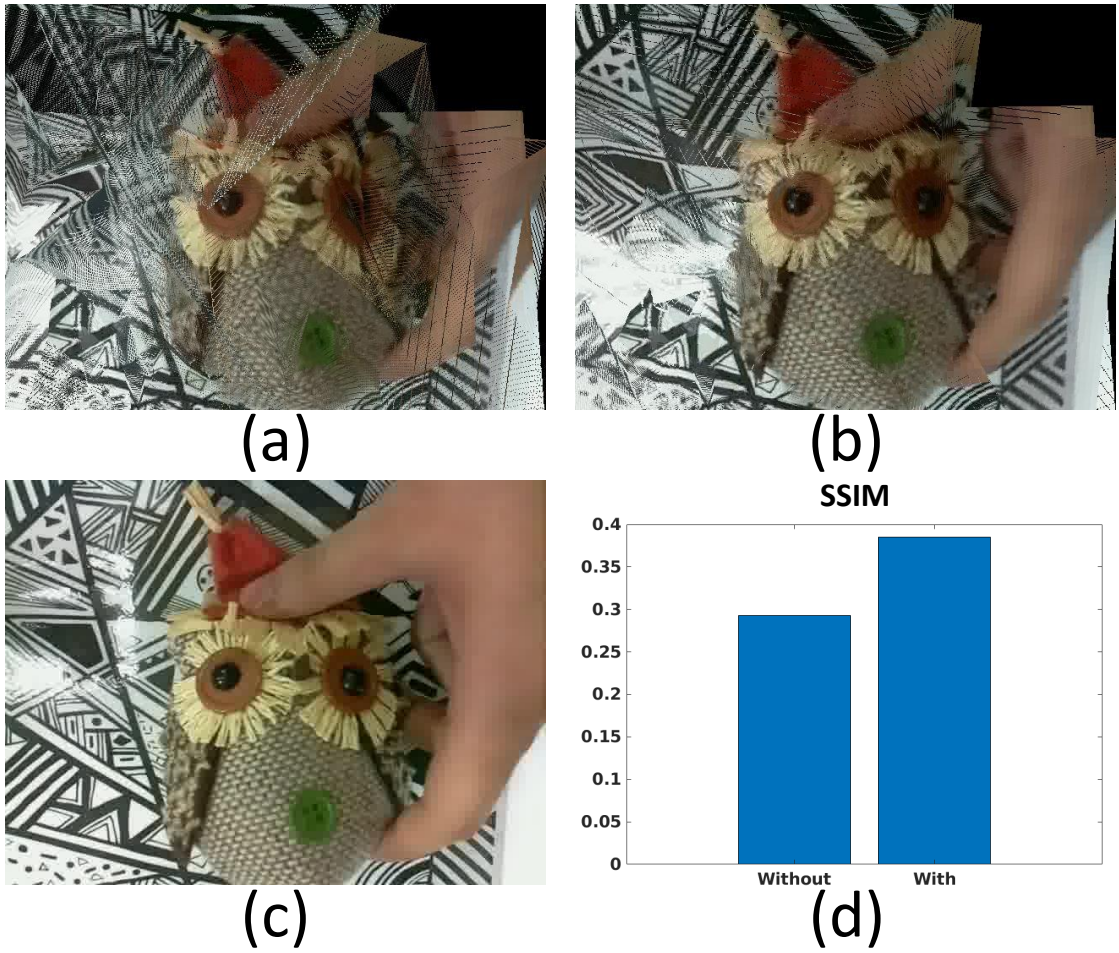


Figure 3.3: Rendering result. (a) rendered view without trifocal tensor filtering (b) with trifocal tensor filtering (c) the ground truth (d) Structural similarity (SSIM) index for rendering result with and without trifocal tensor filtering.

from the view whose viewing angle is smallest with the normal of the 3D surface:

$$\arg\min_{C} \langle \mathbf{v}_C, \mathbf{n} \rangle \tag{3.8}$$

where $\mathbf{v}_C$ is the viewing angle of view C, and $\mathbf{n}$ is the normal of the 3D surface.

Algorithm 1 Trifocal-Tensor-based Track Filtering

```
1: procedure TRACK FILTERING(Tracks, k)
2:     for each track T in Tracks do
3:         for each possible element pair t_i, t_j in T do
4:             Maintain set C for each pair
5:             for each t_r in T\{t_i, t_j} do
6:                 Compute t̂_r by Equation 3.7
7:                 if dist(t_r, t̂_r) < k then
8:                     Add t_r to C
9:                 end if
10:            end for
11:        end for
12:        T = arg max_C |C|                    ▷ keep the largest support set
13:    end for
14: end procedure
```

3.3 Experiment

3.3.1 Setup and Protocol

We evaluate the proposed RT3DV algorithm using the hardware platform shown in Fig. 1.1. Five synchronous video streams are acquired from five micro-cameras (dark squares in Fig. 1.1(a)) simultaneously. The displacements between cameras are around 5 cm, as shown in Fig. 1.1(a). The camera array assembly is mounted on the top of an FLC laparoscope trainer box. The objects are placed at the bottom of the box and will be moved manually during the video capture to emulate a dynamic scene.

This platform is a prototype 3D visualization system developed to enhance the visualization of traditional laparoscope [8, 9]. Each camera has a resolution of 640×480 pixels and has a frame rate of 30 frames per second (fps). Each camera is attached to a Raspberry Pi video capturing board, which compresses the video into Mpeg-4 format. The compressed video is then transmitted through Ethernet cables to a desktop PC to be processed. The PC is equipped with an 8-core 4.00 Hz

i7-6770k CPU, a GeForce GTX 2080 Ti GPU, 16 GB main memory running Ubuntu 16.04 operating system.

For the baseline pipeline, we used the C++ implementation as SfM[10]. We chose the Cuda implementation of MVS[1]. The surface reconstruction[40] is implemented in C++ by[82]. RT3DV is implemented in C++ using OpenCV with CUDA 9 enabled, where SURF[32] and FLANN[33] matching run in GPU.

We generated three sets of multiview video streams, one for each object, using the experiment platform described above. Besides, we perform the same experiment on the public available DTU MVS dataset[2], where underlying point clouds, the camera poses, and the images for each camera are all available. Since the objects are stationary, we first translate the underlying point cloud and then back-project it to all cameras to generate the multiview videos of moving objects.

We ran the baseline pipeline and RT3DV on all the multiview video frames in both datasets. We chose the running time to be the time interval between the completion of texture mapping between the successive frames. For the baseline pipeline, we excluded SfM and only measured the running time between the end of SfM to the end of texture mapping because SfM is only performed once as the initiation step in RT3DV. The processing time per frame is the average running time of all successive frames for each video. We conducted three trials of the experiment and reported the average processing time per frame of the three trials.

3.3.2 Results

Timing

Table 3.1: Timing for baseline pipeline

Objects	Dense[1]	Surface	Texture	Total
Mushroom	1.4 s	1.2 s	1.4 s	4.0 s
Post Office	1.3 s	2.0 s	2.1 s	5.4 s
Owl	1.4 s	1.9 s	2.4 s	5.7 s

Table 3.2: Timing for RT3DV. (Feature detection is performed every 10 frames)

Objects	Detect	KLT	Tri	Surface	Texture	Avg
Mushroom	66 ms	9 ms	10 ms	10 ms	13ms	43 ms
Post Office	57 ms	8 ms	9 ms	8ms	11ms	38 ms
Owl	63 ms	10 ms	11 ms	10 ms	12 ms	45 ms

For the three multiview videos collected in the FLC laparoscope trainer box, we remove the tracked features whose error is greater than five and triangulate the rest. The algorithm is set to re-detect features per 10 frames. The running time is related to the number of feature points being processed. With a large number of features, the processing time for tracking, epipolar-geometry-aided matching, and trifocal-tensor-based track filtering and surface reconstruction can worsen.The baseline pipeline tries to find the dense feature point cloud. However, for scenes that have few feature points, it fails to compute the scene geometry, which introduces holes in the reconstruction, as shown in the last row of Fig. 3.8. The timing for RT3DV and the baseline pipeline are summarized in Table 3.1 and 3.2. The number of feature points and 3D triangles can be found in Table 3.4.

Quality Metric and Evaluatoin

A key result of this work is that the visual quality of rendered images using a sparse point cloud is comparable to that using a dense point cloud. To facilitate objective visual quality evaluation, we adopt a protocol similar to the leave one out cross-validation method: we render a view at a viewing angle that coincides with one of the cameras and compute the peak signal to noise ratio (PSNR) between the rendered video frame and the acquired video frame (ground truth) without using any video frames from that validation camera.

Fig. 3.4 shows the original image and the rendered view generated by our method and the baseline pipeline. In our experiment, we track feature points for ten frames and re-detect new feature points. Fig. 3.5 shows the tracking result of our method as opposed to the classic pipeline. The averaged PSNR of Fig. 3.5

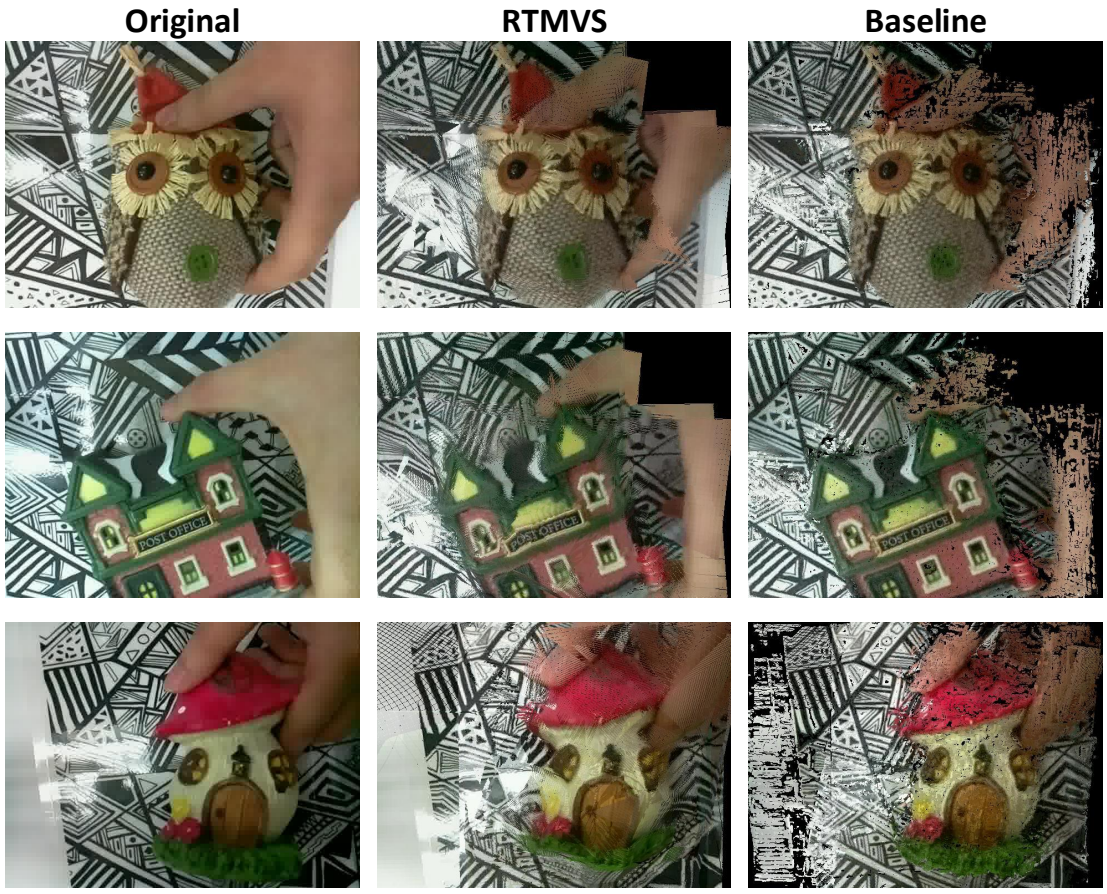


Figure 3.4: Rendered view to an original view using depthmaps only. Images in the 1st column are original images. The 2nd column are generated by RTMVS. The 3rd column are generated by the Patchmatch-based MVS implemented by Galliani et al. [1]

is recorded Table 3.4. The experiment is repeated with and without the epipolar geometry aided matching and the proposed trifocal tensor based filtering procedure. The rendered view and the structural similarity (SSIM) are computed and shown in Fig. 3.3. A qualitatively and quantitatively better result is obtained with the proposed matching and filtering procedure.

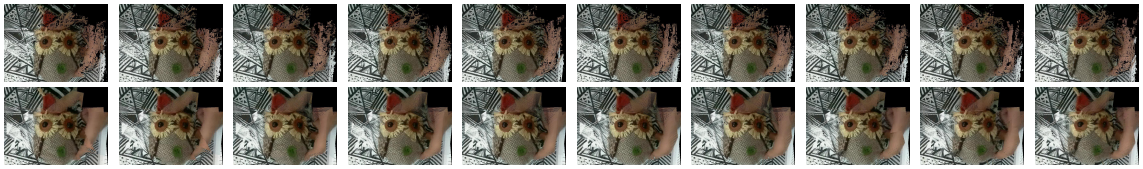


Figure 3.5: From left to right are the rendered view for the moving objects. The 1st row are the results of the baseline pipeline which takes more than 5s per frame. The 2nd row are the rendered results for RTMVS that takes 44ms per frame on average.

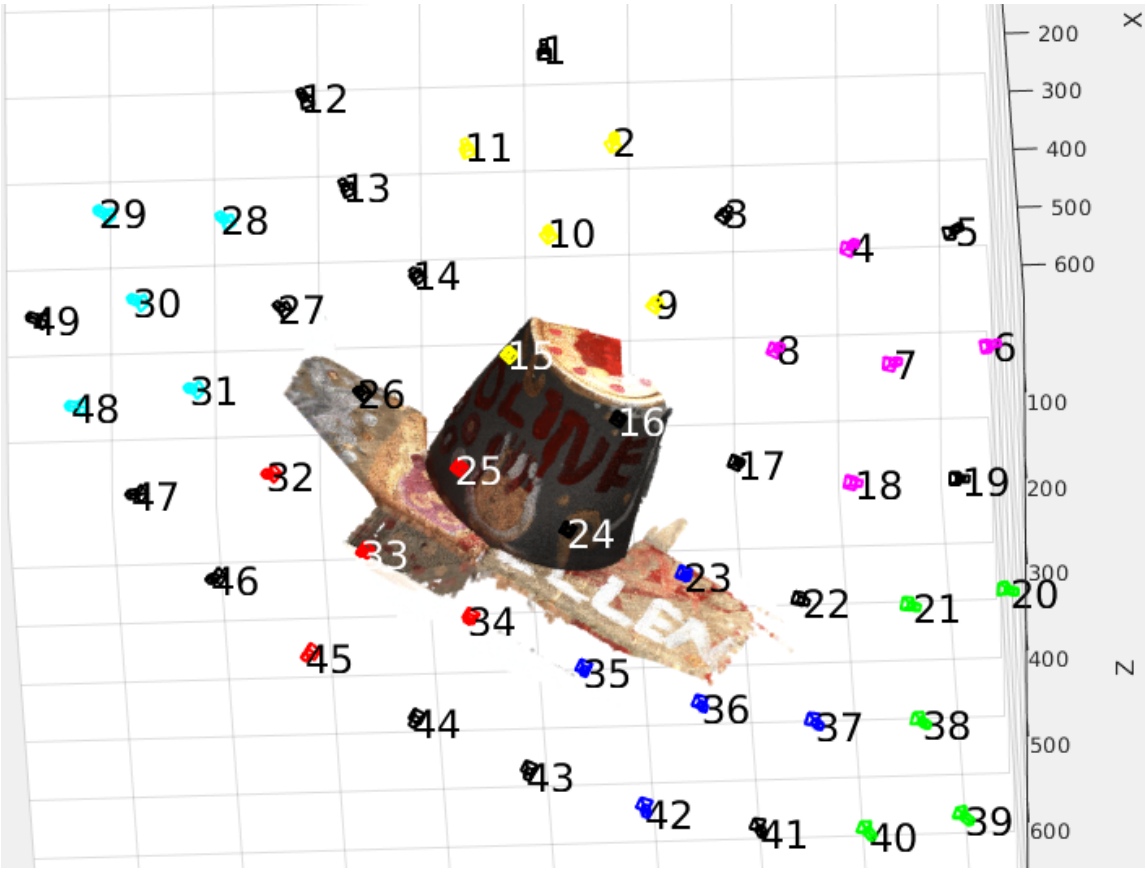


Figure 3.6: Camera configuration for computing PNSR for virtual view. Each camera configuration has five cameras. We reconstruct the middle view and compute the PSNRs with the real image of the middle view using RTMVS and the baseline pipeline. Config 1-6 correspond to camera group of Green, Blue, Red, Cyan, Magenta, Yellow.

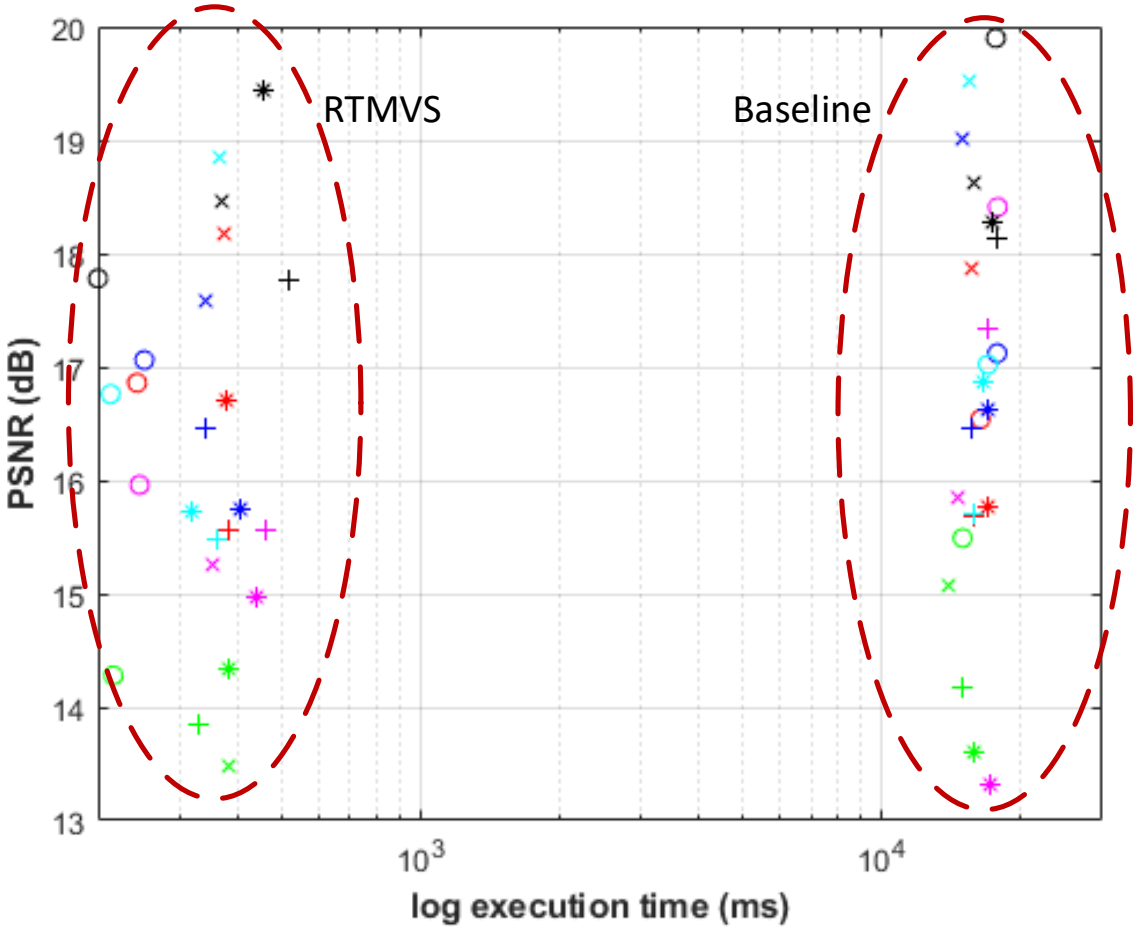


Figure 3.7: Comparison of processing time per frame and PSNR between RTMVS and the baseline pipeline. Config 1-6 correspond to Green, Blue, Red, Cyan, Magenta, Yellow. Object 1, 5, 6, 45 correspond to markers of cycle, plus, star, cross.

Table 3.3: PSNR and Processing time per frame for reconstructing the middle view of different camera configurations in Fig. 3.6

(PSNR,time)		Config1	Config2	Config3	Config4	Config5	Config6
Object1	RTMVS	(14.28 dB, 215 ms)	(17.06 dB, 251ms)	(16.86 dB, 242ms)	(16.76 dB, 212 ms)	(15.96 dB, 245 ms)	(17.78 dB, 199 ms)
	Baseline	(15.49 dB, 14929 ms)	(17.12 dB, 17745 ms)	(16.54 dB, 16329 ms)	(17.02 dB, 16936 ms)	(18.41 dB, 17789 ms)	(19.90 dB, 17622 ms)
Object5	RTMVS	(13.85 dB, 328 ms)	(16.46 dB, 342 ms)	(15.57 dB, 383 ms)	(15.48 dB, 359 ms)	(15.57 dB, 461 ms)	(17.76 dB, 518 ms)
	Baseline	(14.18 dB, 14993 ms)	(16.45 dB, 15643 ms)	(15.69 dB, 15867 ms)	(15.71 dB, 15709 ms)	(17.34 dB, 16878 ms)	(18.14 dB, 17752 ms)
Object6	RTMVS	(14.34 dB, 383 ms)	(15.75 dB, 405 ms)	(16.71 dB, 377 ms)	(15.73 dB, 318 ms)	(14.96 dB, 440 ms)	(19.43 dB, 457 ms)
	Baseline	(13.60 dB, 15845 ms)	(16.62 dB, 16871 ms)	(15.76 dB, 17025 ms)	(16.86 dB, 16470 ms)	(13.31 dB, 17129 ms)	(18.28 dB, 17241 ms)
Object45	RTMVS	(13.48 dB, 382 ms)	(17.59 dB, 339 ms)	(18.17 dB, 375 ms)	(18.84 dB, 366 ms)	(15.26 dB, 351 ms)	(18.45 dB, 371 ms)
	Baseline	(15.08 dB, 13863 ms)	(19.01 dB, 14967 ms)	(17.87 dB, 15598 ms)	(19.52 dB, 15475 ms)	(15.84 dB, 14614 ms)	(18.62 dB, 15736 ms)

For the DTU dataset, the camera configurations are shown in Fig. 3.6. We test the baseline pipeline and RT3DV on four objects (object 1, 5, 6, 45) for six camera configurations (Green, Blue, Red, Cyan, Magenta, Yellow), as shown in Fig. 3.6. For each camera configuration, we reconstruct the image of the center view and computer the PSNRs with the original image using RT3DV and the baseline pipeline. Fig. 3.7 shows the generated view. Table 3.3 and Fig. 3.9 show the PNSR and averaged processing time of our pipeline and the baseline pipeline, from which we see that the results of RT3DV and the baseline pipeline have similar PSNR but RT3DV is orders of magnitude faster than the baseline.

Table 3.4: Comparison of number of features and triangles and PSNR

	Features		Triangles		PNSR	
	Baseline	Ours	Baseline	Ours	Baseline	Ours
Mushroom	14733	370	29418	715	14.9 dB	14.4 dB
Post Office	15710	281	31383	585	17.1 dB	16.1 dB
Owl	17637	303	35240	622	16.9 dB	16.0 dB

3.4 Discussion

Number of features and processing time. The processing time of the proposed pipeline heavily relies on the accuracy and the number of feature points extracted from the scene. Our virtual view is generated by rendering the 3D model computed by the sparse point cloud. If the number of points in the point cloud is too few, the point cloud will not capture the 3D geometry of the scene well. On the other hand, if the number of features is too large with similar accuracy, the reconstruction would take too long to complete, as the number of features has a direct relation to each stage of the pipeline. In our experiment of the laparoscope training box, the number of features is between 280 to 400, and the averages reconstruction time per frame is around 42 ms.

Stationary camera poses The proposed pipeline assumes the relative camera poses are stationary. If the relative camera poses have changed, then we need to calibrate the cameras, which can be done by running SfM from scratch.

3.5 Conclusion

In this work, we propose RT3DV for near-field scenes. The proposed algorithm utilizes the temporal and spatial correlation of multiview videos and is faster than the state-of-the-art pipeline in order of magnitude. While the state-of-the-art pipeline reconstructs fine details on parts of the scene, it introduces holes on the part that has fewer features. Our efficient and straightforward pipeline can preserve the integrity of the scene and provide an adequate visualization result.

Figure 3.8: Virtual views using textured 3D surfaces for the objects Owl, Post Office, and Mushroom at a given frame. The virtual view is generated by 3D rendering engine. The 1st, 2nd, 3rd columns are the left, front, and right virtual view. The 1st row of each object is the result generated by the baseline pipeline, which takes around 5 s. The 2nd row is the result of the proposed pipeline, which takes around 42 ms on average.

Figure 3.9: Generated virtual center views for object 1, object 5, object 6, object 45 (in order) of the DTU dataset[2] by baseline pipeline and RT3DV.

CHAPTER 4
Real-time Moving Surgical Tool Reconstruction

4.1 Overview

In a typical laparoscopic surgery setting, the surgical tools, specifically graspers, interact against a background objects that remains relatively stable between successive frames. These graspers play a pivotal role in surgical procedures, enabling precise maneuvers that result in deliberate and controlled movements of tissues and organs within human interior body. The accurate perception of the 3D states of these surgical tools during operation is of utmost importance. Hence, the graspers demand special attention due to their central role in the surgeon's focus. A laparoscopic grasper, depicted in Fig. 4.1, is characterized by a semi-rigid structure consisting of a rod, two finger tips, and a joint point.

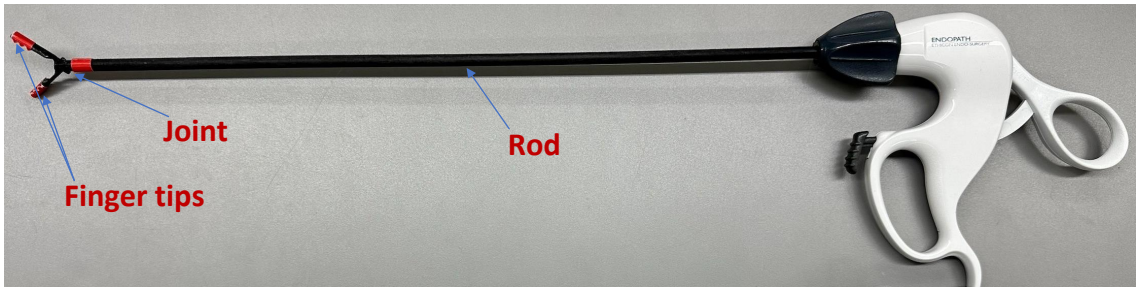


Figure 4.1: The surgical tool for a laparoscopic surgery, e.g. a grasper, consist of a rod, two finger tips and a joint point.

The primary objective is to efficiently and precisely reconstruct the 3D pose of the laparoscopic grasper through the utilization of synchronized multiview video

streams. Our assumption is that the laparoscopic environment maintains a static background, with two graspers serving as the surgical tools tasked with manipulating beans within a laparoscopic surgery training box. The foreground objects encompass rigid beans and the two graspers, simulating the tools in action, while the background scene incorporates one or more cups to introduce depth variations within the interior body, as shown in Fig. 4.2. To enhance the reconstruction and rendering of these textureless foreground objects, we introduce color markers for identifying key components and pre-established 3D models for generating virtual views.

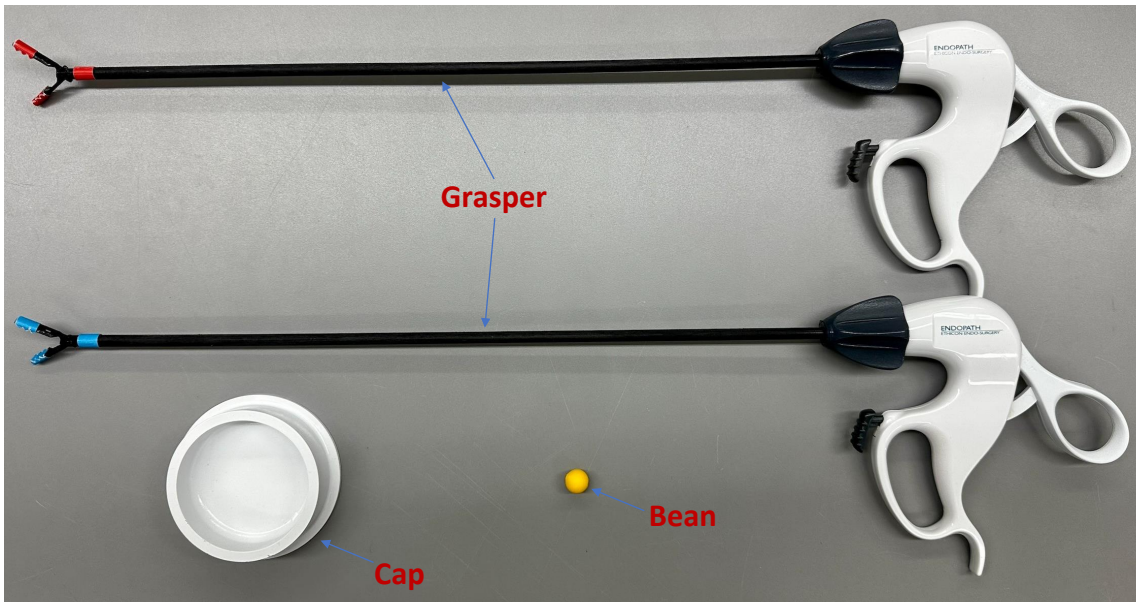


Figure 4.2: Foreground and background objects

Our hardware system comprises a laparoscopic training box replicating the abdominal cavity, a camera array facilitating synchronized real-time video streams, and specialized testing objects emulating surgical operations. The camera array consists of a camera holder and five Raspberry Pi units, each equipped with a Pi Camera. These Raspberry Pi units record real-time video streams, transmitting them to a PC via Ethernet cables. Positioned approximately 15 to 20 cm away, the camera array is oriented towards the objects undergoing simulated procedures, as

illustrated in Figure 4.3, showcasing the comprehensive hardware setup.

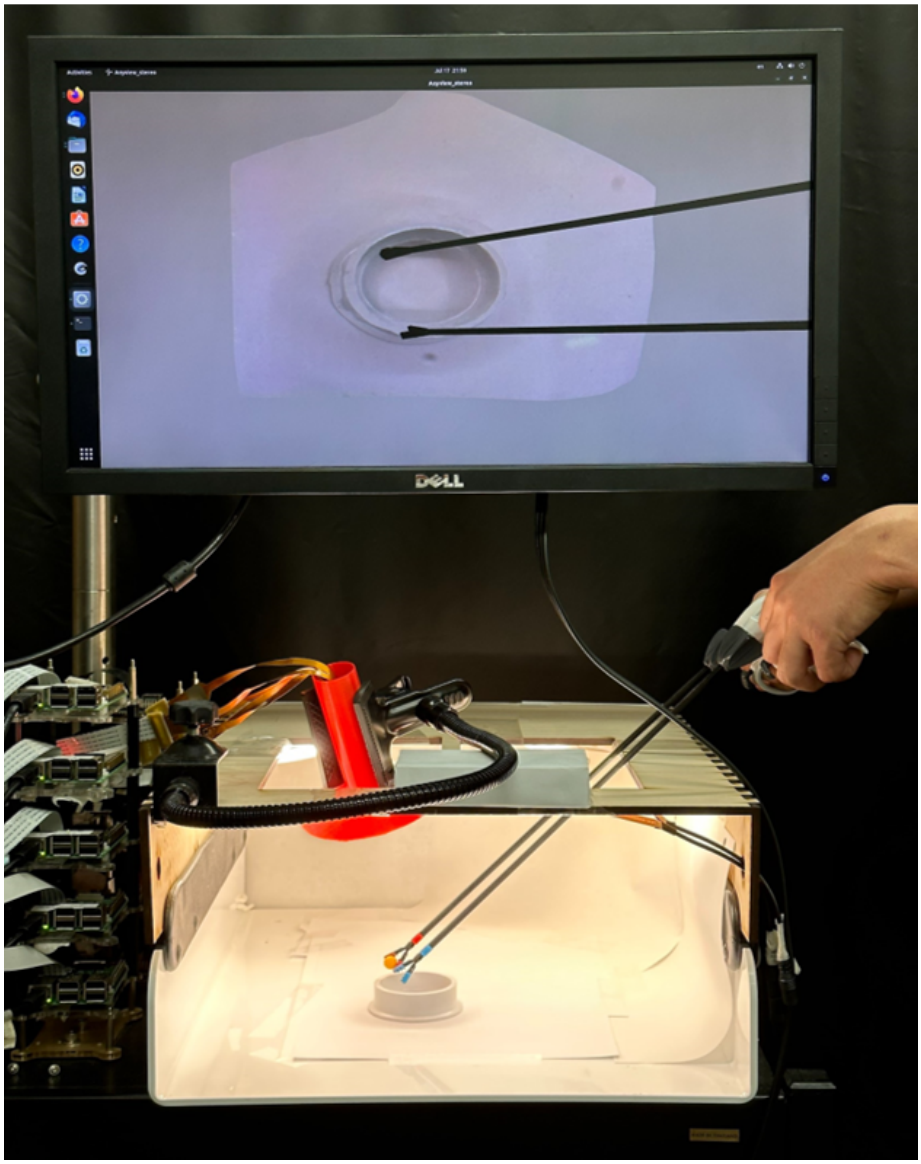


Figure 4.3: Hardware system

The proposed algorithm framework initiates with an offline calibration stage, where we determine camera poses, obtain dense 3D models for background objects, and establish a pre-built 3D model for the foreground object. In acquiring camera

poses and dense 3D models for background objects, a structured light with a random stride pattern is shined to backgrounds. Subsequently, Structure from Motion (SfM) [10] is applied to derive camera poses. Next, Multi-View Stereo (MVS)[1] is utilized to generate a dense 3D model for background objects. We employ SolidWorks to acquire the 3D models of the grasper and beans, stored as .PLY format.

The foreground objects testing encompass two graspers, one cup, and one bean A grasper is a semi-rigid object whose real-time configuration, i.e., shape and pose, can be determined by the 3D position of the joint point, the two fingertips, and the center line of the rod. The real-time configuration of the ball-shaped bean is determined by the 3D position of its center. Upon establishing the 3D configuration, we augment the pre-built 3D model of the object with respect to that configuration for virtual view generation. To facilitate robust reconstruction of these important key points and the center line of the rod, color patches is attached to the joint and the two fingertips of a grasper, as shown in Figure. 4.2.

In the reconstruction of the 3D pose for foreground objects, specifically the graspers and beans, our method involves an initial tracking phase for the two finger tips, the joint point, and the center line of the rod in each 2D view. To enhance the stability of 2D tracking and mitigate random jitters, we employ the Kalman filter. Subsequently, correspondences are established for the finger tips, joint point, and rod center line. With these correspondences in place, we perform real-time calculation of the corresponding 3D poses using a combination of Direct Linear Transformation (DLT)[11] and the Kalman filter.

4.2 2D foreground detection tracking

The foreground objects consist of two graspers and one bean. To reconstruct the 3D pose for a grasper, our focus lies in determining the 3D positions of the joint point, two finger tips, and the center line of the rod. This simplified approach is attributed to the semi-rigid nature of the grasper. In the case of beans, the reconstruction solely requires the 3D position of the center, as it represents a known 3D ball. In short, he reconstruction process for foreground objects can be distilled into two main

components: 3D key point reconstruction and 3D line reconstruction, summarized as followed:

- **Key point reconstruction**: grapser joint point, grasper finger tip, bean center
- **Line reconstruction**: grasper rod.

4.2.1 2D key point detection

The joint point and finger tips of the grasper and the bean are attached with color patches. In each frame, we initiate the process by obtaining a binary mask through moving object subtraction. Within this mask, template matching in the YCbCr space is employed to precisely locate the color patch. In instances where moving objects are absent or the color patch cannot be located, further processing of the object associated with the color patch is omitted.

Even upon locating the color patch, it often appears incomplete and is susceptible to noise contamination. To address this, we employ a series of iterations involving erosion and dilation to eliminate noise. Ultimately, the centroid of the detected mask serves as the 2D location for the color patch. This entire procedure is iteratively repeated for each color patch.

4.2.2 Rod center line detection

Given our knowledge of the rod's 3D model, deriving its 3D pose becomes feasible through analysis of its center line, considering one end of the rod is the joint point. Recognizing that the mask of the rod always have a pair of straight lines—due to the 3D-to-2D transformation of lines in perspective projection—we propose employing Hough transform to identify edges within the foreground mask.

In cases where the foreground mask includes the grasper, the Hough transform may yield multiple lines on each edge. To refine this, we utilize K-means with two classes, and the final two centroids resulting from K-means are considered as the images of the two edges of the rod. The ultimate step involves computing the 2D center line of the rod as the midpoint between the two detected edges.

4.2.3 Tracking

To alleviate the impact of noise and jitters associated with detected 2D key points and edge lines in each view, we suggest employing the Kalman filter with a constant-acceleration model, i.e.

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \mathbf{v}_t \Delta t + \frac{1}{2}\mathbf{a}_t \Delta t^2 \tag{4.1}$$

where $\mathbf{s}_t, \mathbf{v}_t, \mathbf{a}_t$ are the states, velocity, acceleration at time t.

For 2D key points $\mathbf{s}_p = [s_x, x_y] \in \mathbf{R}^2$ and 2D directions $\mathbf{s}_l = [s_x, s_y, s_z] \in \mathbf{R}^3$, the state vectors are

$$\mathbf{s}_p = \begin{bmatrix} s_x & s_y & v_x & v_y & a_x & a_y \end{bmatrix}^T \in \mathbf{R}^6$$

$$\mathbf{s}_l = \begin{bmatrix} s_x & s_y & s_z & v_x & v_y & v_z & a_x & a_y & a_z \end{bmatrix}^T \in \mathbf{R}^9$$

The state transition equation is

$$\tilde{\mathbf{s}}_{n+1,n} = \mathbf{F} \cdot \tilde{\mathbf{s}}_{n,n} + \mathbf{G}\mathbf{u}_n + \boldsymbol{\omega}_n$$

where

- $\tilde{\mathbf{s}}_{n+1,n}$ is the predicted system state vector at time $n+1$
- $\tilde{\mathbf{s}}_{n,n}$ is the estimated system state vector at time n.
- $\mathbf{u}_n$ is the control vector
- $\boldsymbol{\omega}_n$ is the process noise
- $\mathbf{F}$ is the state transition matrix
- $\mathbf{G}$ is the control matrix

Since the model has no input (control) vector, $\mathbf{u}_n$ is 0. Hence,

$$\tilde{\mathbf{s}}_{n+1,n} = \mathbf{F} \cdot \tilde{\mathbf{s}}_{n,n} + \boldsymbol{\omega}_n$$

where **F** for 2D key points and lines are

$$\mathbf{F}_{p} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbf{R}^{6\times 6}$$

$$\mathbf{F}_{l} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbf{R}^{9\times 9}$$

The observation matrix for 2D key points and lines are

$$\mathbf{H}_{p} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbf{R}^{2\times 6}$$

$$\mathbf{H}_{l} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbf{R}^{3\times 9}$$

Assuming that measurements are uncorrelated, the measurement covariance

matrices are respectively

$$\mathbf{R}_p = \begin{bmatrix} \sigma^2_{x_m x_m} & 0 \\ 0 & \sigma^2_{y_m y_m} \end{bmatrix} \in \mathbf{R}^{2\times 2}$$

$$\mathbf{R}_l = \begin{bmatrix} \sigma^2_{x_m x_m} & 0 & 0 \\ 0 & \sigma^2_{y_m y_m} & 0 \\ 0 & 0 & \sigma^2_{z_m z_m} \end{bmatrix} \in \mathbf{R}^{3\times 3}$$

First we compute the Kalman Gain

$$\mathbf{K}_n = \mathbf{P}_{n,n-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{n,n-1}\mathbf{H}^T + \mathbf{R})^{-1} \tag{4.2}$$

Once we have the measurement, i.e. 2D key points or lines, we update the estimates with the measurement.

$$\hat{\mathbf{s}}_{n,n} = \hat{\mathbf{s}}_{n,n-1} + \mathbf{K}_n(\mathbf{z}_n - \mathbf{H}\hat{\mathbf{s}}_{n,n-1}) \tag{4.3}$$

Followed by updating the estimate uncertainty

$$\mathbf{P}_{n,n} = (\mathbf{I} - \mathbf{K}_n\mathbf{H})\mathbf{P}_{n,n-1}(\mathbf{I} - \mathbf{K}_n\mathbf{H})^T + \mathbf{K}_n\mathbf{R}_n\mathbf{K}_n^T \tag{4.4}$$

Finally, state prediction is done by

$$\tilde{\mathbf{s}}_{n+1,n} = \mathbf{F}\cdot\tilde{\mathbf{s}}_{n,n} + \boldsymbol{\omega}_n \tag{4.5}$$

and extrapolating uncertainty by

$$\mathbf{P}_{n+1,n} = \mathbf{F}\mathbf{P}_{n,n}\mathbf{F}^T + \mathbf{Q} \tag{4.6}$$

The corrected state will be used as the final detected value for 2D key points or the line.

4.3 Real-time 3D reconstruction

Given the relative camera poses, the process of reconstructing a 3D point or line simplifies to establishing 2D correspondences across the multiple views.

4.3.1 Building Correspondence

With the bean possessing a unique color, establishing the correspondence for the bean is straightforward upon detection. Distinguishing between the two graspers relies on their respective colors, although the joint point and the two finger tips share the same color within a grasper.

Following the processing of the foreground mask in Section 4.2.1, three centroids remain for a grasper after filtering out other colors. Among these centroids, the one closest to the grasper center is identified as the joint point. To differentiate between the two fingers, we compute normal vectors: n_{cj} from the grasper centroid to the joint point, n_{jl} from the joint point to the left finger tip, and n_{jl} from the joint point to the left finger tip. The signs of the cross products $n_{jr} \times n_{cj}$ and $n_{jl} \times n_{cj}$ uniquely correspond to the left and right finger tips.

4.3.2 Reconstruction

Upon establishing the corresponding relationships for key points or center lines across different views, the reconstruction of 3D key points or center lines can be achieved through DLT[11]. However, recognizing that each tracked 2D observation comes with an associated uncertainty score calculated by the Kalman Filter, we propose employing a weighted DLT approach. This involves using the uncertainties as weights.

Suppose $\mathbf{A}$ is the design matrix in DLT, where we try to solve a homogeneous least square problem

$$\mathbf{AX} = 0$$

For 3D point reconstruction

$$\mathbf{A} = \begin{bmatrix} x^1\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y^1\mathbf{p}^{3T} - \mathbf{p}^{2T} \\ \vdots \\ x^n\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y^n\mathbf{p}^{3T} - \mathbf{p}^{2T} \end{bmatrix} \in \mathbb{R}^{2n\times 4} \tag{4.7}$$

where $\mathbf{p}_i^{jT}$ are the j^{th} row of the projection matrix $\mathbf{P}_i$.

For 3D line reconstruction

$$\mathbf{A} = \begin{bmatrix} \mathbf{l}_1^T\mathbf{P}_1 \\ \vdots \\ \mathbf{l}_n^T\mathbf{P}_n \end{bmatrix} \in \mathbb{R}^{n\times 4} \tag{4.8}$$

where $\mathbf{l}_i$ is the projection of the 3D line in view i, $\mathbf{P}_i$ is the projection matrix for the i^{th} camera.

Suppose the weight matrix is $\mathbf{W}$, then the weighted error e is

$$\mathbf{e} = \mathbf{WAX}$$

Minimizing

$$\mathbf{e}^T\mathbf{e} = \mathbf{X}^T\mathbf{A}^T\mathbf{W}^T\mathbf{WAX}$$

leads to the following system of equations

$$\mathbf{A}^T\mathbf{W}^T\mathbf{WAX} = 0$$

Hence, the right singular vector of the smallest singular value of $\mathbf{A}^T\mathbf{W}^T\mathbf{WA}$ is the least squares solution, where the weight matrix is defined to be

$$\mathbf{W} = \frac{\mathbf{P}_{n,n}}{100}$$

4.4 Result

To assess the correctness of the proposed algorithm, we adapt the leave-one-out strategy, where we compare the hold-out view with the rendered 3D model using the remaining views. If a the 3D model is perfect, the rendered view will be exactly the same as the hold-out view as shown in Figure. 4.4.

We also plot compared trajectory for the joint point with the proposed algorithm and vanilla DLT. As we can see from Figure4.5, the reconstruction result of the proposed algorithm is much more stable than the vanilla DLT.

4.5 Discussion and Conclusion

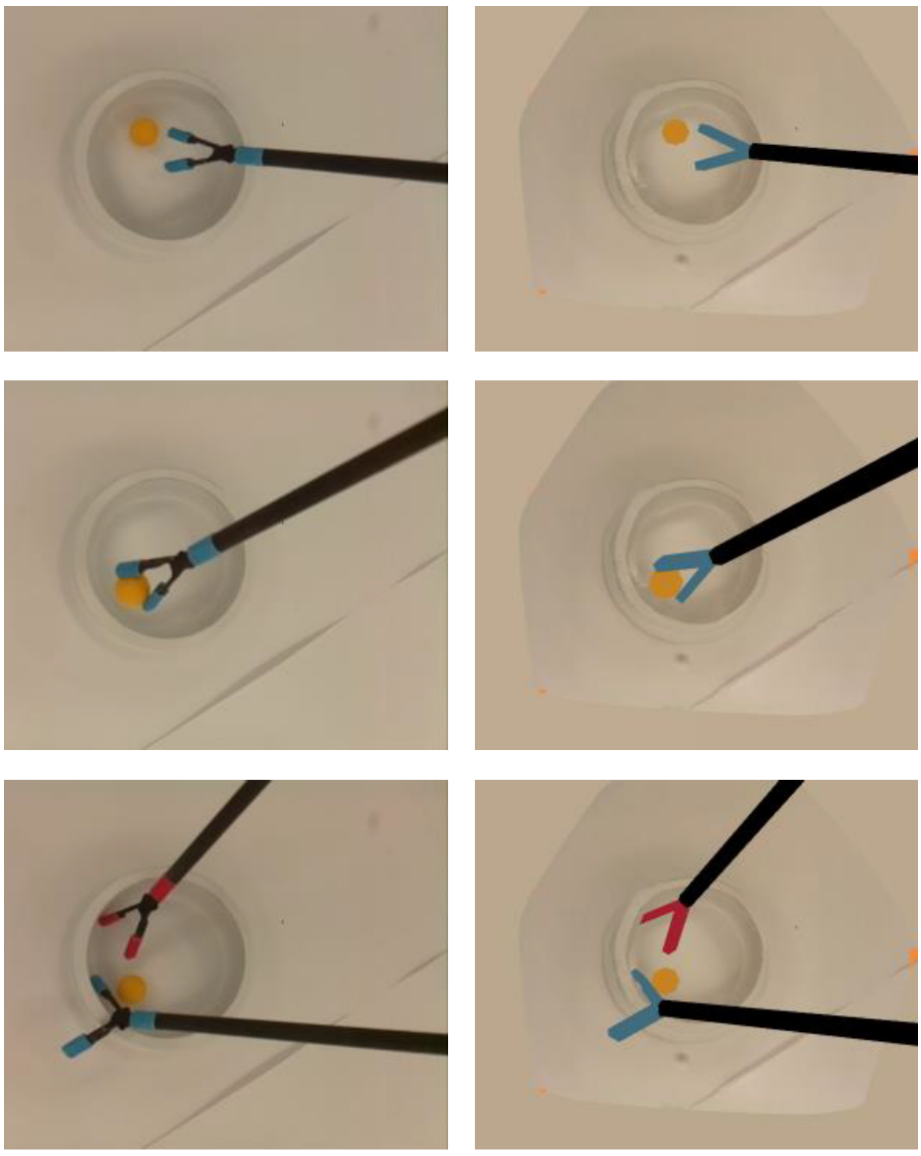
In this work, we focus on reconstructing the foreground objects, i.e. the surgical grasper and a bean. The proposed algorithm framework starts with an offline calibration stage, where we determine camera poses, obtain dense 3D models for background objects, and establish a pre-built 3D model for the foreground object with SfM[10] and MVS[1]. We Use SolidWorks to acquire the 3D models of the surgical grasper and beans.

The foreground objects testing encompass two graspers, and one bean. A grasper is a semi-rigid object whose real-time configuration can be determined by the 3D position of the joint point, the two fingertips, and the center line of the rod. The real-time configuration of the ball-shaped bean is determined by the 3D position of its center. Upon establishing the 3D configuration, we augment the pre-built 3D model of the object with respect to that configuration for virtual view generation. To facilitate robust reconstruction of these important key points and the center line, color patches is attached to the joint and the two fingertips of a grasper.

In the reconstruction of the 3D pose for foreground objects, our approach involves initial tracking of the two finger tips, the joint point, and the direction of the rod in each 2D view. We use the Kalman filter to mitigate random jitters in 2D tracking. Subsequently, we establish correspondences for the finger tips, joint point, and rod directions.

For reconstructing each corresponding track, we propose a framework that seamlessly integrates the Kalman filter and DLT, where DLT is reformulated as a weighted least squares problem, with weights determined by tracking uncertainties from the Kalman filter. This design stabilizes tracking results using the Kalman filter and improves the accuracy of reconstruction by accounting for tracking errors within the standard DLT.

The experimental results affirm the accuracy and efficiency of our algorithm. It's important to note, however, that our setup simplifies the complexity of laparoscopic surgery, involving only two graspers and a bean as foreground objects. Additionally, we make the assumption that the background remains static and possesses a distinct color from the foreground objects.



Ground Truth Ours (to existing view)

Figure 4.4: The rendered result of the proposed algorithm and the ground truth.

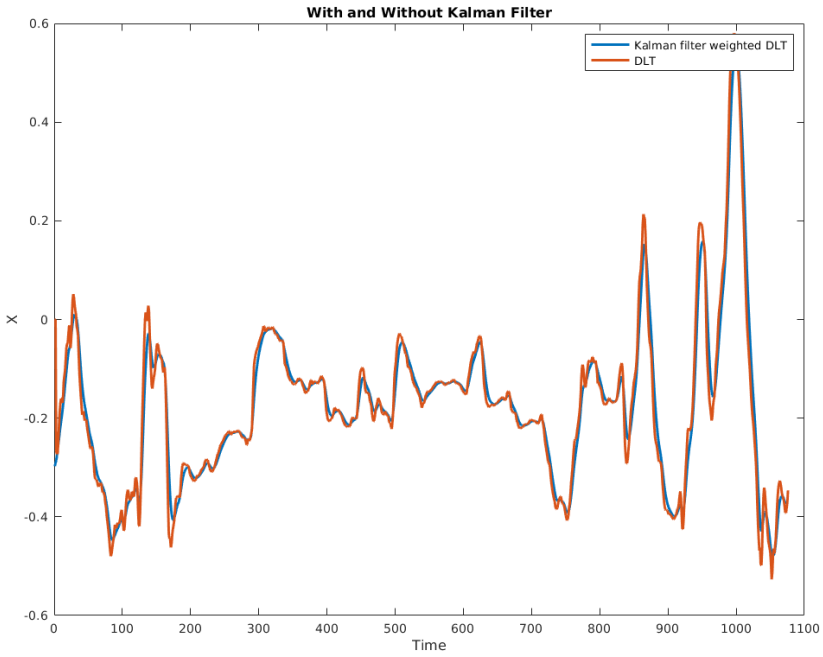


Figure 4.5: Stabilization result for the propose algorithm.

CHAPTER 5
Real-time Video Stabilization

5.1 Overview

This paper proposes an online real-time geometry transformation-based algorithm for video stabilization. Unlike other geometry transformation-based algorithms, we adapt and parallelize a new technique called *a-contrario* RANSAC (AC-RANSAC), which does not require any hard thresholds for inlier/outlier discrimination appearing in RANSAC. Hence, it can compute inter-frame global motions more robustly.

The proposed algorithm starts by estimating the inter-frame global motion between two consecutive frames except for the first one. First, features points for the current frame are extracted and matched over the previous frame. We choose SURF [32] and FLANN [33] as our feature extraction and matching algorithm because of their delicate balance of robustness and efficiency [34, 35]. Then, the parallel AC-RANSAC is used to estimate the inter-frame geometry transformation, from which the motion parameters (translations, rotations, and scales) are derived. The inter-frame global motion estimation is performed for each incoming frame.

Then the algorithm enters the motion smoothing phase, which requires a user-specified parameter N. If the current frame number is less than N, the camera motion for the current frame will be smoothed by our cinematography principles guided modified recursive least squares algorithm (C-MRLS). After the N^{th} frame, we stabilize the current camera motion with our modified sliding window least squares algorithm (MSWLS). Finally, the current frame is warped to the previous stable space in the motion compensation stage to create a stabilized video sequence. The whole algorithm pipeline is shown in Fig. 5.1

The key contribution of the proposed algorithm is a novel least-squares-based

smoothing cost for estimating the intentional motion and its associating solver that minimizes the cost in linear time.

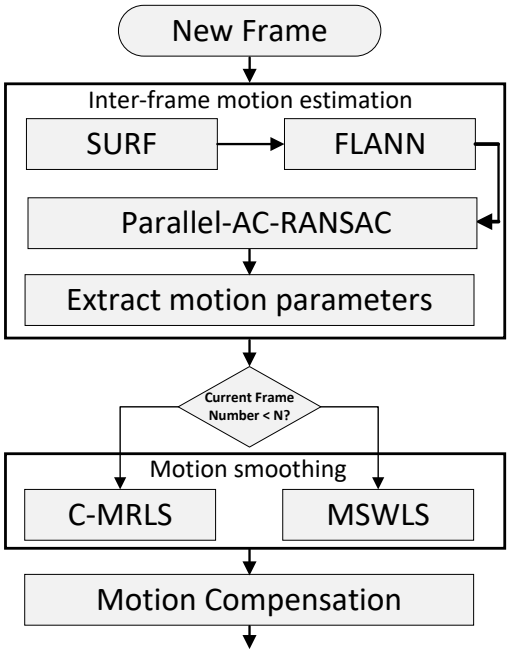


Figure 5.1: Overall flowchart of the proposed algorithm, where N is specified by users.

5.2 Inter-frame global motion estimation

5.2.1 Match Selection

Matches returned by FLANN are outliers contaminated. There are three main types of outliers: (1) matches with low matching scores, (2) matches belonging to moving foreground objects which would disturb the estimation of the global camera motion, and (3) matches with high matching scores but do not correspond to the same 3D point. We could use a threshold for the first type of outliers to filter out the low-score matches. Although finding a generic threshold that works well

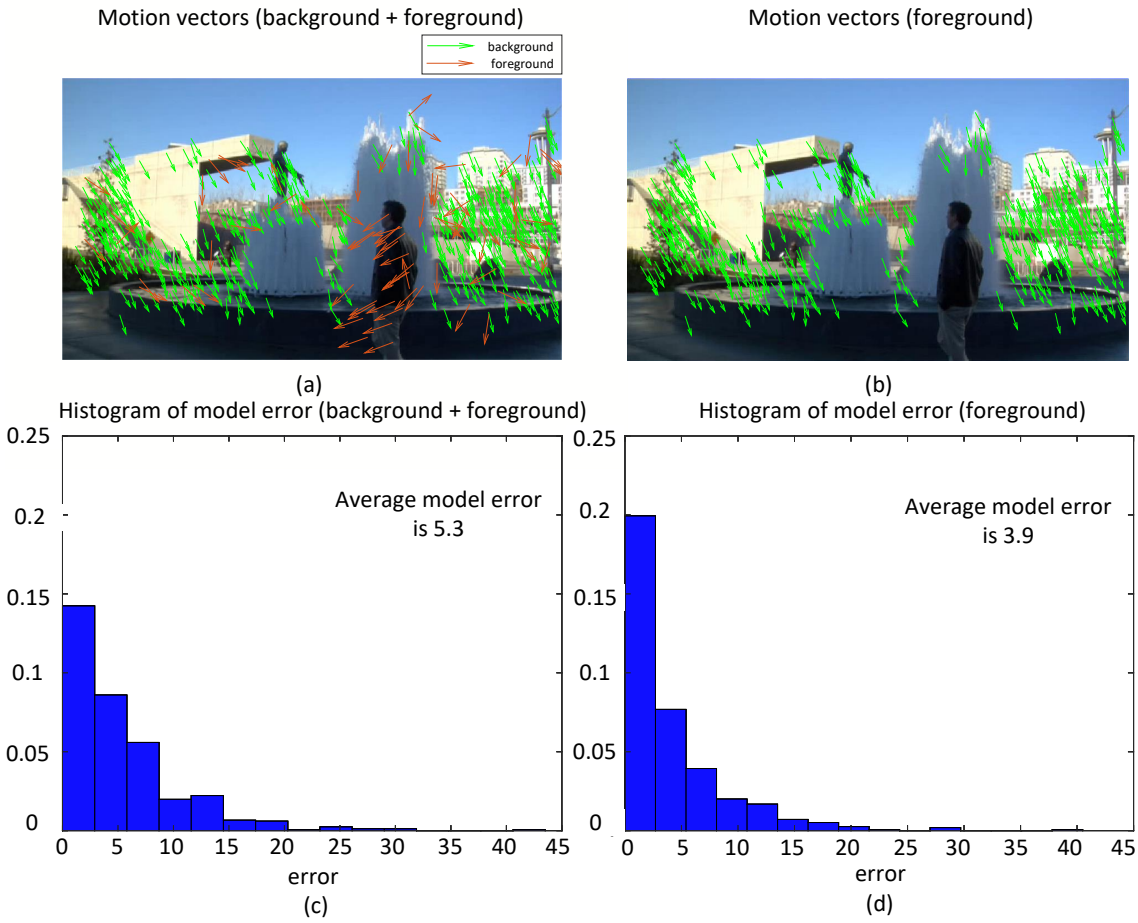


Figure 5.2: Match selection. (a)Motion vectors of feature points belonging to foreground moving and background static feature points. (b)Motion vectors of background only. (c) Histogram of model error estimated with background and foreground feature points and (d)with background feature points only

for all images is nearly impossible, a popular alternative is the ratio test proposed in [29]. Therefore, we use the ratio test to filter the matches returned by FLANN and pick the 1024 matches with the highest matching scores.

Scenes with moving foreground objects are always challenging to a video stabilization algorithm because the algorithm cannot distinguish whether the displacements of image contents in consecutive frames are caused by foreground object

movement or camera motion. For example, a camera can remain stationary while foreground objects actually move. Then, any non-stationary motion estimated by those moving feature matches would be erroneous. On the contrary, the movements of background objects in an image purely result from camera motion and are ideal for camera motion estimation. For a general scene where static background objects are relatively far away from the camera, background objects' 2D motions usually share a similar direction, as shown in Figure 5.2(a). Based on this observation, we first compute the 2D motion direction for each match by first subtracting the corresponding feature point coordinates and then normalizing the difference:

$$\mathbf{v} = \frac{\mathbf{I}_{\mathrm{next}} - \mathbf{I}_{\mathrm{cur}}}{\|\mathbf{I}_{\mathrm{next}} - \mathbf{I}_{\mathrm{cur}}\|_2} \tag{5.1}$$

where $\mathbf{I}_{\mathrm{cur}}$ is a feature point in the current frame, and $\mathbf{I}_{\mathrm{next}}$ is the corresponding feature point in the next frame, $\mathbf{v}$ is the normalized motion vector.

We filter out those matches whose directions are one standard deviation away from the average direction. As shown in Fig.5.2(b)(c)(d), this simple pre-processing step effectively eliminates most matches from foreground objects and results in a more accurate global camera model, where the model error is defined in equation (5.33).

In the next section, we will discuss that the parallel implementation of AC-RANSAC can only handle less than 1024 matches due to GPU's inherent hardware limitation, which makes match selection a necessary step. The third type of outliers can be effectively removed by robust estimation techniques such as RANSAC or AC-RANSAC.

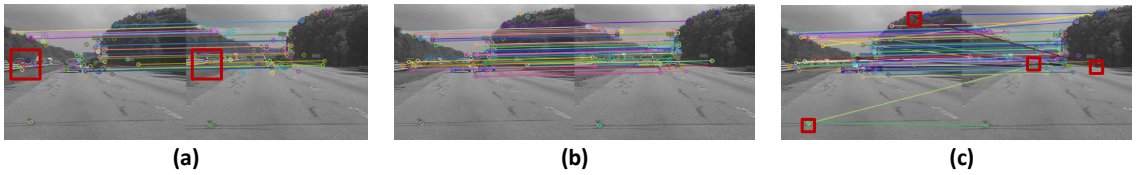


Figure 5.3: Robust Affine estimation. (a)If RANSAC error threshold ($\sigma = 0.05$) is small, good matches are filtered out. (b)AC-RANSAC: The error threshold is statistically computed, which provides a well balanced between the number of matches and the correctness. (c)Error threshold $\sigma = 0.3$. All the matches (including false matches) are returned.

5.2.2 Parallel AC-RANSAC

We choose our camera motion model to be the simplified affine transformation as it provides an excellent trade-off between effectiveness and complexity [26]:

$$A = \begin{bmatrix} s \cdot \cos(\theta) & -s \cdot \sin(\theta) & t_x \\ s \cdot \sin(\theta) & s \cdot \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

where s is the scale, θ is the rotation, and t_x and t_y are translations in x, y axis.

Feature matching plays a vital role in estimating A, but matches returned by matching algorithms are often outlier-contaminated. Therefore robust estimation techniques such as RANSAC [29] are required.

RANSAC randomly draws N_{sample} samples to estimate a temporary model A_{temp}. Then it counts the number of inliers according to a threshold σ(specified by the user) on the residual error. After N_{iter} iterations, the model with the largest number of inliers is returned. Finally, all the inliers are used to generate a final robust model. In our case, we need $N_{sample} = 2$ samples to estimate A_{temp}.

The correct choice of σ is critical but depends profoundly on the underlying data and model. If σ is too small, numerous true inliers are classified as outliers and eliminated. If it is too large, the algorithm treats many outliers as inliers, and an inaccurate model will be generated, as shown in Fig. 5.3.

AC-RANSAC [86, 87] avoids the problem of manually setting σ arising in

RANSAC by leveraging an *a contrario* criterion, which has been successfully applied to estimating Homography [88], Fundamental matrix [86], and Structure-from-Motion [10]. Here, we extend it to estimating affine model and apply it to affine model estimation and video stabilization.

The essence of AC-RANSAC is that an observed geometric event is significant if the expectation of its occurrences is minimal in a random image [87]. In our case, a subset of feature matches is significant to the model A_{temp} if the occurrence of this subset's matches compatible with A_{temp} is minimal, assuming all features and matches are independent and uniformly distributed. Hence, we can use the most meaningful subset of matches that complies with A_{temp} as the proxy of the largest consensus support subset of A_{temp}.

The meaningfulness of a set of feature matches is quantified by the expected number of false alarms (NFA), defined as [86, 87, 88]

$$NFA(k) = N_{outcome}(n - N_{sample})\binom{n}{k}\binom{k}{N_{sample}}\alpha^{k-N_{sample}} \tag{5.3}$$

- $N_{outcome}$ is the number of possible models (In the case of affine model, $N_{outcome} = 1$).
- n is the total number of feature matches.
- k is the number of inliers.
- α is the probability of feature matches being an inlier assuming they follow a uniform distribution

Indeed, if there are k inliers, a total of $\binom{k}{N_{sample}}$ inliers would generate A_{temp}. Under the null hypothesis that all data are independent and uniformly distributed, there are $\binom{n}{k}$ number of k tuples out of the total n feature matches. The number k of inliers is usually unknown, so all values of k (from $N_{sample} + 1$ to n) are tested, which gives rise to the factor $(n - N_{sample})$. The probability of all k matches are inliers is $\alpha^{k-N_{sample}}$ because the N_{sample} ones used to yield A_{tamp} have zero errors by default. For a rigorous proof of (2), we refer the reader to [86, 87].

Similar to [88], let α_0 be the ratio of the area a disk with radius 1 and the area of the image. Then α can be defined as

$$\alpha = \epsilon_k^2 \alpha_0 = \frac{\epsilon_k^2 \pi}{\text{image size}} \tag{5.4}$$

where ϵ_k is the k^{th} least error among all feature matches.

Therefore, the formula of the NFA for our simplified affine transformation is

$$\text{NFA}(k) = (n-2)\binom{n}{k}\binom{k}{2}\left(\frac{\epsilon_k^2 \pi}{\text{image size}}\right)^{k-2} \tag{5.5}$$

To find the largest consensus support subset of A_{temp} generated by $N_{sample} = 2$ random matches, we can first sort all the matches ascendingly by the residual error, then compute the $\text{NFA}(k)$ according to (4) for k from 3 to n (since the selected 2 random matches to generate A_{temp} have 0 error). Finally, the first k matches where k minimizes $\text{NFA}(k)$ are chosen to be the consensus support set of A_{temp}. The above procedure is summarized in Algorithm 2.

As opposed to RANSAC, without any user-specified thresholds, AC-RANSAC adaptively chooses the most meaningful set in the sense of generating the model as the largest consensus support set. However, AC-RANSAC requires sorting all data samples at each iteration. With a large number of iterations, AC-RANSAC becomes impractical. Due to this high demand for computational power and opaqueness in interpreting the *a-contrario* principle, AC-RANSAC has yet to become pervasive in many Computer Vision tasks. To our best knowledge, we are the first to apply AC-RANSAC to video stabilization.

Taking advantage of the parallel nature of AC-RANSAC, we propose a parallel optimization and real-time CUDA implementation of AC-RANSAC. The key idea is to process each AC-RANSAC iteration in parallel and combine the results at the last step. In the CUDA programming model, threads are organized into thread blocks, and grids hold thread blocks. On current GPUs, a thread block can contain no more than 1204 threads. Each thread has its private local memory, and each thread block has shared memory accessible to all threads of the block. The global,

Algorithm 2 Parallel AC-RANSAC for affine model

procedure PARALLEL AC-RANSAC(M, N_{iter})
▷ M is the list of matches
▷ N_{iter} is the number of iterations
    L_{ran} = Generate a list of $2N_{iter}$random numbers
    minNFA = ∞
    nBlocks = dim3($N_{iter}, 1, 1$)
    nThreads = dim3($32, 32, 1$)
    run AC-RANSAC-KERNEL with nBlocks and nThreads
    model = element in *models* with smallest NFA
end procedure
procedure AC-RANSAC-KERNEL($models$)
    Thread0 estimate affine model A_{temp}
    Each thread compute model error for each match
    Block-wise Radixsort for model errors
    Each thread computes NFA for each match
    Use reduction to find the model with min NFA in the block
end procedure

constant, and texture memory optimized for different memory usages are visible to all threads [89].

Specifically, each CUDA thread block is responsible for one AC-RANSAC iteration, and each thread in a thread block is used to estimate the error for each feature meach. The main steps are as follows:

1. Generate $2N_{iter}$ of random integers ranging from 1 to M in CPU host, where N_{iter} is the number of iteration, and M is the total number of feature matches.

2. Thread 0 of each thread block retrieves two random numbers and the corresponding feature matches from the constant memory in GPU, and then uses the two matches to estimate a simplified affine model A_{temp}.

3. Each thread retrieves a feature match according to its thread ID and computes the error to the model A_{temp} associated with the block.

4. The M number of errors computed in the previous step for each block is sorted ascendingly by blockwise RadixSort.

5. Each thread in a block computes $NFA(k)$, where k is the same as the thread ID. Reduction [90] is used to find the minimal NFA within a block. Thread 0 stores the minimal NFA for the block and A_{temp}.

6. Finally, the NFAs and their models are sent back to the CPU host, and CPU finds the minimal NFA and its corresponding model A_{temp} which is the final model.

The whole parallel AC-RANSAC is also shown in Algorithm 2, and the specific configuration of CUDA are detailed in the experiment section.

5.3 Least Squares-based Motion Stabilization

After estimating a robust affine model A, we can extract the motion parameters s, θ, t_x, t_y, from which we need to obtain a stable camera motion without annoying shakes and perturbation. There are multiple ways to approach this problem, such as low pass filter [22], Kalman filter [77, 31, 78], Gaussian Filter [21], Spline Smoothing [20], Motion Vector Integration [26] and so on. Here, we propose a novel least squares-based formulation for finding smooth camera motions obeying cinematography principles.

For ease of notation, from now on, we denote the motion parameter as m, which can be s, θ, t_x or t_y. We start by considering the problem of finding the stable camera motions as a least squares optimization:

$$\underset{\overline{m}_1,\dots\overline{m}_n}{\arg\min} \sum_{i=1}^{n} (m_i - \overline{m}_i)^2 + \lambda \sum_{i=2}^{n} (\overline{m}_i - \overline{m}_{i-1})^2 \tag{5.6}$$

where

- $m_i(\overline{m}_i)$ is the original(stabilized) camera motion for the i^{th} frame.

- n is the current frame number.

The data term $\sum_{i=1}^{n}(m_i - \overline{m}_i)^2$ ensures that the difference between the original and stabilized motions is small in order to reduce the distortion introduced in warping. The regularization term $\sum_{i=2}^{n}(\overline{m}_i - \overline{m}_{i-1})^2$ guarantees that the estimated camera motions $(\overline{m}_1, \ldots, \overline{m}_n)$ are stable. The regularization parameter λ controls the degree of motion smoothness and the tracking ability of the stable camera motions $(\overline{m}_1, \ldots, \overline{m}_n)$

This formulation and its variants are used in offline video stabilization [25, 27] and yield good results. However, (5.6) in [25, 27] operates in a batch fashion that computes all the stable motions at once offline. Next, we propose three new formulations based on (5.6) that robustly estimate the current stabilized motion online in real-time.

5.3.1 A direct approach

For (5.6) to be applicable to an online and real-time manner, a direct modification is

$$\overline{m}_n = \arg\min_{\overline{m}_n} \sum_{i=1}^{n-1}(m_i - \widehat{m}_i)^2 + (m_n - \overline{m}_n)^2 + \lambda \sum_{i=2}^{n}(\widehat{m}_i - \widehat{m}_{i-1})^2 + \lambda(\overline{m}_n - \widehat{m}_{n-1})^2 \tag{5.7}$$

where we try to find the current estimate $\overline{m}_n$ using all initial estimates $m_1, \ldots, m_n$ and all previous stabilized results $\widehat{m}_1, \ldots, \widehat{m}_{n-1}$. We can further simplify (5.7) to:

$$\overline{m}_n = \arg\min_{\overline{m}_n}(m_n - \overline{m}_n)^2 + \lambda(\overline{m}_n - \widehat{m}_{n-1})^2 \tag{5.8}$$

Since (5.8) is quadratic, the minimum occurs when the derivative is zero. The solution to (5.8) is

$$\overline{m}_n = \frac{m_n + \lambda \widehat{m}_{n-1}}{1 + \lambda} \tag{5.9}$$

However, (5.9) is simply the weighted average of the last stabilized motion $\widehat{m}_{n-1}$ and the current motion m_n, where the weight is controlled by λ. Hence, (5.9) has a

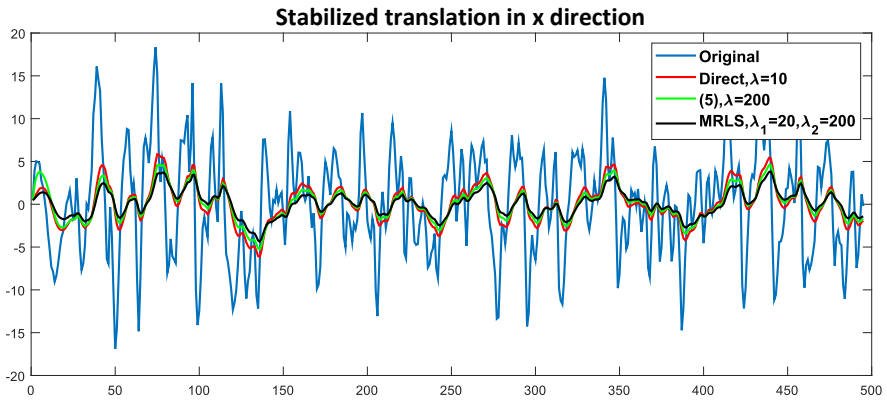
limited capability of stabilizing the current frame, as shown in Fig.5.4.

5.3.2 Modified Recursive Least Squares Stabilization (MRLS)

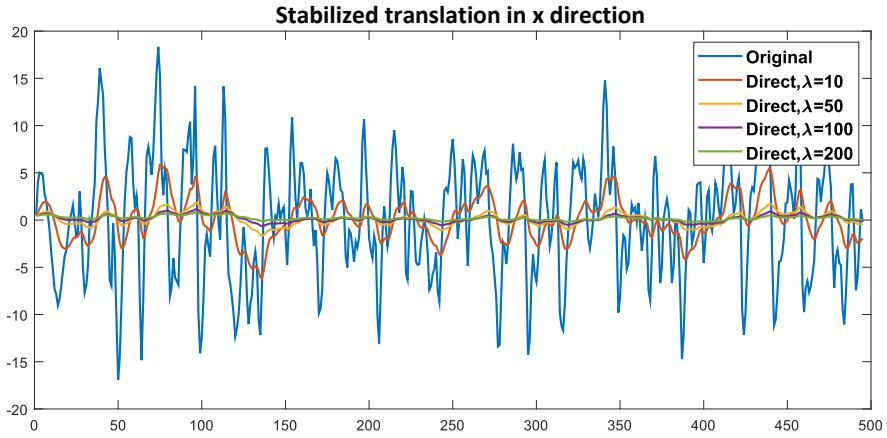
The direct approach provides a formulation that runs online and in real-time but with a limited stabilization ability. Although (5.6) operates in a batch fashion, it is still suitable for real-time stabilization if we can efficiently and robustly estimate the current stabilized motion.

At the n^{th} frame, suppose we compute $(\overline{m}_1, \ldots, \overline{m}_n)$ in (5.6) in real-time, then we can use $\overline{m}_n$ as the stabilized motion for current frame, i.e. $\hat{m}_n = \overline{m}_n$. However, (5.6) does not take into account the past stabilized motions $(\hat{m}_1, \ldots, \hat{m}_{n-1})$ when computing $\overline{m}_n$, hence there is no guarantee that $(\hat{m}_1, \ldots, \hat{m}_{n-1}, \hat{m}_n = \overline{m}_n)$ will form a smooth path. To remedy this, we further propose the following cost function:

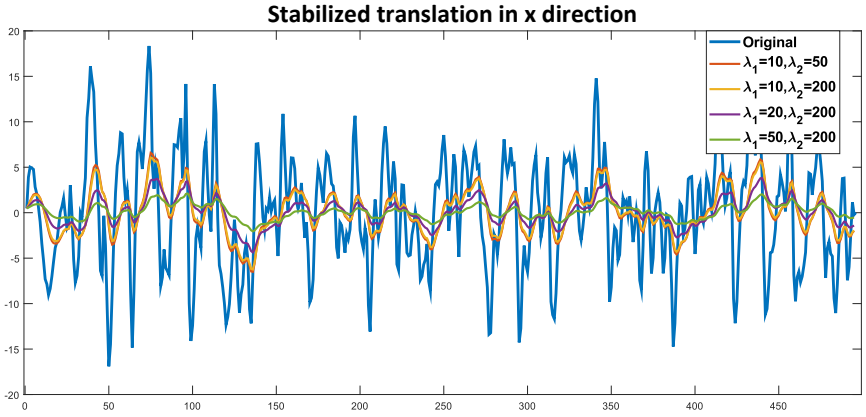
$$\underset{\overline{m}_1, \ldots \overline{m}_n}{\arg\min} \sum_{i=1}^{n} (m_i - \overline{m}_i)^2 + \lambda_1 \sum_{i=2}^{n} (\overline{m}_i - \overline{m}_{i-1})^2 + \lambda_2 \sum_{i=1}^{n-1} (\overline{m}_i - \hat{m}_i)^2 \tag{5.10}$$



(a)



(b)



(c)

Figure 5.4: Stabilization of translation in x direction, i.e. $m = t_x$, via the original approach (5.6), the direct approach (5.7) and MRLS (5.10). (a) Comparison of stabilization effects of the three formulae. (b) The direct approach has limited freedom to stabilize the motions. As λ increases, it can over-stabilize the original motions resulting in a static, instead of stabilized, video. (c) MRLS makes use of all the previous estimates for stabilization and avoids over-stabilizing the motions. The stabilized motions can still reflect the intentional motion of the camera.

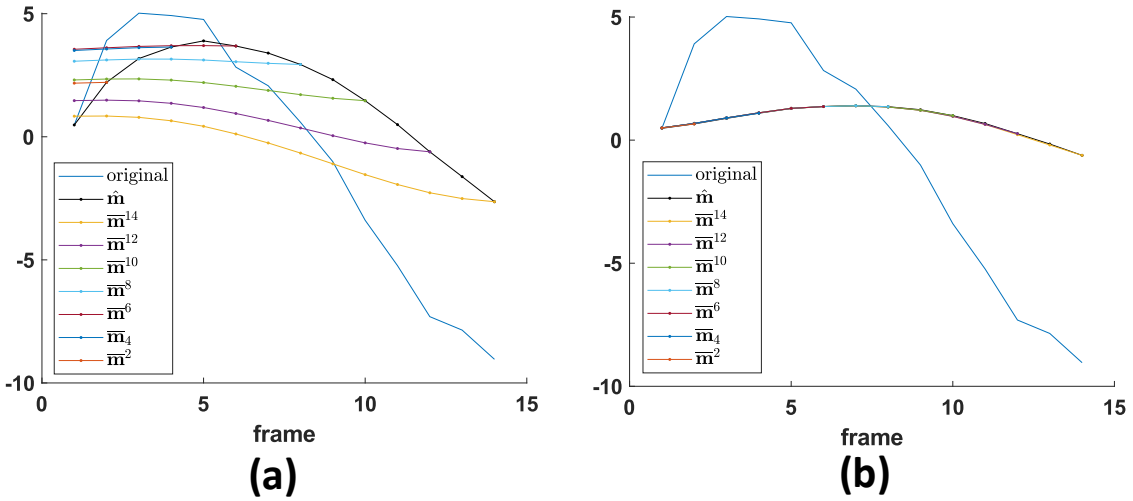


Figure 5.5: Stabilization of translation in x direction, i.e. $m = t_x$. $\hat{\mathbf{m}}$ denotes final stabilized estimates. $\overline{\mathbf{m}}^k$ denotes the optimal solution $(\overline{m}_1, \ldots, \overline{m}_k)$ to (5.6) for (a) or (5.10) at frame k for (b). (a) At each frame, (5.6) tries to estimate an augmented path $(\overline{m}_1, \ldots, \overline{m}_k)$ without considering the previous estimate $(\widehat{m}_1, \ldots, \widehat{m}_{k-1})$. (b) (5.10) assures that $\widehat{m}_i$ and $\overline{m}_i, i = 1, \ldots, n-1$ are similar, so $(\widehat{m}_1, \ldots, \widehat{m}_{k-1}, \widehat{m}_k = \overline{m}_k)$ will be smooth.

$$\underbrace{\begin{bmatrix} (1+\lambda_1+\lambda_2) & -\lambda_1 & 0 & \cdots & \cdots & 0 \\ -\lambda_1 & (1+2\lambda_1+\lambda_2) & -\lambda_1 & \ddots & \ddots & \vdots \\ 0 & -\lambda_1 & (1+2\lambda_1+\lambda_2) & -\lambda_1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -\lambda_1 & (1+2\lambda_1+\lambda_2) & -\lambda_1 \\ 0 & \cdots & \cdots & 0 & -\lambda_1 & (1+\lambda_1) \end{bmatrix}}_{\mathbf{S}_n} \underbrace{\begin{bmatrix} \overline{m}_1 \\ \overline{m}_2 \\ \overline{m}_3 \\ \vdots \\ \overline{m}_{n-1} \\ \overline{m}_n \end{bmatrix}}_{\overline{\mathbf{m}}_n} = \underbrace{\begin{bmatrix} m_1 + \lambda_2 \widehat{m}_1 \\ m_2 + \lambda_2 \widehat{m}_2 \\ m_3 + \lambda_2 \widehat{m}_3 \\ \vdots \\ m_{n-1} + \lambda_2 \widehat{m}_{n-1} \\ m_n \end{bmatrix}}_{\mathbf{Y}_n} \tag{5.11}$$

Similar to (5.6), here (5.10) tries to estimate an augmented path $(\overline{m}_1, \ldots, \overline{m}_n)$ that starts at the first frame and ends at the current frame, and $\widehat{m}_n$ is set to $\overline{m}_n$ for the next round of estimation.

The data term $\sum_{i=1}^{n}(m_i - \overline{m}_i)^2$ ensures that the distortion between the original path $(m_1, \ldots, m_n)$ and the augmented path is small. The augmented path is guaranteed to be smooth by the minimization of the first regularization term $\lambda_1 \sum_{i=2}^{n}(\overline{m}_i - \overline{m}_{i-1})^2$. Furthermore, the second regularization term $\lambda_2 \sum_{i=1}^{n-1}(\overline{m}_i - \widehat{m}_{i-1})^2$ secures that the augmented path is similar to the previous smoothed path $(\widehat{m}_1, \ldots, \widehat{m}_{n-1})$, making sure that $(\widehat{m}_1, \ldots, \widehat{m}_{n-1}, \widehat{m}_n = \overline{m}_n)$ will form a smooth curve, as shown in Fig. 5.5.

Similarly, (5.10) is quadratic, so we can find the optimal augmented path $(\overline{m}_1, \ldots, \overline{m}_n)$ to achieve the minimum of (5.10) by setting all partial derivatives to zero, which is equivalent to solve the system of linear equations (5.11). The derivation for (5.11) is detailed in Appendix 5.9. Note that (5.11) can be written compactly in matrix form $\mathbf{S}_n\overline{\mathbf{m}}_n = \mathbf{Y}_n$.

For small $\mathbf{S}_n$, we could just invert it to find $\widehat{m}_n$, i.e. $\widehat{m}_n = \overline{m}_n = \left[\mathbf{S}_n^{-1}\mathbf{Y}_n\right]_n$, where $[\cdot]_i$ denotes i^{th} element of a vector. However, the size of $\mathbf{S}_n$ increases with n, which makes computing $\mathbf{S}^{-1}$ not practical to a real-time application for large n. Since $\widehat{m}_n$ is the quantity that we try to estimate, we do not have to compute the complete inverse of $\mathbf{S}_n$. Instead, knowing the last row of $\mathbf{S}_n^{-1}$ is enough for computing $\widehat{m}_n$. Thanks to the unique structure of $\mathbf{S}_n$, we derive a recursive formula for computing the last row of $\mathbf{S}_n^{-1}$ by recursively constructing the row echelon form of $\mathbf{S}_n$ from $\mathbf{S}_{n-1}$ in $O(n)$ operations.

The key observations are (1) that the $(n-2)^{th}$ row of $\mathbf{S}_{n-1}$'s echelon form is the same as $\mathbf{S}_n$'s echelon form and (2) that the operations to derive the last row of $\mathbf{S}_n$'s echelon form from the $(n-2)^{th}$ row is straightforward.

Theorem 5.1. *Let* $\mathcal{S}_{n-1} = [S_{n-1}|I]$ *be an augmented matrix, where* I *is the identity matrix.* $\mathcal{S}_{n-1}^e = [S_{n-1}^e|A_{n-1}]$ *is the echelon form of* $\mathcal{S}_{n-1}$ *with* $b = -\lambda_1$ *as leading term for each*

row. Then $\{S_{n-1}\}_{n-2} \in \mathbb{R}^{2(n-1)}$ *and* $\{S_n\}_{n-2} \in \mathbb{R}^{2n}$ *have the form:*

$$\{\mathcal{S}_{n-1}\}_{n-2} = \begin{bmatrix} 0 & \cdots & b & x_1^{(n-1)} & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 \end{bmatrix} \tag{5.12}$$

$$\{\mathcal{S}_n\}_{n-2} = \begin{bmatrix} 0 & \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \end{bmatrix} \tag{5.13}$$

where $\{\cdot\}_i$ *denotes the* i^{th} *row of a matrix, and*

$$x_1^{(n-1)} = \frac{b^2}{c - x_1^{(n-2)}} \tag{5.14}$$

$$y_i^{(n-1)} = \frac{-y_i^{(n-2)} b}{c - x_1^{(n-2)}}, \quad i = 1 \dots n-3 \tag{5.15}$$

$$y_{n-2}^{(n-1)} = \frac{b}{c - x_1^{(n-2)}} \tag{5.16}$$

$$\tag{5.17}$$

where

$$a = (1 + \lambda_1 + \lambda_2) \tag{5.18}$$

$$b = -\lambda_1 \tag{5.19}$$

$$c = (1 + 2\lambda_1 + \lambda_2) \tag{5.20}$$

$$d = (\lambda_1 + 1) \tag{5.21}$$

Proof. See Appendix 5.9

□

Once we have $\{\mathcal{S}_n\}_{n-2}$, we can obtain the $\{\mathcal{S}_n\}_n$ by two more row operations:

$$\frac{b^2}{c - x_1^{(n-1)}} \left(\{\mathcal{S}_n\}_{n-1} - \{\mathcal{S}_n\}_{n-2}\right) \rightarrow \{\mathcal{S}_n\}_{n-1} \tag{5.22}$$

$$\frac{c - x_1^{(n-1)}}{q_n} \left(\{\mathcal{S}_n\}_n - \{\mathcal{S}_n\}_{n-1}\right) \rightarrow \{\mathcal{S}_n\}_n \tag{5.23}$$

where $q_n = cd - x_1^{(n-1)}d - b^2$. Note that the first of half of $\{\mathcal{S}_n\}_n$, denoted as $\{\mathcal{S}_n\}_n^{1st}$, is a unity row vector with the last element being 1, and its second half, denoted as $\{\mathcal{S}_n\}_n^{2nd}$, is $\{\mathbf{S}_n^{-1}\}_n$ (the last row of $\mathbf{S}_n^{-1}$), which can be used to find $\widehat{m}_n$:

$$\widehat{m}_n = \overline{m}_n = \{\mathbf{S}_n^{-1}\}_n \cdot \mathbf{Y}_n = \{\mathcal{S}_n\}_n^{2nd} \cdot \mathbf{Y}_n \tag{5.24}$$

To estimate $\widehat{m}_n$ recursively, we will first need to store $\{S_{n-1}\}_{n-2}$ and $\mathbf{Y}_n$ which takes $O(n)$ storage, then use (5.22),(5.23) to compute $\widehat{m}_n$ which takes $O(n)$ operations, and finally store $\{S_n\}_{n-1}$ for estimating $\widehat{m}_{n+1}$ which also takes $O(n)$ storage. Therefore, solving (5.11) has $O(n)$ time and space complexity in total.

Note that (5.11) differs from a typical recursive least squares [91] in that $\mathbf{S}_n$, $\overline{\mathbf{m}}_n$ and $\mathbf{Y}_n$ change at each time step. Hence we refer (5.11) as the modified recursive least square(MRLS) in our algorithm pipeline.

5.3.3 Cinematography Principles guided Modified Recursive Least Squares Stabilization (C-MRLS)

MRLS assures that the estimated path is smooth and close to the original path by adding the two regularization terms. However, according to cinematography principles [22, 24], the desired stabilized path should have constant velocity and constant accelerations, i.e., the second and third derivatives for the path are zero. For our stabilized path to have these cinematographic characteristics, we first define two proxies as the variations of velocity and acceleration.

From finite difference methods [92], the backward difference approximation

for the second and the third derivatives are[92]

$$(dv)_i = \overline{m}_i - 2\overline{m}_{i-1} + \overline{m}_{i-2} \tag{5.25}$$

$$(dc)_i = \overline{m}_i - 3\overline{m}_{i-1} + 3\overline{m}_{i-2} - \overline{m}_{i-3} \tag{5.26}$$

Note that (5.25) and (5.26) have first-order truncation error[92]. We could also choose other more complex forms for approximating dv and dc with smaller truncation errors in principle.

Hence, the desired cost function following the cinematography principle is

$$\begin{aligned} \underset{\overline{m}_1,\ldots\overline{m}_n}{\arg\min} & \sum_{i=1}^{n}(m_i - \overline{m}_i)^2 + \lambda_1 \sum_{i=2}^{n}(\overline{m}_i - \overline{m}_{i-1})^2 + \lambda_2 \sum_{i=1}^{n-1}(\overline{m}_i - \widehat{m}_i)^2 \\ & + \lambda_3 \sum_{i=3}^{n}(dv)_i^2 + \lambda_4 \sum_{i=4}^{n}(dc)_i^2 \end{aligned} \tag{5.27}$$

The path that minimizes (5.27) will be smooth and have near-constant velocity and near-constant acceleration, which obeys the cinematography principles.

Since (5.27) is also quadratic, we can find the solution by solving the corresponding system of linear equations whose data matrix is (5.28). The derivation for (5.28) and its linear solver are the same as (5.11). Hence we refer (5.28) as the cinematography principles guided modified recursive least square(C-MRLS). As shown in Fig. 5.6, C-MRLS avoids rapid changes in the resulting path, and thus the result is smoother than that of MRLS but well still preserves the intentional camera global motion.

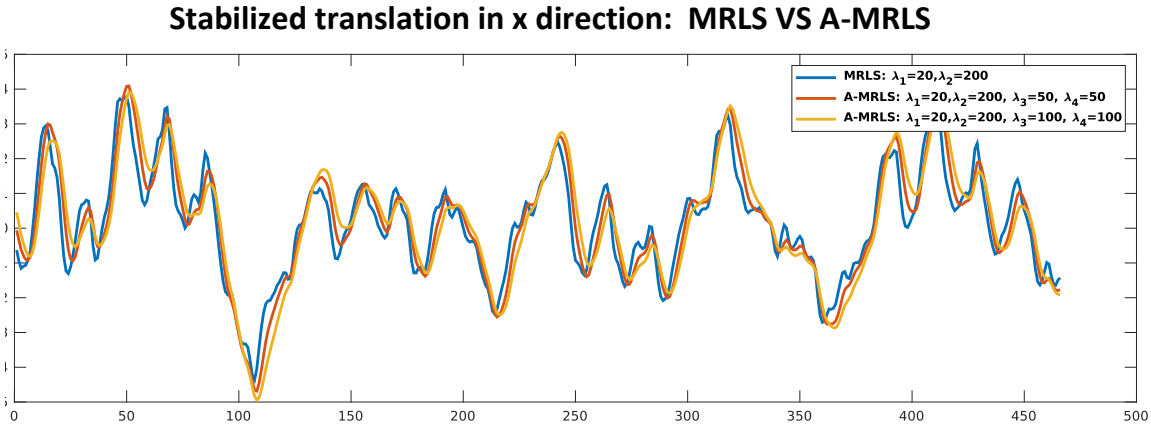


Figure 5.6: Comparison of MRLS and C-MRLS in translation in x direction, i.e. $m = t_x$. The result for C-MRLS is smoother than MRLS but well preserves the intentional camera global motion.

$$\underbrace{\begin{bmatrix}
(1+\lambda_1+\lambda_2+\lambda_3+\lambda_4) & -(\lambda_1+\lambda_3+3\lambda_4) & \lambda_3+3\lambda_4 & \lambda_4 & \cdots & \cdots & \cdots & 0 \\
-(\lambda_1+\lambda_3+\lambda_4) & (1+2\lambda_1+\lambda_2+3\lambda_3+4\lambda_4) & -(\lambda_1+3\lambda_3+6\lambda_4) & \lambda_3+4\lambda_4 & -\lambda_4 & \cdots & \cdots & 0 \\
(\lambda_3+\lambda_4) & -(\lambda_1+3\lambda_3+4\lambda_4) & (1+2\lambda_1+\lambda_2+4\lambda_3+7\lambda_4) & -(\lambda_1+3\lambda_3+7\lambda_4) & (\lambda_3+4\lambda_4) & -\lambda_4 & \cdots & 0 \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\
0 & \cdots & (\lambda_3+\lambda_4) & -(\lambda_1+3\lambda_3+4\lambda_4) & (1+2\lambda_1+\lambda_2+4\lambda_3+7\lambda_4) & -(\lambda_1+3\lambda_3+7\lambda_4) & (\lambda_3+4\lambda_4) & -\lambda_4 \\
0 & \cdots & \cdots & -\lambda_4 & \lambda_3+4\lambda_4 & -(\lambda_1+3\lambda_3+6\lambda_4) & (1+2\lambda_1+\lambda_2+3\lambda_3+4\lambda_4) & -(\lambda_1+\lambda_3+\lambda_4) \\
0 & \cdots & \cdots & \cdots & \lambda_4 & \lambda_3+3\lambda_4 & -(\lambda_1+\lambda_3+3\lambda_4) & (1+\lambda_1+\lambda_2+\lambda_3+\lambda_4)
\end{bmatrix}}_{\mathbf{S_n}}
\underbrace{\begin{bmatrix}
\overline{m}_1 \\ \overline{m}_2 \\ \overline{m}_3 \\ \vdots \\ \overline{m}_{n-1} \\ \overline{m}_n
\end{bmatrix}}_{\overline{\mathbf{m}_n}} = \mathbf{\Upsilon_n}$$

(5.28)

5.3.4 Modified Sliding Window Least Squares Stabilization (MSWLS)

In MRLS and C-MRLS, all previous frames are used to estimate the stable motion, which takes $O(n)$ time and space complexity. As n increases, MRLS and C-MRLS will eventually become impractical. In practice, the frames far before the current frame contain little information for the inference of the current frame's stable motion. Hence, we propose to use only the latest N frames to estimate the motion of the current frame, where N is specified by users:

$$\begin{aligned}\underset{\overline{m}_{n-N+1},\dots\overline{m}_n}{\arg\min} & \sum_{i=n-N+1}^{n}(m_i-\overline{m}_i)^2+\lambda_1\sum_{i=n-N+2}^{n}(\overline{m}_i-\overline{m}_{i-1})^2 \\ & +\lambda_2\sum_{i=n-N+1}^{n-1}(\overline{m}_i-\widehat{m}_i)^2+\lambda_3\sum_{i=n-N+1}^{n}(dv)_i^2+\lambda_4\sum_{i=n-N+1}^{n}(dc)_i^2\end{aligned} \tag{5.29}$$

(5.29) not only emphasizes the most relevant information in the time domain but also can be regarded as a failsafe for MRLS. Similarly, to find the $\overline{\mathbf{m}}$ that minimizes (5.10), we can set all the partial derivatives to zero, which is equivalent to solving the linear system:

$$\mathbf{S}_N\cdot\underbrace{\begin{bmatrix}\overline{m}_{n-N+1}\\ \overline{m}_2\\ \vdots\\ \overline{m}_{n-1}\\ \overline{m}_n\end{bmatrix}}_{\overline{\mathbf{m}}'_N}=\underbrace{\begin{bmatrix}m_{n-N+1}+\lambda_2\widehat{m}_{n-N+1}\\ m_{n-N+2}+\lambda_2\widehat{m}_{n-N+2}\\ \vdots\\ m_{n-1}+\lambda_2\widehat{m}_{n-1}\\ m_n\end{bmatrix}}_{\mathbf{Y}_N} \tag{5.30}$$

$$\widehat{m}_n=\overline{m}_n=\{\mathbf{S}_N^{-1}\}_N\mathbf{Y}_N \tag{5.31}$$

where $\{\mathbf{S}_N^{-1}\}_N$ can be precomputed or set to $\{\delta_n\}_n^{2nd}$ when $n=N$ in MRLS, and N is specified by users.

We see that saving $\mathbf{Y}_N$ requires $O(N)$ storage and computing $\widehat{m}_n$ by (5.31) takes

$O(N)$ operation, so MSWLS also has $O(N)$ time and space complexity.For similar reason, we refer (5.30) as the modified sliding window least square(MSWLS) in our algorithm pipeline. Our least squares motion stabilization is summarized in Algorithm 3.

Algorithm 3 Least Squares-based Motion Stabilization

procedure C-MRLS($m, n, \mathbf{Y}, \mathcal{S}$)
  append m_n to $\mathbf{Y}$
  construct $\mathcal{S}$ in (5.13) from (5.12) ▷ differs from MSWLS
  estimate $\widehat{m}_n$
  set the last element of $(m_n + \lambda_2 \widehat{m}_n)$
  return $\widehat{m}, \mathbf{Y}, \mathcal{S}$
end procedure
procedure MSWLS($m, n, \mathbf{Y}$)
  append m_n to $\mathbf{Y}$
  estimate $\widehat{m}_n$ using (5.31) ▷ differs from C-MRLS
  remove $m_{n-N+1} + \lambda_2 \widehat{m}_{n-N+1}$ from $\mathbf{Y}$
  set the last element of $(m_n + \lambda_2 \widehat{m}_n)$
  return $\widehat{m}, \mathbf{Y}$
end procedure
procedure LS-STABILIZE($m, n, \mathbf{Y}, \mathcal{S}$)
  if $n < 1$ **then**
    $\widehat{m} = m_n$
    append $m_1 + \lambda_2 \widehat{m}_1$ to $\mathbf{Y}$
  else if $n < N$ **then**
    $[\widehat{m}, \mathbf{Y}, \mathcal{S}] =$ C-MRLS$(m, n, \mathbf{Y}, \mathcal{S})$
  else
    $[\widehat{m}, \mathbf{Y}] =$ MSWLS$(m, n, \mathbf{Y})$
  end if
  return $\widehat{m}, \mathbf{Y}, \mathcal{S}$
end procedure

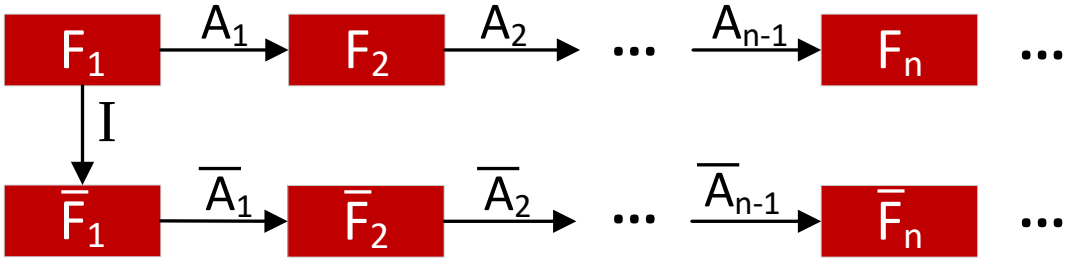


Figure 5.7: Global transformation relationship between original F_i and stable $\bar{F}_i$ frames

5.4 Motion Compensation

After the stabilized motions are estimated, we need to warp the current frame to generate the final stabilized frame. Let $F_i, \bar{F}_i$ denote the i^{th} original frame and stabilized frame, A_i denote the simplified affine between F_i and, F_{i+1}, $\bar{A}_i$ denote the stabilized affine transformation between $\bar{F}_i$ and $\bar{F}_{i+1}$. From Fig. 5.7, we see that F_n relates F_1 with

$$F_n = \left(\prod_{i=1}^{n} A_i\right) F_1 = (A_n \cdot \ldots \cdot A_1)\, F_1$$

and $\bar{F}_n$ relates $\bar{F}_1$ with

$$\bar{F}_n = \left(\prod_{i=1}^{n} \bar{A}_i\right) \bar{F}_1 = (\bar{A}_n \cdot \ldots \cdot \bar{A}_1)\, \bar{F}_1$$

Therefore, the n^{th} stabilized frame $\bar{F}_n$ can be obtained from the current input F_n by

$$\begin{aligned} \bar{F}_n &= \left(\prod_{i=1}^{n} \bar{A}_i\right) \cdot \left(\prod_{i=1}^{n} A_i\right)^{-1} F_n \\ &= (\bar{A}_n \cdot \ldots \cdot \bar{A}_1) \cdot (A_1^{-1} \cdot \ldots \cdot A_n^{-1})\, F_n \end{aligned} \tag{5.32}$$

5.5 Experiment and Result

We implement our algorithm on a PC with a Core i7-6770 4.0 GHz CPU and a GeForce GTX2080 Ti GPU. SURF and FLANN are based on the GPU implementation of OpenCV. We implement both Parallel AC-RANSAC and RANSAC on GPU with CUDA, and the rest is run on CPU. Table 5.1 presents the timing profile for the proposed algorithm. For videos with resolutions of 1920×1080 or higher, we downsample them to 1280×720 before further processing.

Table 5.1: Timing Performance of the proposed algorithm in milliseconds (Total time is computed with parallel AC-RANSAC)

Resolution	SURF and FLANN	Parallel AC-RANSAC	AC-RANSAC	Motion Smoothing	Total
320×240	6.83	3.19	79.75	0.14	12.59
640×360	8.48	5.10	128.52	0.56	16.21
1280×720	17.00	7.28	181.27	1.04	31.67
1920×1080	17.10	7.30	183.23	1.25	32.80

5.5.1 Feature Detection, Matching, and Match Selection

We use SURF and FLANN as our feature detection and matching method, as they provide a fair tradeoff of robustness and efficiency [34, 35]. We use OpenCV's parallel implementation for SURF and FLANN on GPU. As we can see from Table 5.1, SURF and FLANN are the most time-consuming compared to other parts because of the computation of SURF features and the descriptors. Since OpenCV's GPU implementation of SURF is at the pixel level, the number of pixels easily dominates the number of CUDA cores in a GPU, which diminishes the gain of parallelism when the image resolution is high.

5.5.2 RANSAC vs AC-RANSAC

Compared to RANSAC, AC-RANSAC automatically selects thresholds for inlier/outlier discrimination, but the extra computation of AC-RANSAC prevents it from

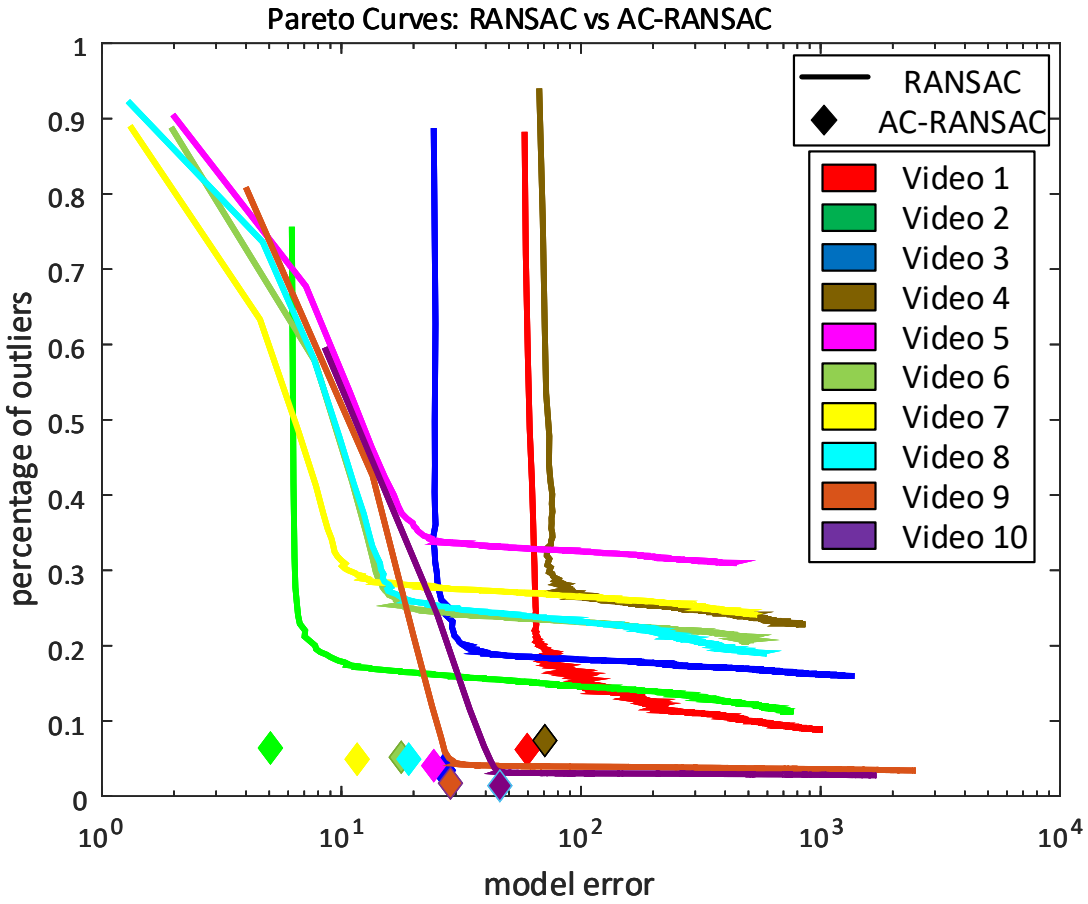


Figure 5.8: The threshold for RANSAC ranges from 0.001 to 0.3. Each data point in RANSAC is the result of a run with a threshold value. For all the 10 testing data, AC-RANSAC computes a model with more inliers (less outliers in the plot) and smaller model error.

being used in real-time applications. We hence propose the parallel AC-RANSAC implemented with CUDA on GPU. To further reduce the processing time, we massively utilize the on-chip shared memory. In our implementation, each thread inside a thread block independently estimates the model error for a match. Since a maximum of 1024 threads is allowed in a thread block in GPU, only consecutive frames that generate less than 1024 matches can be handled inherently, which is usually valid for videos with less than 1920x1080 pixels. Otherwise, as described in 5.2.1, the matches selection method is applied to choose the most 1024 robust

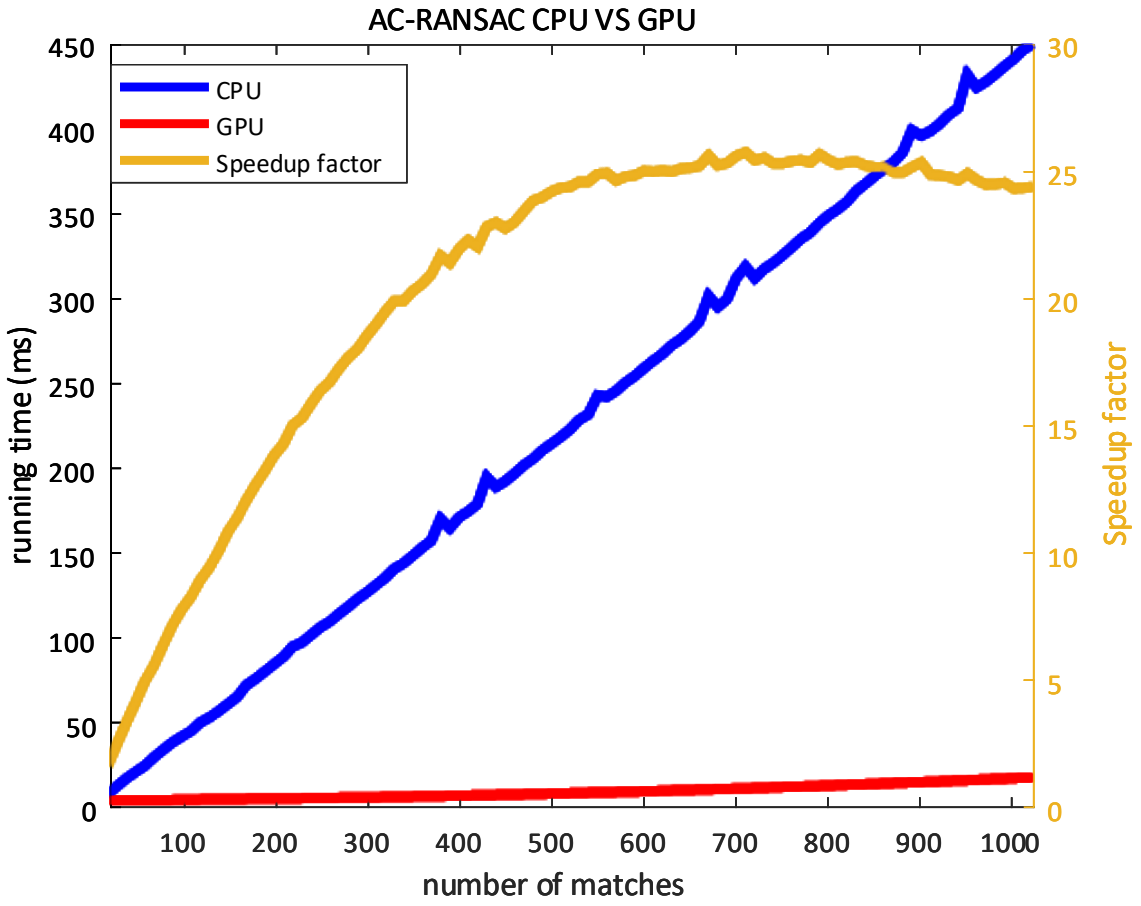


Figure 5.9: Timing performance between GPU and CPU implementation of AC-RANSAC as number of matches increases. The result is the average of three trials.

matches for AC-RANSAC.

For challenging frames that do not have enough feature points or undergo significant non-ridge motion, etc., AC-RANSAC (or RANSAC) will not be able to estimate a robust affine/homography model that well explains the geometric transformation between the frames. In such cases, the model returned by AC-RANSAC (or RANSAC) is ill-conditioned. Hence, we discard the estimated simplified affine transformation and set it to the identity matrix if the smallest singular value of it is less than 0.001.

As discussed in 5.2.2, the drawback of AC-RANSAC is the high computational complexity compared to RANSAC. As shown in Fig.5.9, our parallel implementation achieves 25X speed up and around 20 ms for the 1024 matches. We use the number of inliers and model errors to assess AC-RANSAC's and RANSAC's effectiveness. We define the model error as

$$e = \max\left(A\mathbf{I}_l - \mathbf{I}_r, \mathbf{I}_l - A^{-1}\mathbf{I}_r\right) \tag{5.33}$$

where A is the estimated affine model, $\mathbf{I}_l$ is the feature point in the current frame, and $\mathbf{I}_r$ is the matched feature point in the next frame.

For RANSAC, we range the discrimination threshold from 0.001 to 0.5 and plot the number of inliers and model errors for each run. In AC-RANSAC, the inliers are the matches whose evaluated errors are smaller than the match that achieves minNFA. Since AC-RANSAC doesn't require a threshold, there is only one data point for each video for AC-RANSAC. As shown in Fig.5.8, AC-RANSAC results in a model that has more inliers and a smaller model error than RANSAC.

5.5.3 Stabilization

We compare three real-time [30, 78, 93] and three offline video stabilization algorithms [21, 24, 94] with the proposed algorithm, and further quantify two essential aspects (distortion and smoothness) of stabilization quality with two objectives metrics.

We use Inter-frame Transformation Fidelity (ITF) to compute the distortion introduced by stabilization algorithms. ITF is a popular evaluation metric of stabilization quality, which is defined as

$$\mathrm{ITF} = \frac{1}{N-1}\sum_{k=1}^{N-1}\mathrm{PSNR}(F_{k+1}, F_k) \tag{5.34}$$

where N is the total number of frames and

$$\mathrm{PSNR}(F_{k+1}, F_k) = 10\log_{10}\left(\frac{peakval^2}{MSE(F_{k+1}, F_k)}\right) \tag{5.35}$$

is the peak signal-to-noise ration between the two consecutive frames with $peakval$ being the maximum pixel value and $MSE(F_{k+1}, F_k)$ the mean square error between consecutive frames F_k and F_{k+1}.

A higher value of ITF indicates smaller average inter-frame distortion across the video and thus better stabilization quality. We compare the proposed algorithm with other state-of-the-art stabilization algorithms in terms of ITF. Since [21, 78] use inpainting to fill the missing borders of the stabilized frame, for a fair comparison, we crop all the stabilized video by 20 pixels before computing ITF for the stabilized videos produced by other algorithms.

We use the normalized decrease in feature point acceleration to measure the smoothness of a stabilized video,, as feature points in a smooth video should have zero or constant acceleration [24]. [78] adopts such an idea to assess the smoothness of a stabilized video. For each output video, we first extract SURF features for each frame and perform matching to build feature trajectories. Then we compute the sum of acceleration for each trajectory by

$$V(I) = \sum \sqrt{\left(I_{i+1}^x - 2I_i^x + I_{i-1}^x\right)^2 + \left(I_{i+1}^y - 2I_i^y - I_{i-1}^y\right)^2} \tag{5.36}$$

where (I_i^x, I_i^y) is the image coordinate of feature point I at the i^{th} frame. Finally, the smoothness can be calculated as the average normalized decrease in trajectory acceleration [78]:

$$smoothness = \frac{1}{N} \sum_{k=1} \frac{|V_k^o - V_k^s|}{V_k^o} \tag{5.37}$$

where V_k^o is the sum of acceleration of the trajectory in the original video, and V_k^s is the sum of acceleration in the stabilized video. An output video with higher smoothness is superior.

We use the OpenCV implementations for [21, 24], the official implementation for [94] provided by *Adobe After Effect*, and the original implementation or software available in GitHub for [30, 78, 93] provided by the authors. All algorithms are run on public data set available in [24, 93, 94] and compared in terms of distortion and smoothness. We compared LSstab with state-of-the-art real-time algorithms [30, 78, 93] and offline algorithms [24, 94, 21] for videos with different resolutions.

Because the undefined borders of the stabilized outputs for the algorithms above [21, 24, 94, 93, 30, 78] are removed or inpainted, we remove 10% border, i.e., 0.9 cropping ratio, before comparing the results.

As shown in Table 5.2, the proposed method does the best in minimizing distortion, and [93] does the best in encouraging smoothness. However, the processing time for [93] grows rapidly as video resolution increases, even after we decrease the grid resolution to 0.8 of the default value. As shown in Fig. 5.11, [93, 78] and the proposed algorithm are competitive to [21, 24] compared to offline algorithms. [22] and [93] have the best performance on average in terms of distortion and smoothness.

To investigate the impact of the four regularization parameters $\lambda_1, \lambda_2, \lambda_3$, and λ_4, we run the proposed algorithm with different values on the first video in Fig. 5.11.

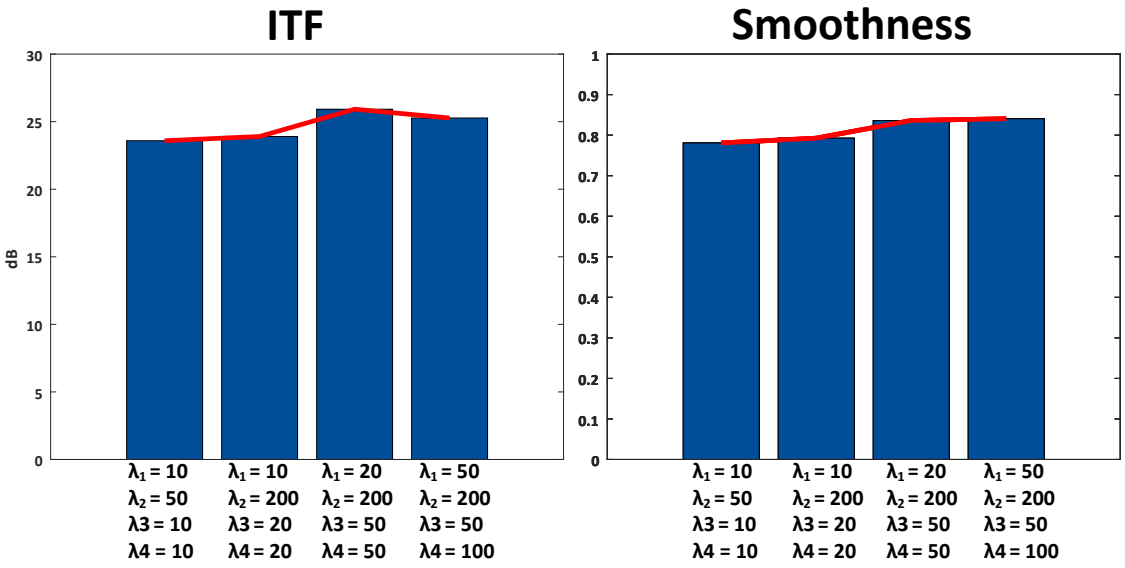


Figure 5.10: Comparison with different combination of $\lambda_1, \lambda_2, \lambda_3, \lambda_4$

As Fig. 5.10 shows, larger regularization parameters will encourage smoothness. However, as they get larger and larger, the algorithm will try to compensate for inter-frame global motions as much as possible, resulting in a static video instead of a stable video. Moreover, overcompensating inter-frame global motions will also

introduce an undefined area in the result frames, which makes the distortion more significant.

Table 5.2: Comparison with real-time video stabilization algorithms. The average time per frame is reported in table. The best results in ITF and Smoothness are bolded. Processing times below 33 ms are also highlighted.

Methods	320 × 240			640 × 360			1280 × 720			1920 × 1080		
	ITF(dB)	Smooth	Time	ITF(dB)	Smooth	Time	ITF(dB)	Smooth	Time	ITF(dB)	Smooth	Time
Meshflow[93]	**31.10**	**0.88**	483.9 ms	26.48	**0.75**	612.5 ms	23.33	0.75	3.421 s	22.57	**0.22**	7.562 s
LP-Kalman-LP[30]	24.54	0.38	**20.08** ms	22.15	0.33	**21.58** ms	20.35	0.23	53.01 ms	21.44	0.12	94.91 ms
Frame-Orbit[78]	27.87	0.75	**4.000** ms	24.25	0.71	**7.300** ms	22.55	0.75	**16.20** ms	21.97	0.13	**27.70**ms
LSstab(ours)	28.68	0.82	**12.59** ms	**26.69**	0.64	**16.21** ms	**24.41**	**0.79**	**31.67** ms	**22.62**	**0.22**	**32.80** ms

Figure 5.11: Comparison with state-of-the-art video stabilization algorithms. Higher ITF and Smoothness means better stabilization result.

5.6 Limitations

The choice of parameters has a direct impact on the quality of stabilization. If λ_1, λ_2, λ_3, and λ_4 are set too large, the proposed algorithm will produce an over-stabilized video and introduce a large cropping area. On the other hand, if they are too small, the algorithm can not remove the jitters in the input video. We found that $\lambda_1 = 2, \lambda_2 = 200, \lambda_3 = 50, \lambda_4 = 50$ produce good empirical results. The other limitation is the accumulated error in the proposed algorithm. The stabilized frame

is obtained through the accumulating products of $\hat{m}$ and m_i based on (5.32) in the motion compensation stage so that any wrong estimates will affect the subsequent frames.

5.7 Conclusions and Future Work

While most video stabilization algorithms are offline and require future frames for smoothing, we propose an online real-time video stabilization algorithm, LSstab, which stabilizes the incoming video frame in real-time using only past frames. LSstab features a parallel realization of the *a-contrario* RANSAC (AC-RANSAC) algorithm to estimate the inter-frame camera motion parameters and a novel fast recursive least squares algorithm to find the stable camera motion obeying cinematography principles. Techniques such as inpainting can be readily included in our algorithm. Currently, the cost function (5.27) does not include the cropping area yet. In the future, one possible direction is to incorporate a proxy of cropping area that is a function of λ_1 and λ_2 to the cost function and further develop an adaptive approach for setting λ_1 and λ_2. The keyframe selection technique proposed in [24] can smoothly be adapted to mitigate the accumulated error.

5.8 Derivation of (5.10) in matrix form

$$\begin{aligned} E &= P + \lambda_1 Q + \lambda_2 U \\ &= \sum_{i=1}^{n} (m_i - \overline{m}_i)^2 + \lambda_1 \sum_{i=2}^{n} (\overline{m}_i - \overline{m}_{i-1})^2 + \lambda_2 \sum_{i=1}^{n-1} (\overline{m}_i - \widehat{m}_i)^2 \end{aligned}$$

Taking all partial derivatives of E w.r.t $\overline{m}_1, \ldots, \overline{m}_n$ and setting them to zero, we have

$$\begin{aligned} \frac{\partial E}{\partial \overline{m}_1} =& \frac{\partial}{\partial \overline{m}_1} [(m_1 - \overline{m}_1)^2 + \lambda_1 (\overline{m}_2 - \overline{m}_1)^2 + \lambda_2 (\overline{m}_1 - \widehat{m}_1)^2] = 0 \\ & \Rightarrow (1 + \lambda_1 + \lambda_2)\overline{m}_1 - \lambda_1 \overline{m}_2 = m_1 + \lambda_2 \widehat{m}_1 \end{aligned}$$

For $i = 2, \ldots, n-1$

$$\begin{aligned}\frac{\partial E}{\partial \overline{m}_i} =& \frac{\partial}{\partial \overline{m}_i}[(m_i - \overline{m}_i)^2 + \lambda_1(\overline{m}_i - \overline{m}_{i-1})^2 + \lambda_1(\overline{m}_{i+1} - \overline{m}_i)^2 + \\ & \lambda_2(\overline{m}_i - \widehat{m}_i)^2] = 0 \\ & \Rightarrow -\lambda_1\overline{m}_{i-1} + (1 + 2\lambda_1 + \lambda_2)\overline{m}_i - \lambda_1\overline{m}_{i+1} = m_i + \lambda_2\widehat{m}_i\end{aligned}$$

Finally,

$$\begin{aligned}\frac{\partial E}{\partial \overline{m}_n} &= \frac{\partial}{\partial \overline{m}_n}[(m_n - \overline{m}_n)^2 + \lambda_1(\overline{m}_n - \overline{m}_{n-1})^2] = 0 \\ &\Rightarrow -\lambda_1\overline{m}_{n-1} + (1 + \lambda_1)\overline{m}_n = m_n\end{aligned}$$

Therefore, solving (5.10) is equivalent to solve the system (5.11) of linear equations.

5.9 Recursive solution for finding $\overline{m}_n$ in (5.11)

Let $a = (1 + \lambda_1 + \lambda_2), b = -\lambda_1, c = (1 + 2\lambda_1 + \lambda_2), d = (1 + \lambda_1)$

For $n = 2$, We have $\mathbf{S}_2 = \begin{bmatrix} a & b \\ b & d \end{bmatrix}$, $\mathbf{S}_2^{-1} = \frac{1}{ad-b^2}\begin{bmatrix} d & -b \\ -b & a \end{bmatrix}$, so

$$\widehat{m}_2 = \frac{-b(1 + \lambda_2)m_1 + am_2}{ad - b^2} \tag{5.38}$$

For $n = 3$,

We perform the Gauss−Jordan elimination on $\mathbf{S}_3$:

$$\left[\begin{array}{ccc|ccc} a & b & 0 & 1 & 0 & 0 \\ b & c & b & 0 & 1 & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(1)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ b & c & b & 0 & 1 & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(2)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & \frac{ac-b^2}{a} & b & -\frac{b}{a} & 1 & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(3)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & b & \frac{ab^2}{ac-b^2} & -\frac{b^2}{ac-b^2} & \frac{ab}{ac-b^2} & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(4)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & b & \frac{ab^2}{ac-b^2} & -\frac{b^2}{ac-b^2} & \frac{ab}{ac-b^2} & 0 \\ 0 & 0 & d-\frac{ab^2}{ac-b^2} & \frac{b^2}{ac-b^2} & -\frac{ab}{ac-b^2} & 1 \end{array}\right]$$

$$\xrightarrow{(5)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & b & \frac{ab^2}{ac-b^2} & -\frac{b^2}{ac-b^2} & \frac{ab}{ac-b^2} & 0 \\ 0 & 0 & 1 & \frac{b^2}{q_3} & -\frac{ab}{q_3} & \frac{ac-b^2}{q_3} \end{array}\right]$$

Then, $\widehat{m}_3 = \begin{bmatrix} \frac{b^2}{q_3} & -\frac{ab}{q_3} & \frac{ac-b^2}{q_3} \end{bmatrix} \cdot \mathbf{Y}_3$ where $q_3 = acd - b^2d - ab^2d$

For $n = 4$,

Note that first 3 row operations on $\mathbf{S}_3$ will yield the same result for $\mathbf{S}_4$. Hence, we can reuse the partial result of Gauss−Jordan elimination of $\mathbf{S}_3$ for $\mathbf{S}_4$. We first record the 2^{nd} row of the above echelon form of $\mathbf{S}_3$, $x_1^{(3)} = \frac{ab^2}{ac-b^2}, y_1^{(3)} = -\frac{b^2}{ac-b^2}, y_2^{(3)} =$

$\frac{ab}{ac-b^2}$, then we have

$$\left[\begin{array}{cccc|cccc} a & b & 0 & 0 & 1 & 0 & 0 & 0 \\ b & c & b & 0 & 0 & 1 & 0 & 0 \\ 0 & b & c & b & 0 & 0 & 1 & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(3)} \left[\begin{array}{cccc|cccc} b & \frac{b^2}{a} & 0 & 0 & \frac{b}{a} & 0 & 0 & 0 \\ 0 & b & x_1^{(3)} & 0 & y_1^{(3)} & y_2^{(3)} & 0 & 0 \\ 0 & b & c & b & 0 & 0 & 1 & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(4)} \left[\begin{array}{cccc|cccc} b & \frac{b^2}{a} & 0 & 0 & \frac{b}{a} & 0 & 0 & 0 \\ 0 & b & x_1^{(3)} & 0 & y_1^{(3)} & y_2^{(3)} & 0 & 0 \\ 0 & 0 & b & \frac{b^2}{c-x_1^{(3)}} & \frac{-y_1^{(3)}b}{c-x_1^{(3)}} & \frac{-y_2^{(3)}b}{c-x_1^{(3)}} & \frac{b}{c-x_1^{(3)}} & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(5)} \left[\begin{array}{cccc|cccc} b & \frac{b^2}{a} & 0 & 0 & \frac{b}{a} & 0 & 0 & 0 \\ 0 & b & x_1^{(3)} & 0 & y_1^{(3)} & y_2^{(3)} & 0 & 0 \\ 0 & 0 & b & \frac{b^2}{c-x_1^{(3)}} & \frac{-y_1^{(3)}b}{c-x_1^{(3)}} & \frac{-y_2^{(3)}b}{c-x_1^{(3)}} & \frac{b}{c-x_1^{(3)}} & 0 \\ 0 & 0 & 0 & 1 & \frac{y_1 b}{q_4} & \frac{y_2 b}{q_4} & \frac{-b}{q_4} & \frac{c-x_1}{q_4} \end{array}\right]$$

Then, $\widehat{m}_4 = \left[\begin{array}{cccc} \frac{y_1 b}{q_4} & \frac{y_2 b}{q_4} & \frac{-b}{q_4} & \frac{c-x_1}{q_4} \end{array}\right] \cdot \mathbf{Y}_4$ where $q_4 = cd - x_1^{(3)}d - b^2$. Note that we directly use the result from the previous iteration in (3), and the row operations(4) (5) are the same for each iteration.

For $n = 5$, we would need to store 3^{rd} row of the above echelon form of $\mathbf{S}_4$, i.e. $x_1^{(4)} = \frac{b^2}{c-x_1^{(3)}}, y_1^{(4)} = \frac{-y_1^{(3)}b}{c-x_1^{(3)}}, y_2^{(4)} = \frac{-y_2^{(3)}b}{c-x_1^{(3)}}, y_3^{(4)} = \frac{b}{c-x_1^{(3)}}$.

Suppose we have compute the echelon form for $\mathbf{S}_{n-1}$ and store the $(n-1)^{th}$ row as $x_1^{(n-1)}, y_1^{(n-1)}, \ldots, y_{n-3}^{(n-1)}$, Then we have the Gauss−Jordan elimination for $\mathbf{S}_n$ as:

$$\left[\begin{array}{cccc|cccc} a & b & \dots & 0 & 1 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & b & c & b & 0 & 0 & 1 & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{cccc|cccccc} \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & y_2^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \\ \cdots & b & c & b & 0 & 0 & 0 & 0 & 1 & 0 \\ \cdots & 0 & b & d & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{ccc|ccccc} \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \\ \cdots & 0 & b & \frac{b^2}{c-x_1^{(n-1)}} & \frac{-y_1^{(n-3)}b}{c-x_1^{(n-1)}} & \cdots & \frac{-y_{n-3}^{(n-1)}b}{c-x_1^{(n-1)}} & \frac{b}{c-x_1^{(n-1)}} & 0 \\ \cdots & 0 & b & d & 0 & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{ccc|ccccc} \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \\ \cdots & 0 & b & \frac{b^2}{c-x_1^{(n-1)}} & \frac{-y_1^{(n-3)}b}{c-x_1^{(n-1)}} & \cdots & \frac{-y_{n-3}^{(n-1)}b}{c-x_1^{(n-1)}} & \frac{b}{c-x_1^{(n-1)}} & 0 \\ \cdots & 0 & 0 & 1 & \frac{y_1^{(n-1)}b}{q_n} & \cdots & \frac{y_{n-3}^{(n-1)}b}{q_n} & \frac{-b}{q_n} & \frac{c-x_1^{(n-1)}}{q_n} \end{array}\right]$$

Therefore,

$$\widehat{m}_n = \left[\frac{y_2^{(n-1)}b}{q_n} \dots \frac{y_{n-3}^{(n-1)}b}{q_n} \frac{-b}{q_n} \frac{c-x_1^{(n-1)}}{q_n}\right] \cdot \mathbf{Y}_n \tag{5.39}$$

where $q_n = cd - x_1^{(n-1)}d - b^2$. Then we store the $(n-1)^{th}$ row of the above echelon form of $\mathbf{S}_n$ for the next iteration.

CHAPTER 6
Conclusion

The thesis is focused on the development of algorithms for 3D visualization in laparoscopic surgery, particularly utilizing real-time video feeds from multiple camera arrays. Here's a summary of the main contributions:

- **Quasi Real-time Image-based 3D Reconstruction and Rendering**:
  - The proposed algorithm reconstructs a 3D model in quasi real-time using multiview images.
  - No special hardware is required for the reconstruction process.
  - Depth estimation is focused specifically on sparse feature points rather than estimating depth for each pixel.
  - Two novel filters, based on epipolar geometry and trifocal tensor, are introduced to robustly estimate depths for these sparse feature points.
- **Efficient and Robust Surgical Grasper Reconstruction**:
  - A novel method is presented for reconstructing the surgical grasper in an efficient and robust manner.
  - The framework seamlessly integrates the Kalman filter and Direct Linear Transform (DLT).
  - DLT is reformulated as a weighted least squares problem, with weights determined by tracking uncertainties from the Kalman filter.
  - This design stabilizes tracking results using the Kalman filter and improves the accuracy of reconstruction by accounting for tracking errors within the standard DLT.

- **Online Real-time Video Stabilization Algorithm**:
  - A novel algorithm, LSstab, is introduced for real-time video stabilization, focusing on suppressing unwanted motion jitters.
  - The algorithm incorporates cinematography principles in its approach.
  - It features a parallel realization of AC-RANSAC for estimating inter-frame camera motion parameters.
  - A new least squares-based smoothing cost function is proposed to mitigate undesirable camera jitters based on cinematography principles. A recursive least square solver is derived to minimize the smoothing cost function with linear computation complexity.
  - Results demonstrate that the proposed algorithm achieves comparable or superior performance, especially in terms of real-time processing speed when utilizing a GPU.

Overall, this work addresses key challenges in 3D visualization for laparoscopic surgery, providing innovative solutions for real-time 3D visualization, surgical grasper reconstruction, and real-time video stabilization, with a focus on practical applicability in a practical setting.

Bibliography

[1] S. Galliani, K. Lasinger, and K. Schindler, "Massively parallel multiview stereopsis by surface normal diffusion," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 873–881, 2015.

[2] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanæs, "Large scale multi-view stereopsis evaluation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 406–413, 2014.

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

[4] S. Shen, "Accurate multiple view 3d reconstruction using patch-based stereo for large-scale scenes," *IEEE Transactions on Image Processing*, vol. 22, no. 5, pp. 1901–1914, 2013.

[5] E. Zheng, E. Dunn, V. Jojic, and J. Frahm, "Patchmatch based joint view selection and depthmap estimation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1510–1517, 2014.

[6] Q. Xu and W. Tao, "Multi-scale geometric consistency guided multi-view stereo," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5478–5487, 2019.

[7] A. J. Watras, J.-J. Kim, H. Liu, Y. H. Hu, and H. Jiang, "Optimal camera pose and placement configuration for maximum field-of-view video stitching," *Sensors*, vol. 18, no. 7, 2018.

[8] J.-J. Kim, A. Watras, H. Liu, Z. Zeng, J. A. Greenberg, C. P. Heise, Y. H. Hu, and H. Jiang, "Large-field-of-view visualization utilizing multiple miniaturized cameras for laparoscopic surgery," *Micromachines*, vol. 9, no. 9, 2018.

[9] A. Watras, J. Ke, Z. Zeng, J.-J. Kim, H. Liu, H. Jiang, and Y. H. Hu, "Parallax mitigation for real-time close field video stitching," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 568–571, 2017.

[10] P. Moulon, P. Monasse, and R. Marlet, "Adaptive structure from motion with a contrario model estimation," in *Computer Vision – ACCV 2012* (K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, eds.), (Berlin, Heidelberg), pp. 257–270, Springer Berlin Heidelberg, 2013.

[11] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. USA: Cambridge University Press, 2 ed., 2003.

[12] P. L. M. Bouttefroy, A. Bouzerdoum, S. L. Phung, and A. Beghdadi, "Integrating the projective transform with particle filtering for visual tracking," *EURASIP Journal on Image and Video Processing*, vol. 2011, p. 839412, Nov 2010.

[13] J. Sánchez, "Comparison of Motion Smoothing Strategies for Video Stabilization using Parametric Models," *Image Processing On Line*, vol. 7, pp. 309–346, 2017. `https://doi.org/10.5201/ipol.2017.209`.

[14] J. Sánchez, "Comparison of motion smoothing strategies for video stabilization using parametric models," *Image Processing On Line*, vol. 7, pp. 309–346, 11 2017.

[15] J. Chen, J. Ke, F. Lu, J. Fernandes, B. King, Y. H. Hu, and H. Jiang, "Gait monitoring using an ankle-worn stereo camera system," in *2022 IEEE Sensors*, pp. 1–4, 2022.

[16] J. Ke, A. J. Watras, J.-J. Kim, H. Liu, H. Jiang, and Y. H. Hu, "Towards real-time 3d visualization with multiview rgb camera array," *Journal of Signal Processing Systems*, vol. 94, pp. 329–343, Mar 2022.

[17] A. J. Watras, J.-J. Kim, J. Ke, H. Liu, J. A. Greenberg, C. P. Heise, Y. H. Hu, and H. Jiang, "Large-field-of-view visualization with small blind spots utilizing tilted micro-camera array for laparoscopic surgery," *Micromachines*, vol. 11, no. 5, 2020.

[18] J. Ke, A. J. Watras, J.-J. Kim, H. Liu, H. Jiang, and Y. H. Hu, "Towards real-time, multi-view video stereopsis," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1638–1642, 2020.

[19] W. G. Aguilar and C. Angulo, "Real-Time Model-Based Video Stabilization for Microaerial Vehicles," *Neural Processing Letters*, vol. 43, no. 2, pp. 459–477, 2016.

[20] Y. Wang, Z. J. Hou, K. Leman, and R. Chang, "Real-time video stabilization for Unmanned Aerial Vehicles," *Proceedings of the 12th IAPR Conference on Machine Vision Applications, MVA 2011*, pp. 336–339, 2011.

[21] Y. Matsushita, W. Ge, X. Tang, and H. Y. Shum, "Full-frame video stabilization with motion inpainting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1150–1163, 2006.

[22] F. Liu, M. Gleicher, H. Jin, and A. Agarwala, "Content-preserving warps for 3D video stabilization," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[23] W. Guilluy, L. Oudre, and A. Beghdadi, "Video stabilization: Overview, challenges and perspectives," *Signal Processing: Image Communication*, vol. 90, p. 116015, 2021.

[24] M. Grundmann, V. Kwatra, and I. Essa, "Auto-directed video stabilization with robust L1 optimal camera paths," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, no. 1, pp. 225–232, 2011.

[25] H. C. Chang, S. H. Lai, and K. R. Lu, "A robust real-time video stabilization algorithm," *Journal of Visual Communication and Image Representation*, vol. 17, no. 3, pp. 659–673, 2006.

[26] S. Battiato, G. Gallo, G. Puglisi, and S. Scellato, "Sift features tracking for video stabilization," in *14th International Conference on Image Analysis and Processing (ICIAP 2007)*, pp. 825–830, Sep. 2007.

[27] S. Liu, L. Yuan, P. Tan, and J. Sun, "Bundled camera paths for video stabilization," in *ACM Transactions on Graphics (SIGGRAPH 2013)*, vol. 32, ACM, July 2013.

[28] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala, "Subspace video stabilization," *ACM Trans. Graph.*, vol. 30, Feb. 2011.

[29] 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, p. 381–395, June 1981.

[30] L. Kejriwal and I. Singh, "A hybrid filtering approach of digital video stabilization for uav using kalman and low pass filter," *Procedia Computer Science*, vol. 93, pp. 359 – 366, 2016. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[31] A. Litvin, J. Konrad, and W. C. Karl, "Probabilistic video stabilization using Kalman filtering and mosaicing," in *Image and Video Communications and Processing 2003*, vol. 5022, pp. 663 – 674, International Society for Optics and Photonics, SPIE, 2003.

[32] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006.

[33] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *In VISAPP International Conference on Computer Vision Theory and Applications*, pp. 331–340, 2009.

[34] S. A. K. Tareen and Z. Saleem, "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk," in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pp. 1–10, 2018.

[35] F. K. Noble, "Comparison of opencv's feature detectors and feature matchers," in *2016 23rd International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pp. 1–6, 2016.

[36] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004.

[37] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 430–443, Springer Berlin Heidelberg, 2006.

[38] C. R. Michael Bleyer and C. Rother, "Patchmatch stereo - stereo matching with slanted support windows," in *Proceedings of the British Machine Vision Conference*, pp. 14.1–14.11, BMVA Press, 2011. http://dx.doi.org/10.5244/C.25.14.

[39] F. Besse, C. Rother, A. Fitzgibbon, and J. Kautz, "Pmbp: Patchmatch belief propagation for correspondence field estimation," *International Journal of Computer Vision*, vol. 110, pp. 2–13, Oct 2014.

[40] P. Labatut, J. Pons, and R. Keriven, "Robust and efficient surface reconstruction from range data," *Computer Graphics Forum*, vol. 28, pp. 2275 – 2290, 10 2009.

[41] M. Jancosek and T. Pajdla, "Multi-view reconstruction preserving weakly-supported surfaces," in *CVPR 2011*, pp. 3121–3128, 2011.

[42] M. Waechter, N. Moehrle, and M. Goesele, "Let there be color! large-scale texturing of 3d reconstructions," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 836–850, Springer International Publishing, 2014.

[43] Q.-Y. Zhou and V. Koltun, "Color map optimization for 3d reconstruction with consumer depth cameras," *ACM Trans. Graph.*, vol. 33, July 2014.

[44] Y. Fu, Q. Yan, L. Yang, J. Liao, and C. Xiao, "Texture mapping for 3d reconstruction with rgb-d sensor," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4645–4653, 2018.

[45] C.-C. Lee, A. Tabatabai, and K. Tashiro, "Free viewpoint video (fvv) survey and future research direction," *APSIPA Transactions on Signal and Information Processing*, vol. 4, p. e15, 2015.

[46] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan, "High-quality streamable free-viewpoint video," *ACM Trans. Graph.*, vol. 34, July 2015.

[47] C. Lipski, F. Klose, and M. Magnor, "Correspondence and depth-image based rendering a hybrid approach for free-viewpoint video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 6, pp. 942–951, 2014.

[48] A. Mustafa, H. Kim, J.-Y. Guillemaut, and A. Hilton, "Temporally coherent 4d reconstruction of complex dynamic scenes," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4660–4669, 2016.

[49] A. Mustafa, H. Kim, J.-Y. Guillemaut, and A. Hilton, "General dynamic scene reconstruction from multiple view video," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[50] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt, and M. Stamminger, "Real-time non-rigid reconstruction using an rgb-d camera," *ACM Trans. Graph.*, vol. 33, July 2014.

[51] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 343–352, 2015.

[52] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger, "VolumeDeform: Real-time Volumetric Non-rigid Reconstruction," October 2016.

[53] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escolano, C. Rhemann, D. Kim, J. Taylor, P. Kohli, V. Tankovich, and S. Izadi, "Fusion4d: Real-time performance capture of challenging scenes," *ACM Trans. Graph.*, vol. 35, July 2016.

[54] M. Slavcheva, M. Baust, D. Cremers, and S. Ilic, "Killingfusion: Non-rigid 3d reconstruction without correspondences," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5474–5483, 2017.

[55] M. Slavcheva, M. Baust, and S. Ilic, "Sobolevfusion: 3d reconstruction of scenes undergoing free non-rigid motion," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2646–2655, 2018.

[56] G. Chaurasia, A. Nieuwoudt, A.-E. Ichim, R. Szeliski, and A. Sorkine-Hornung, "Passthrough+: Real-time stereoscopic view synthesis for mobile mixed reality," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, may 2020.

[57] G. Chaurasia, A. Nieuwoudt, A.-E. Ichim, R. Szeliski, and A. Sorkine-Hornung, "Passthrough+: Real-time stereoscopic view synthesis for mobile mixed reality," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, Apr. 2020.

[58] E. Dupont, B. Miguel Angel, A. Colburn, A. Sankar, C. Guestrin, J. Susskind, and Q. Shan, "Equivariant neural rendering," *arXiv preprint arXiv:2006.07630*, 2020.

[59] K. Olszewski, S. Tulyakov, O. Woodford, H. Li, and L. Luo, "Transformable bottleneck networks," *The IEEE International Conference on Computer Vision (ICCV)*, Nov 2019.

[60] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), (Cham), pp. 405–421, Springer International Publishing, 2020.

[61] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron, "Nerv: Neural reflectance and visibility fields for relighting and view synthesis," in *CVPR*, 2021.

[62] Q. Wang, Z. Wang, K. Genova, P. Srinivasan, H. Zhou, J. T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser, "Ibrnet: Learning multi-view image-based rendering," in *CVPR*, 2021.

[63] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, "pixelNeRF: Neural radiance fields from one or few images," in *CVPR*, 2021.

[64] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-nerf: Neural radiance fields for dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10318–10327, 2021.

[65] T. Li, M. Slavcheva, M. Zollhoefer, S. Green, C. Lassner, C. Kim, T. Schmidt, S. Lovegrove, M. Goesele, R. Newcombe, *et al.*, "Neural 3d video synthesis from multi-view video," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5521–5531, 2022.

[66] A. Goldstein and R. Fattal, "Video stabilization using epipolar geometry," *ACM Trans. Graph.*, vol. 32, no. 5, 2012.

[67] S. Bell, A. Troccoli, and K. Pulli, "A non-linear filter for gyroscope-based video stabilization," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele,

and T. Tuytelaars, eds.), (Cham), pp. 294–308, Springer International Publishing, 2014.

[68] H. Ovrén and P.-E. Forssén, "Gyroscope-based video stabilisation with auto-calibration," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2090–2097, 2015.

[69] S. Liu, Y. Wang, L. Yuan, J. Bu, P. Tan, and J. Sun, "Video stabilization with a depth camera," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 89–95, 2012.

[70] B. M. Smith, L. Zhang, H. Jin, and A. Agarwala, "Light field video stabilization," in *2009 IEEE 12th International Conference on Computer Vision*, pp. 341–348, 2009.

[71] M. Wang, G.-Y. Yang, J.-K. Lin, S.-H. Zhang, A. Shamir, S.-P. Lu, and S.-M. Hu, "Deep online video stabilization with multi-grid warping transformation learning," *IEEE Transactions on Image Processing*, vol. 28, no. 5, pp. 2283–2292, 2019.

[72] S.-Z. Xu, J. Hu, M. Wang, T.-J. Mu, and S.-M. Hu, "Deep video stabilization using adversarial networks," *Computer Graphics Forum*, vol. 37, no. 7, pp. 267–276, 2018.

[73] M. Zhao and Q. Ling, "Pwstablenet: Learning pixel-wise warping maps for video stabilization," *IEEE Transactions on Image Processing*, vol. 29, pp. 3582–3595, 2020.

[74] J. Yu and R. Ramamoorthi, "Learning video stabilization using optical flow," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[75] J. Choi and I. S. Kweon, "Deep iterative frame interpolation for full-frame video stabilization," *ACM Trans. Graph.*, vol. 39, jan 2020.

[76] Z. Shi, F. Shi, W.-S. Lai, C.-K. Liang, and Y. Liang, "Deep online fused video stabilization," in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 865–873, 2022.

[77] S. Ertürk, "Real-time digital image stabilization using Kalman filters," *Real-Time Imaging*, vol. 8, no. 4, pp. 317–328, 2002.

[78] J. Dong and H. Liu, "Video stabilization for strict real-time applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, pp. 716–724, April 2017.

[79] K. Ratakonda, "Real-time digital video stabilization for multi-media applications," *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 69–72, 1998.

[80] U. Technologies, "Unity," 2019.

[81] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3d," *ACM Trans. Graph.*, vol. 25, p. 835–846, July 2006.

[82] J. L. Schönberger and J. Frahm, "Structure-from-motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4104–4113, 2016.

[83] C. Wu, "Towards linear-time incremental structure from motion," in *2013 International Conference on 3D Vision - 3DV 2013*, pp. 127–134, 2013.

[84] J. Zhang, M. Boutin, and D. G. Aliaga, "Robust bundle adjustment for structure from motion," in *2006 International Conference on Image Processing*, pp. 2185–2188, 2006.

[85] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), p. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[86] L. Moisan and B. Stival, "A probabilistic criterion to detect rigid point matches between two images and estimate the fundamental matrix," *International Journal of Computer Vision*, vol. 57, no. 3, pp. 201–218, 2004.

[87] A. Desolneux, L. Moisan, and J.-M. Morel, "Meaningful alignments," *International Journal of Computer Vision*, vol. 40, no. 1, pp. 7–23, 2000.

[88] L. Moisan, P. Moulon, and P. Monasse, "Automatic Homographic Registration of a Pair of Images, with A Contrario Elimination of Outliers," *Image Processing On Line*, vol. 2, pp. 56–73, 2012.

[89] "Programming guide :: Cuda toolkit documentation," 2020.

[90] M. Harris, "Optimizing parallel reduction in cuda."

[91] Q. Zhang, "Some implementation aspects of sliding window least squares algorithms," *IFAC Proceedings Volumes*, vol. 33, no. 15, pp. 763 – 768, 2000.

[92] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007.

[93] S. Liu, P. Tan, L. Yuan, J. Sun, and B. Zeng, "Meshflow: Minimum latency online video stabilization," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 800–815, Springer International Publishing, 2016.

[94] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala, "Subspace video stabilization," *ACM Trans. Graph.*, vol. 30, Feb. 2011.