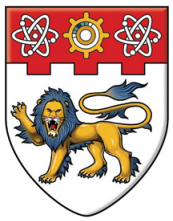

NANYANG
TECHNOLOGICAL
UNIVERSITY

Embedded Computing Techniques for Remote Mobile Video Surveillance Systems

Nirmala Ramakrishnan

School of Computer Science and Engineering

A thesis submitted to Nanyang Technological University

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

2017

Acknowledgements

My journey this far in my research has been made possible by the wonderful people around me. Here is my humble attempt to express my gratitude for all their time, efforts and motivation.

First, I would like to express my deepest gratitude to my supervisor Prof. Thambipillai Srikanthan for being a constant source of energy and inspiration throughout this research work, with his relentless guidance and motivation. Despite his numerous commitments, Prof. Sri ensured that he engaged with me in weekly discussions that kept up my momentum. I also appreciate him for always pushing me beyond my comfort zones that allowed me to grow as a researcher.

I would like to thank Dr. Siew-Kei Lam for all the discussions to strengthen my work and his patient guidance throughout all my research publications. I am grateful to Dr. Meiqing Wu for all the brainstorming sessions and for inspiring me with the rigour she brings into her research. I would like to thank Dr. Alok Prakash for helping me during the critical phase of thesis writing. Thank you Supriya Sathyanarayana, Kratika Garg, Dr. Suchitra Sathyanarayana and Dr. Ravi Satzoda for the numerous brainstorming sessions and for making computer vision research fun. For all the technical and logistic support, I wish to thank Mr. Jeremiah Chua and Ms. Nah Kiat Joo.

I am forever grateful to be blessed with the most supportive family and loving friends without whom this journey would have been impossible. Thank you Bhaskar, my pillar of strength. Thank you Amma, Appa and Sowmya, and Paatti for being there always. My friends Kavitha, Karthik, Kavitha and Deepthi, thank you.

Dedicated to Shreyas, my family, friends and teachers who have always inspired me to pursue excellence relentlessly

Abstract

Unmanned aerial vehicles (UAVs) equipped with cameras are increasingly being deployed for performing vision-based wide area surveillance with minimal human intervention. Existing techniques employing global motion estimation (GME) for automatic surveillance are typically complex and compute-intensive. In this thesis, low-complexity techniques for the key functional blocks of the GME, namely, corner detection, feature tracking and robust estimation have been proposed. The proposed techniques are capable of adapting to varying image content, camera motions and moving targets, thereby making them suitable for real-time processing of aerial videos on resource-limited UAVs.

A novel compute-efficient pruning technique (called PP-ER) for corner detection is proposed using simple approximations of the Shi-Tomasi and Harris corner measures for rapid and efficient extraction of high-quality corner candidates. This allows for restricting the complex corner measure computations to only a small pool of corner candidates. Evaluations on a Nios-II platform, without floating point unit (FPU) show a speedup in execution time of 48-82% in Shi-Tomasi and 45-81% in Harris corner detection when detecting 300 corners, at the same time achieving comparable accuracy as the conventional Shi-Tomasi/Harris detectors, when applied to the Oxford repeatability dataset.

In order to eliminate the need for manual setting of optimal threshold, to guarantee the required number of corners for a wide range of image content, an automated thresholding method based on iterative thresholding is proposed, in this thesis. This has led to notable reduction in the number of trials needed to release the required corner candidates. In order to further enhance compute efficiency, a mask-based non-maximal suppression scheme is employed to turn off neighbours of already selected corners. Evaluations show that with automated thresholding, an average of only 1.8% and 4.3% of the corner candidates are released, when compared with fixed threshold based detection of 1000 corners on Shi-Tomasi and Harris detectors respectively.

The low complexity pruning and automated thresholding were integrated into the corner detection process, to reduce the computational complexity and to eliminate the need for manual intervention for threshold selection, respectively. Evaluations on the Nios-II platform with a floating point unit, show an average speed-up in execution time of 67% for Shi-Tomasi and 51% in Harris corner detectors without compromising on the quality of the corners reported.

Next, a low-complexity GME method (called sparse-GME) is proposed by employing a minimum number of well-distributed sparse corner features. A selective and systematic re-population strategy has also been introduced to improve the accuracy prior to necessitating a uniform increase in the overall density of features. Methods for rapid evaluation of GM estimations were introduced to facilitate this iterative process. Evaluations on aerial video datasets show that for 95% of the frames, GME with the first pass sparse estimation is performed, while achieving a similar accuracy as the dense set of features for 97% of the cases. Results with simulated data show that the proposed method is able to deterministically ramp up the features when the number of moving objects is increased.

To cope with significant distortions, a novel adaptive windowing method was introduced within the Kanade-Lucas-Tomasi (KLT) feature tracker. This has eliminated unnecessary computations and enhanced the robustness of GME during fast rotations and scale changes in the camera motion. Evaluations with a benchmark tracking dataset show that the proposed adaptive windowing method outperforms the conventional fixed-window KLT in terms of robustness. In addition, compared to the well-known affine KLT, the proposed method achieves comparable robustness at an average runtime speedup of 7x. On simulated frames with global motions of in-plane rotations and scale changes, applying the robust adaptive windowing for KLT leads to 70% reduction in the GME error for the proposed sparse-GME.

Finally, a unified computation framework is proposed to show how the proposed techniques for the individual modules of GME, can be fully integrated to realize an adaptive and low-complexity GME for deployment on low-resource platforms in aerial video surveillance systems.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
CCP	Cascaded Candidate Pruning
CCTV	Closed Circuit TeleVision
CIT	Coarse Iterative Thresholding
CPU	Central Processing Unit
EIT	Exhaustive Iterative Thresholding
ER	Edge Removal
FAST	Features from Accelerated Segment Test
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
GME	Global Motion Estimation
GPS	Global Positioning System
GPU	Graphics Processing Unit
HB	Harris with Box filter
HD	High Definition
HG	Harris with Gaussian filter
IMU	Inertial Measurement Unit
IT	Iterative Thresholding
KLT	Kanade-Lucas-Tomasi feature tracker
MBC	Mosaic-Based Compression
MPEG	Moving Picture Experts Group
MSE	Mean-Square Error
NMS	Non-Maximal Suppression
PP	Partial Pruning
PROSAC	Progressive Sample Consensus
PSNR	Peak Signal-to-Noise Ratio
RanSaC	Random Sample Consensus

ROI	Region-Of-Interest
SIFT	Scale-Invariant Feature Transform
SIMD	Single Instruction, Multiple Data
STB	Shi-Tomasi with Box filter
STG	Shi-Tomasi with Gaussian filter
SURF	Speeded-Up Robust Features
SUSAN	Smallest Univalue Segment Assimilating Nucleus
UAV	Unmanned Aerial Vehicle

1

Introduction

Motivation

Wide area surveillance and monitoring is critical for disaster recovery and response, conservation efforts and large infrastructure management. Commercial unmanned aerial vehicles (UAVs) are being increasingly deployed to collect high resolution data of the area under surveillance, as they are affordable, provide a unique aerial perspective and can be sent to remote and inaccessible areas, requiring minimal human intervention. On-board cameras are a primary form of data collection, resulting in huge volumes of high resolution still/video imagery. Information such as the presence of moving objects needs to be extracted from these videos, as they represent "interesting" events in surveillance. It is critical that these events are relayed back to the ground station, for speedy and meaningful response. However,

the battery-powered UAVs operate on very stringent on-board power budgets that determine the duration of their flight. Therefore on-board power consumption for communication needs to be kept minimal. In addition, the communication bandwidth itself may not allow high resolution videos to be transmitted. Therefore, on-board processing of videos is essential to extract relevant information and communicate only this information, in real-time. Several computer vision techniques for motion detection have been proposed that separate moving objects from the scene. This has also enabled efficient compression techniques such as mosaic-based compression (MBC) of aerial videos, by compressing the moving objects separately from the background scene. However, these algorithms are highly complex in terms of computations, and on-board processing has been demonstrated only for short duration flights. A major contributor to the complexity is the global motion estimation (GME) step, which removes the artificial motion induced in the video frames due to moving camera. This is the first step in most aerial video processing algorithm chains. As GME needs to handle wide range of image content and diverse scene complexity in terms of camera motion and moving targets, the computational complexity of existing methods is prohibitively high. Compute-efficient techniques for GME need to be developed that can be deployed on resource-limited surveillance UAVs. These challenges motivate the proposed research in this thesis.

Scope and Objectives

The aim of this research is to develop embedded computing techniques for remote mobile video surveillance systems such as the surveillance UAVs, specifically for global motion estimation (GME). The literature will be surveyed for existing algorithms for on-board vision processing for UAVs, specifically for performing GME, as this is the primary step for processing videos from moving cameras. The existing techniques will be analysed to identify causes of high complexity. The application specific constraints for aerial video processing shall be mapped to the algorithms to explore opportunities to reduce computations. Architecture-aware

algorithm innovations will be proposed for the key functional blocks in global motion estimation, namely corner detection, feature tracking and robust estimation, leading to low-complexity GME suitable for low-resource platforms on surveillance UAVs. Extensive evaluations shall be undertaken to ensure that low-complexity is achieved without compromising the accuracy of the algorithms.

Research Contributions

The main research contributions of this thesis are summarised below:

1. A low-complexity pruning technique is proposed, in order to accelerate corner detection with the widely used Shi-Tomasi and Harris algorithms. Novel computationally simple approximations of the complex corner measure are employed to prune away non-corner regions efficiently. The proposed pruning technique leads to rapid extraction of corner candidates, at the same time achieving comparable accuracy as the baseline detectors.

2. An automated thresholding method is proposed for Shi-Tomasi and Harris corner detection algorithms that eliminates the need for user-specified thresholds for quality of corners. A novel iterative threshold sampling scheme supported by a mask-based non-maximal suppression step is employed to release minimum number of corner candidates for corner detection. The proposed method guarantees the extraction of the required number of corners on a wide range of image content, as is common on aerial surveillance videos.

3. The proposed automated thresholding and pruning techniques are combined, leading to adaptive yet low-complexity corner detection with Shi-Tomasi and Harris algorithms, by processing minimum number of corner candidates. A novel bin-based approach is proposed to collect and process corner candidates in a controlled manner, achieving comparable accuracy as the baseline detectors while substantially reducing the complexity of corner detection.

4. A novel strategy for low-complexity global motion estimation is proposed that employs minimum number of corner features for estimation. Sparse feature sets that are spatially well-distributed are employed for the estimation. A fast evaluation strategy is proposed to detect failure in estimation with the sparse set, which triggers progressive repopulation of features only when estimation fails.

5. A novel adaptive windowing method for the widely used Kanade-Lucas-Tomasi (KLT) feature tracking algorithm is proposed, to reduce the overall complexity and substantially improve the robustness of the tracker to drastic rotations and scale changes caused by fast camera motions. A window size sampling scheme is employed that selects the optimal window size with minimal computation overhead. This leads to fast yet accurate feature correspondences, which in turn substantially improve the accuracy of the GME.

6. A unified framework for low-complexity and adaptive GME is proposed that combines all the individual functional blocks - corner detection, feature tracking and robust estimation. The GME with sparse features is employed as a controller, and its interactions with the corner detection and tracking blocks are detailed that leads to the overall computations being adapted to the scene content and complexity of the camera motion.

Research Publications

Journal

- Nirmala Ramakrishnan, Meiqing Wu, Siew-Kei Lam, and Thambipillai Srikanthan, **"Enhanced low-complexity pruning for corner detection"**, *Journal of Real-Time Image Processing*, Vol. 2, No. 1, pp. 197-213, 2016. [1]

Conference

- Meiqing Wu, Nirmala Ramakrishnan, Siew-Kei Lam, and Thambipillai Srikanthan, **"Low-complexity pruning for accelerating corner detection"**. *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp 1684-1687, 2012. [2]

- Nirmala Ramakrishnan, Meiqing Wu, Siew-Kei Lam, and Thambipillai Srikanthan, **"Automated thresholding for low-complexity corner detection"**, *IEEE NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, pp 97-102, 2014. [3]

- Nirmala Ramakrishnan, Meiqing Wu, Siew-Kei Lam and Thambipillai Srikanthan, **"Mask-based non-maximal suppression with iterative pruning for low complexity corner detection"**, *IEEE Intl. Symp. on Integrated Circuits (ISIC)*, pp 368-371, 2014. [4]

- Nirmala Ramakrishnan, Thambipillai Srikanthan, Siew-Kei Lam, and Gauri Ravindra Tulsulkar, **"Adaptive window strategy for high-speed and robust KLT feature tracker"**, *Image and Video Technology: 7th Pacific Rim Symp. (PSIVT)*, pp 355-367, 2015. [5]

In Preparation

- Nirmala Ramakrishnan, Thambipillai Srikanthan and Gauri Ravindra Tulsulkar, **"Low-complexity global motion estimation for aerial videos with sparse features"**.

Organisation of the Thesis

The thesis is organized as described below:

- **Chapter 2**: A detailed literature survey is presented. The literature on computer vision tasks for UAV surveillance applications is reviewed. The common functional blocks that enable these surveillance tasks are identified. Existing methods for global motion estimation are investigated to understand open challenges in deploying them on low-resource platforms that are typical in surveillance UAVs.

- **Chapter 3**: A low-complexity pruning technique for rapid extraction of corner candidates is proposed, in order to address the high computational complexity of Shi-Tomasi and Harris corner detectors. A cost analysis of the proposed pruning technique in comparison to the conventional corner measures is presented. A thorough accuracy evaluation is conducted with the well-known Oxford repeatability dataset to demonstrate the accuracy of the pruning-based corner detection. The Nios-II embedded platform is used to demonstrate the computational benefits of the proposed method by turning on/off the floating point unit and the on-board cache. The pruning based corner detection is applied for global motion estimation in an aerial surveillance video dataset and the accuracy of GME is evaluated.

- **Chapter 4**: In this chapter, an automated thresholding method is proposed, that eliminates human intervention to set the threshold for quality in corner detection. An iterative threshold sampling scheme is proposed to extract the required number of corners by processing the minimum corner candidates in least number of sampling trials. In addition, a mask-based non-maximal suppression scheme is also proposed to complement the iterative thresholding, in order to further reduce the number of candidates. Evaluations in comparison with existing methods for automating thresholds is presented in terms of the reduction in the number of corner candidates processed for corner detection.

- **Chapter 5**: The low-complexity pruning technique presented in Chapter 3 is combined with the automated thresholding method proposed in Chapter 4, into an adaptive and low-complexity alternative for Shi-Tomasi and Harris corner detectors that requires no manual intervention. A bin-based mechanism for the collection of corner candidates is proposed that enables this combination to work. Criteria for managing the release of candidates in low-quality ranges of threshold is also proposed enabling deterministic convergence of the method. The accuracy is evaluated by measuring the number of corner matches between the proposed method and the baseline Shi-Tomasi and Harris corner detectors. Evaluations on the Nios-II platform are conducted to demonstrate the computational efficiency of the proposed method.

- **Chapter 6**: The number of features used for global motion estimation has a direct impact on the computational complexity. In this chapter, the assumption of a dense feature set for estimation is challenged and the conditions when a sparse feature set can be used are identified. A novel sparse GME method is proposed to substantially reduce the computations needed for GME. The proposed method employs very sparse but well-distributed features. A simple evaluation strategy for determining failure in estimation with the sparse set is proposed. A progressive re-population strategy that prioritizes spatial distribution, is employed to deal with failures in estimation. The proposed method is evaluated on aerial datasets captured by UAVs at various flying altitudes. A simulated dataset is also generated for evaluations, in which the scene conditions in terms of number of moving objects and the camera motion (i.e. rotation and scale changes) are varied. Evaluations are performed to demonstrate how the proposed method adapts the number of features to the complexity of the scene conditions.

- **Chapter 7**: The robustness of the feature tracker determines the quality of the feature correspondences used for estimation. In this chapter, it is shown that the window size used by the KLT feature tracker needs to be optimal for each feature and across pyramid levels to improve the robustness of the

tracker in the face of fast rotations and scale changes. A computationally lean window sampling method for KLT is proposed that relies on monitoring the performance of KLT to detect failure in tracking, and arrives at a near optimal window size. The proposed method is evaluated on a well-known annotated tracking dataset and compared with the widely used affine formulation of KLT. A simulated dataset is also used to demonstrate the accuracy of the proposed method when the complexity in global motion is varied. The accuracy of the GME is also evaluated when the proposed robust feature tracker is employed.

- **Chapter 8**: The functional blocks in GME, addressed in all the chapters thus far, are combined into a unified framework for an adaptive and low-complexity GME pipeline for aerial video processing in this chapter. The interactions between the individual blocks is presented, highlighting how the inherent parallelism can be exploited and describing how the entire pipeline adapts to the varying image content, camera and object motion for achieving global motion estimation with minimal computation cost.

- **Chapter 9**: The critical conclusions drawn from this research work are consolidated and presented in this chapter. The future directions for research, based on this thesis, are also identified.

2

Literature Survey

Modern video surveillance systems are ubiquitous in urban as well as military areas. Conventional surveillance systems constitute fixed video camera installations such as the closed-circuit television (CCTV) setups, which relay the surveillance videos captured from the scene to a central server for live monitoring as well as archival. However, remote and mobile video platforms are seen as the next generation surveillance systems due to their ease of deployment and widespread coverage [6]. Unmanned aerial vehicles (UAVs), commonly known as drones, are being increasingly deployed for surveillance due to their flexibility in deployment as well as their ability to reach areas that are inaccessible for humans. In this chapter, the literature is reviewed for compute-efficient techniques that enable the development of such remote mobile video surveillance systems.

Unmanned Aerial Vehicles for Surveillance

Computers have become pervasive in our lives today, but in their early stage they were used only by the military and needed specialized training to operate. UAVs are now making a similar shift from being used only in the military sector to becoming easily available in the commercial markets [7]. Teal Group's 2015 market study estimates that worldwide UAV production will soar from current $4 billion annually to $14 billion, totalling $93 billion in the next ten years. Military UAV research spending would add another $30 billion over the decade [8]. Independent open-hardware projects such as the *Pixhawk* [9] have enabled faster setup and use of small drones at low cost, leading to their wide spread use.

UAVs are equipped with a variety of sensors such as, visual, multi-spectral, thermal and hyper-spectral sensors [10] to capture imagery of the target scene in multiple modes. Such UAVs are being deployed in a wide range of applications, as follows:

1. *Filmography*: Media and film productions employ aerial cinematography using commercial UAVs (such as the popular *DJI Phantom* series [11] and the tethered UAV *Fotokite* [12]) for movie productions and event coverage, providing a unique aerial perspective.

2. *Delivery of goods*: UAVs have been tested for autonomous transportation and delivery of goods (such as the *Matternet* drone [13] and *Amazon Prime Air* [14] service).

3. *Remote sensing*: UAVs have been seen as lower-cost and easy-to-use alternatives to manned flights or satellite imagery for routine remote sensing [15, 16].

4. *Post-disaster and emergency response*: UAVs are employed to assess the damage by collecting still and video imagery, after a natural disaster enabling the efficient distribution of rescue efforts, as such areas are often too dangerous or inaccessible for humans [17].

5. *Infrastructure inspection and monitoring*: Infrastructures such as oil pipelines, power transmission grids, solar panel arrays and bridges need to be monitored and inspected routinely for assessing their health and identifying any structural issues. UAVs have been used extensively to collect data for such inspections [18].

6. *Traffic monitoring*: A survey of methods applying UAVs for traffic monitoring can be found in [19] and demonstrates the applicability of aerial surveillance in monitoring traffic flow.

7. *Wide-area surveillance*: In [20], UAVs are considered for homeland/border security applications which need 24x7 surveillance to detect intrusions and illegal activities. Conservation teams use UAVs to monitor large forest areas to detect illegal poaching and to keep track of animals (such as the *Conservation Drones* project [21]).

In the applications that require monitoring of distant scenes (whether routine or in response to an emergency), collection of high-resolution still/video imagery using the on-board cameras on the UAVs is a primary task. Unlike the UAVs used in filmography, that are operated within line of sight by a human operator, the UAVs employed for surveillance cover wide areas in a routine manner. These areas are often inaccessible to humans. UAVs provide a safe, easily deployable and cost effective solution that requires less human intervention for wide-area surveillance/monitoring applications. Such UAVs belong to the class of remote and mobile video surveillance systems.

The UAVs come in a wide range of weights and capabilities: from micro UAVs weighing <0.9 kg, to medium and high altitude long endurance UAVs that weigh 100's of kg. However, the greatest uptake for commercial applications is predicted for the lighter end of the scale: for platforms with a weight of <15 kg, because this is where the cost benefits are likely to be most significant and the risk with a blunt force impact is reduced [22]. The airframes (mechanical structures) of such smaller UAVs can be classified into: rotorcrafts and fixed-wing planes. Rotorcrafts

use rotors (vertical propellers) for creating lift. Quadcopters (i.e. rotorcrafts that use four rotors) as in [23] and other rotorcrafts have seen a lot of research in recent years owing to their small sizes, ability to hover and hold precise positions for an on-board camera, which make them suitable for exploring indoor and cluttered urban scenes. However, for longer duration flights such as those necessary for the routine wide-area surveillance applications, fixed wing airframes have been preferred as they can carry greater payloads for longer durations with less power [21]. Also in terms of safety, when there is a failure, the fixed wing UAVs safely glide down without power, as opposed to how the rotorcrafts fall.

Military drones used for surveillance need to be operated by, (often a team of) human operators from a ground control station. In contrast, the navigation of the smaller UAVs, used for civilian outdoor wide-area surveillance applications, are typically autonomous, managed by an on-board auto-pilot software. In Fig. 2.1, an example of UAV flight planning [22] is shown. As shown in Fig. 2.1 (a), the flight path is predetermined by specifying waypoints on a satellite image or a map. In Fig. 2.1 (b) the overlapping images captured along the flight path during the flight is shown. Such a flight path, similar to a lawn mower's trajectory is typical for surveillance applications to get maximum coverage. The global positioning system (GPS) receivers provide the position information and an inertial measurement unit (IMU) provides the flight attitude information at any given time. The autopilot uses the position and attitude information to make the necessary course adjustments to keep the UAV on course. The autopilot can be overridden by a ground operator at any time. For indoor and GPS-denied environments, UAVs use on-board cameras [24] or range sensors for obstacle avoidance and dynamic path planning.

Evidently, UAVs deployed in modern surveillance systems are moving towards small sizes and the flight time of these battery powered UAVs is highly dependent on the system power consumption. This constrains the payloads and on-board computations of the UAV. At the same time, in order to operate with minimal human intervention, higher levels of autonomy in control and navigation is envisaged [25]. It is recognized that choosing an appropriate embedded platform

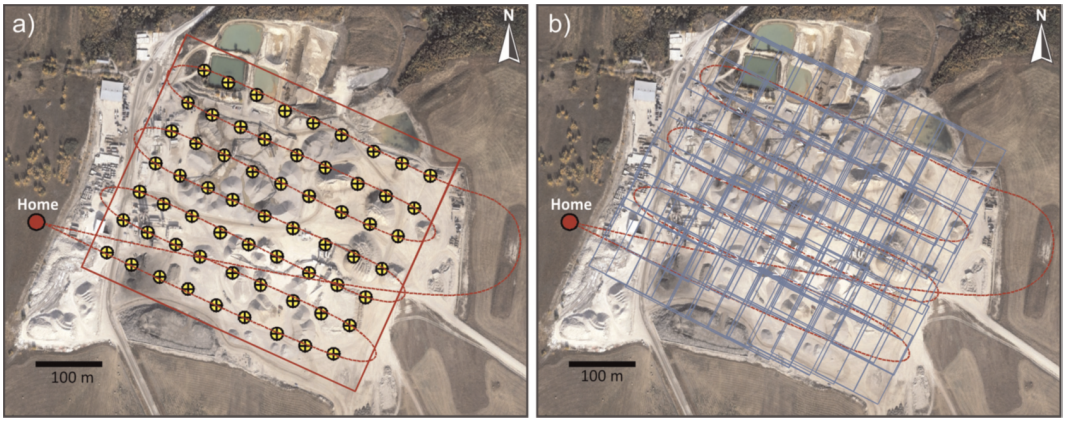

Figure 2.1: Flight planning example (a) showing image waypoints and flight lines and (b) image footprints with overlap. [22]

to perform the required on-board processing, to achieve higher autonomy, on the resource-constrained UAV platforms is a challenging problem [26].

In the next section, the vision-based algorithms that are required for surveillance by camera-equipped UAVs, are reviewed.

On-board Vision-based Tasks for Surveillance

Traditional video surveillance systems involve human operators who monitor the surveillance videos to detect anomalies. However, human monitoring suffers from fatigue and is highly ineffective in dealing with the huge volumes of video data being gathered continuously, leading to missed events [27]. Therefore in the past decade, intelligent and automated video surveillance systems, powered by computer vision algorithms for video analytics [28, 29] have been growing in importance. The human operator is replaced by a virtual operator that stays focused 24x7, detecting "interesting" events as they happen and then alerting human operators. This transforms the video surveillance system into a real-time, proactive and event driven process leading to swift response to events [27].

In the context of aerial video imagery from UAVs, video analytics is used to convert the large volumes of raw data captured, into useful information. For instance, crop statistics such as plant height and counts are generated for agricultural applications

in [30]. However these techniques are applied offline, on the ground, at the end of the flight, by processing the entire video data at one go. In contrast, wide-area surveillance scenarios need to respond to the events captured by the UAV in a time-critical manner. This requires the video analytics techniques to be performed in real-time and hence, on-board the UAV [25, 26]. In this section, vision-based algorithms that enable video analytics for surveillance are reviewed.

Motion Detection

Detecting and tracking moving objects [31, 32, 33] is a very common surveillance task for all surveillance systems. Motion cues represent an "interesting" event in the surveillance videos - for instance, moving animals or poachers in the forest area being monitored [21], vehicles on the highway under surveillance [19] and intruders in border patrol applications [20]. For a surveillance UAV, on-board detection of moving objects, can provide real-time alerts that can trigger necessary actions in a timely manner, without the need to wait for the UAV to return to the ground station.

Traditional motion detection algorithms are designed for stationary cameras and rely on a fixed background. They focus on accurately modelling the background and then applying background subtraction to segment the moving objects, also referred to as foreground [34]. However, the video streams captured by surveillance UAVs are challenging as they have moving backgrounds due to the motion of the camera. Therefore, the camera motion needs to be separated from the object motion, in order to detect moving targets. This is illustrated in Fig. 2.2 that shows the video processing chain for moving object detection, segmentation and tracking for UAV videos [32]. The *independent motion detection* separates the object motion from the camera induced motion and provides these as inputs to the subsequent object segmentation and multi-object tracking modules. This is achieved by the estimating the parameters for a model for global motion across successive frames (in this case, homography), caused due to the camera motion. This parametric

estimation is also referred to as global motion estimation (described in more detail in Sec. 2.4).

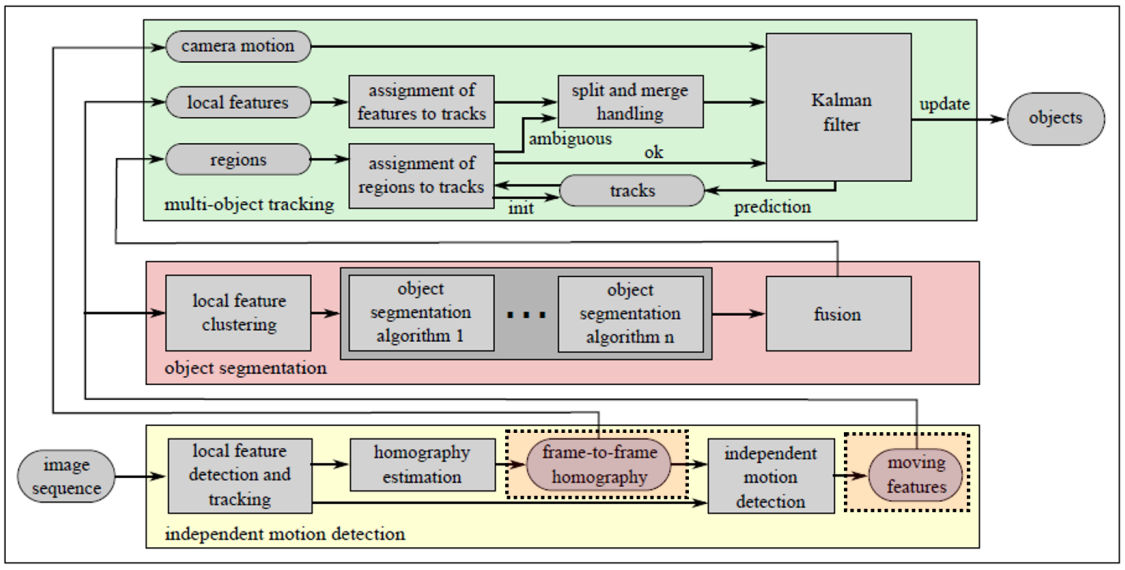

Figure 2.2: UAV video processing chain [32]: Outputs of the independent motion detection (dotted boxes) are necessary for object segmentation and tracking

The detected camera motion can be used in a motion compensation step to create a stable background [35]. Background modelling is then employed and the moving objects are segmented by a background subtraction. Several works skip the background modelling and subtraction by operating directly on the local optical flow vectors (vectors denoting the frame-to-frame motion of a local patch in the image). In [32, 33], the local optical flow vectors for features are determined, and then a motion compensation step removes the motion induced by the camera. The vectors that do not follow the camera motion are then analysed to segment the moving object from the frame. In [36] a motion detection algorithm has been implemented on FPGA showing real time performance, however it relies on simple block matching for camera motion estimation, that is less accurate than using optical flow vectors. The methods [31, 32, 33] that use optical flow vectors have not been demonstrated for embedded platforms.

Video Compression

For remote surveillance systems, *fixed* or *mobile*, the communication network is usually critical for efficient transmission of surveillance data [37]. The commercial drones that operate for short distances and durations have reliable communication channels: for instance, the *Fotokite* [12] sends uncompressed videos using a video down-link and the *DJI Phantom* series [11] transmit high-resolution video data by using broader channels like Wi-fi. However they have a very limited range of operation. UAVs used in wide area surveillance have a longer range of operation. Therefore, the coding technique used by the on-board vision system for encoding the captured videos becomes critical as the UAVs have the following constraints:

1. *Limited power/energy budget*: The transmission of video data from a remote camera to the ground stations is often the most power consuming activity and can affect the operational lifetime of the battery powered remote surveillance systems. Hence, on-board compression resulting in high coding gains, is critical to reduce the power consumption due to video data transmission [38].

2. *Communication data rate*: The video inputs are increasing in resolution from NTSC/PAL standard definition sensors (∼20MBps raw data) to high definition (HD) sensors (∼125 MBps). For instance, in the *Conservation Drones* project [21], Go Pro cameras are used with a resolution of 1080p60 - i.e. screen resolution of 1920x1080 pixels for frame rates of 25, 30, 50 and 60 frames per second. In [39], HD video is compressed for transmission from a surveillance UAV. However the communication bandwidth has not increased accordingly, and is at times more limited because of low data rate satellite links that are shared by multiple vehicles [37].

3. *Encoded video quality*: High degree of compression of the surveillance videos, for optimized use of communication bandwidth, may result in poor quality of decoded videos on the ground station. This leads to the high level video analytics, such as object classification and tracking [27, 28, 29], being misled

by the compression distortions. For instance, in [40], it is shown that conventional coding techniques may result in poor quality of the decoded video, especially in the context of surveillance videos, where the size of the most important moving targets are very small compared to the entire scene. In [41], it is shown that compression needs to be aware of the end application, for instance, object detection or tracking requires the compression engine to ensure the quality of the moving targets.

The conventional block based motion compensation technique, used in MPEG or ITU-T video compression standards, exploits the spatial and temporal redundancies present in all videos. However, the next generation of coding techniques that achieve high coding gains, go beyond the statistical properties of the frame content and extract the semantics, such as objects and their motion. Mosaic-based compression (MBC) technique [40, 42] falls in this category. This method was proposed for the compression of aerial videos because the predominant change across frames is the background motion induced by the moving camera. By efficiently modelling the stable background across frames in the form of a background mosaic, the moving objects are extracted as residuals for each frame.

Figure. 2.3 illustrates the mosaic-based compression pipeline. The algorithm first aligns successive frames by performing global motion estimation (as discussed in Section 2.2.1) and compensation. The residuals represent any new areas uncovered due to camera motion and/or moving objects in the scene, and are extracted by subtracting the frame with the background mosaic. For each frame, only the global motion parameters (that align the current frame with the previous frame) and the residuals need to be transmitted. The background mosaic is updated at both the encoder and decoder, and is used to reconstruct the frame from the motion parameters and residuals at the decoder. More recently, region of interest (ROI) coding [39] has been proposed for aerial videos. ROI coding also relies on a segmentation of background and moving objects for efficient compression of aerial videos.

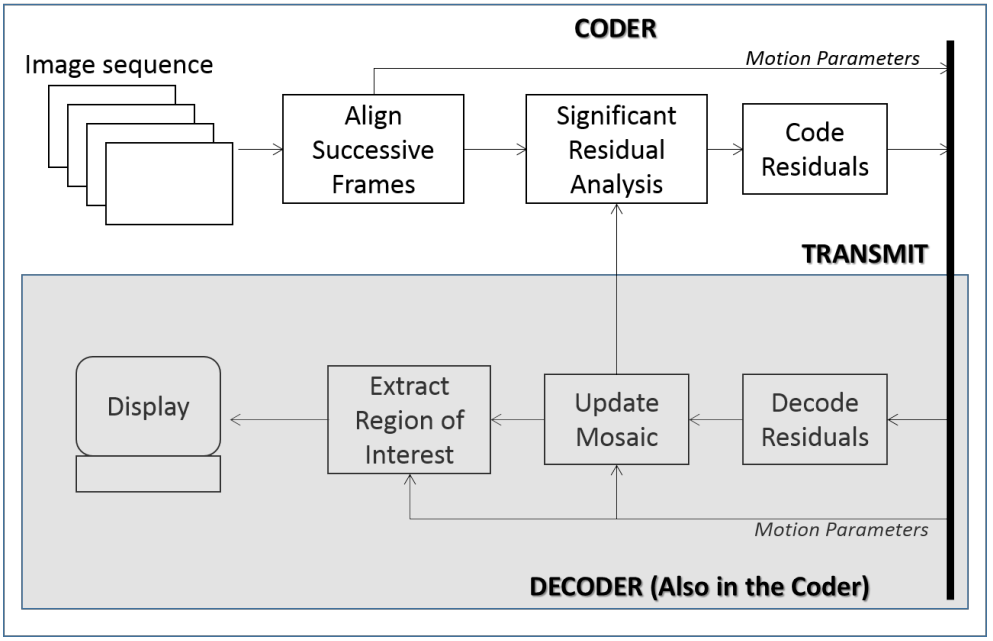

Figure 2.3: Block diagram for mosaic-based compression (MBC) [43]

These coding techniques are highly suited for the compression of aerial videos for the following reasons:

1. *Efficient coding*: The background that is required to be transmitted for each frame is represented by only a few parameters in the camera motion model. In addition, only the foreground and any misalignment errors are transmitted. As the foreground consists of very small moving targets, the MBC achieves very high coding gains in the context of aerial surveillance videos.

2. *Good subjective quality*: As the foreground is represented separately from the background, almost visually lossless coding of the foreground can be achieved resulting in high subjective quality of the decoded videos.

3. *Enhanced visualization*: The representation of the background as a mosaic also lends well with the visualization of the scene being monitored for the pilots/operators on the ground control station in the form of a dynamically updated panoramic background mosaic. Such a representation can also greatly enhance the decision making by human operators on the ground.

In [38, 44] the global motion estimation is incorporated into the H.264 encoder. In [45], the on-board IMU measurements are used to reduce the complexity of GME-based encoding.

Vision-aided Navigation

Vision-based control and navigation of the UAV [46] is an important task for GPS-denied cluttered/indoor environments such as the mini-UAVs in [23]. For the surveillance UAVs, in the absence of GPS inputs, on-board geo-registration has been proposed in [47]. This involves registering the current frame with a geo-registered image to derive the position estimates for the vehicle. Detecting a moving object and tracking it online can also help with autonomous navigation of the vehicle by target following - i.e. the moving object is kept in the centre of the frame [48]. In both these case, the motion due to the camera needs to be compensated for further processing of the aerial video frames.

In this section, it was seen that motion detection, video compression and vision-aided navigation are critical tasks that need to be performed in real-time and on-board the surveillance UAV. As the motion of the UAV induces artificial motion in all the frames in the video captured by the on-board camera, global motion estimation algorithm becomes the critical first step in any vision-based processing for aerial videos. In [39], it is seen that the global motion estimation step is still computationally intensive executing at $\sim$10 frames/second for HD video on a typical modern desktop PC [1]. Estimating the global motion requires the extraction of local motion, i.e. motion of the individual pixels or small patches as shown in the independent motion estimation pipeline in Fig. 2.2. Therefore, the next section surveys methods for local motion estimation which is followed by a detailed survey of global motion estimation.

[1] Intel Core i7-3770K CPU at a clock rate of 3.5 GHz

Local motion estimation

A local feature represents image patterns which differ from their immediate neighbourhood. They provide individually identifiable anchor points in an image, whose location can be determined accurately in a stable manner across time [49]. In literature, the terms *local* feature, *point* feature, *interest points* and *corners* are interchangeably used. In this thesis, the local features will be referred to as corners or features.

The feature correspondence between successive video frames provides the local motion of the pixels representing the features, i.e. for a feature at location (x, y) in the current frame, it's local motion is the displacement $\mathbf{d} = (d_x, d_y)$ such that the new location (x', y') of the feature in the next frame is obtained by $x' = x + d_x$ and $y' = y + d_y$. In literature, this is also referred to as *optical flow*. Feature correspondences are then processed for the entire frame in higher level algorithms such as global motion estimation, in order to derive the motion induced due to the camera.

Feature correspondences can be obtained in two ways:

1. *Feature detection and tracking*: Point feature (or corner) detectors such as Shi-Tomasi [50], Harris [51], FAST[2] [52] and SUSAN[3][53] detect patches in the image that have high degree of intensity variations, allowing them to be uniquely found in the successive frame. These features are represented by their locations. A feature tracker such as the Kanade-Lucas-Tomasi (KLT) [54] is used to locally search for the feature patches in the successive frame, around the current location, to determine the correspondence.

2. *Feature detection/descriptor and matching*: When frames undergo a large degree of change in viewpoint (due to rotation or scale change), a more sophisticated descriptor is employed to represent the feature in a scale and

[2]Features from Accelerated Segment Test
[3]Smallest Univalue Segment Assimilating Nucleus

rotation invariant manner. SIFT[4] [55] and SURF[5] [56] are examples of widely used feature descriptors. The feature descriptors are computed for both frames and then matched using a similarity measure to obtain the feature correspondence.

A feature detector and tracking algorithm has been extensively used for video sequences [35, 39, 57] as it can provide accurate correspondences for the small local motion experienced between frames. However, when the viewpoint variation is significant between two images, the feature descriptor and matching algorithm is needed to compute the feature correspondences. It has to be noted that computing feature descriptor is computationally intensive, and therefore, a feature detector and tracking algorithm is used for fast feature correspondence computations in video sequences.

Feature Detection

Excellent surveys on corner detectors proposed in the last 30 years can be found in [49] and [58]. In this section, the focus is on related work in corner detection from the standpoint of efficiency. Earlier work on corner detection involved looking for high curvature points along the contours in the image which typically correspond to true corners in 3D. In [59], such a method is proposed that looks for maxima of curvature where the gradient is large. Recent work in corner detection selects points that are robust, stable and distinctive, that need not always correspond to true corners [49]. As distinct patches exhibit a large variation in the pixel intensity compared to the neighbouring pixels, detectors that compute the second-derivatives of intensity into a Hessian matrix, have been proposed [49]. However, they have been shown to be relatively less reliable. Shi-Tomasi [50] and Harris [51] corner detectors compute an auto-correlation matrix using the first-order derivatives of the intensity values and this matrix represents the degree of intensity variations in various directions around a pixel. SUSAN detector [53]

[4]Scale Invariant Feature Transform
[5]Speeded Up Robust Features

operates directly on the image intensity (without the use of derivatives), by computing the fraction of pixels within a neighbourhood that have similar intensity as the centre pixel. FAST [52] extends this idea to consider only pixels on a circle around the centre pixel and uses an efficient decision tree to classify the centre pixel as a corner.

It is well recognized that corner detection is a compute-intensive step. There are typically two approaches that have been reported in the literature for increasing the computation efficiency of corner detection:

1. *Hardware acceleration*: Hardware acceleration techniques have been proposed to exploit the inherent parallelism in the corner detectors. Efficient field programmable gate array (FPGA) implementation for SUSAN has been presented in [60]. Although FAST is computationally simpler to Harris and Shi-Tomasi detectors, recent evaluations [61] have shown that FAST can be unreliable in many scenes, and hence, the Harris detector is preferred. The evaluations in [58] also show that Harris and Shi-Tomasi achieve among the best results in low-level corner detection. Numerous approaches to accelerate these detectors have, therefore, been reported. Various fast implementations of Harris have been proposed in the literature. The target accelerator platforms include application specific integrated circuit (ASIC) [62], FPGA [63], Cell Processor [64] and single instruction multiple data (SIMD) architecture [65]. In [66], a simpler floating-point format is used by customizing instructions on the Nios-II processor. In [67], a hardware implementation that performs Harris corner detection on a rank transform image instead of the original image is presented. FPGA implementation for Shi-Tomasi in [68] employs an alternative corner measure that uses only integer arithmetic consisting of additions and multiplications and avoids the transcendental operations. In [69] and [70] the corner detection step for Shi-Tomasi is implemented on graphics processing unit (GPU) and the final step of non-maximal suppression (NMS) is parallelized.

2. *Algorithm innovations*: Low-complexity in corner detection is also achieved by modifying the algorithms, independent of the underlying hardware architecture. In [71], the time for corner response computation is kept constant, without depending on the window size, by the use of the integral image representation. In [72], a pruning technique is proposed that selects pixels with high gradient magnitude as corner candidates for Shi-Tomasi and Harris algorithms.

As the scene content in aerial videos changes drastically, scene-adaptive corner detection techniques are of interest. In [73], an adaptive sampling technique is proposed for FAST corner detector, which locally adapts the computation steps for corner detection based on the image content, which results in lower computations in homogeneous regions that are not likely to contain corners. Optimal parameter selection for corner detection is another way to be adaptive to scene content. The problem of automatic optimal parameter selection has been addressed in the context of edge detection in [74] and later generalized to all types of features in [75]. It involves statistical analysis of detections, by using a range of parameter sets, and identifying the set that results in maximum number of useful features and minimum amount of noise. However, it is not practical to use this technique in a real-time resource-constrained system, as the detector is executed as many times as the evaluated parameter sets. The OpenCV implementation (*dynamicadaptedfeaturedetector*) that selects the optimal parameters for detecting corners in [76] also requires multiple executions of the detector and hence, it is not a low-complexity solution.

Feature Tracking

The Kanade-Lucas-Tomasi (KLT) feature tracker [54] is widely used to compute the feature correspondence that represents the local motion of a feature. The goal of KLT feature tracker is to find the displacement $\mathbf{d} = (d_x, d_y)$ by which the feature patch at (x, y) in the current frame has moved to a new location

$(x + d_x, y + d_y)$ in the next frame. By relying on the patterns of image intensity gradient surrounding the feature, KLT examines far fewer potential matches to find the corresponding location in the next frame. Two basic assumptions are made: (1) Intensity changes smoothly with position (2) Intensity pattern of the feature itself does not change over time. These assumptions enable the formulation of a least-square estimation problem that iteratively determines the displacement $\mathbf{d}$ through a Newton-Raphson method, such that the residual error between the feature patch and the potential match is minimized. A detailed derivation of the mathematical formulation for KLT is provided in Appendix A.

Although KLT can be applied to any image patch with sufficient intensity variations in its neighbourhood, it is often combined with the Shi-Tomasi [50] corner detector. This is because the corner measure computed by Shi-Tomasi is derived from the KLT algorithm itself - essentially, the patches that *track well* with KLT are detected by the Shi-Tomasi as *good* features.

The KLT tracker employs linear approximation in its core step (refer to Eq. A.5 in Appendix A) and this works only if the displacement $\mathbf{d}$ is very small. In order to overcome this limitation, a multi-resolution pyramidal approach was proposed in [77] to deal with larger inter-frame displacements as shown in Fig. 2.4. The intuition is that a large motion can be reduced to a small displacement if it is considered for the sub-sampled image. Therefore, the image I is sub-sampled L_m times resulting in a pyramidal representation of the form $\{I_0, I_1, ..., I_{L_m}\}$ where I_0 is the original image and I_1 is obtained by sub-sampling I_0 by 2, and so on. A large displacement of $\mathbf{d}$ can then be captured at a higher level of the image pyramid I_2 as the disparity at this level $\mathbf{d_2}$ is small enough for the linearity assumption in KLT to work.

As it is computationally simple, KLT is preferred on low-resource platforms onboard UAVs to compute feature correspondences [78]. However, the classical KLT (as described in Appendix A) assumes that the patch around the feature undergoes only translation motion. Therefore, in the presence of rotation and scaling, KLT is

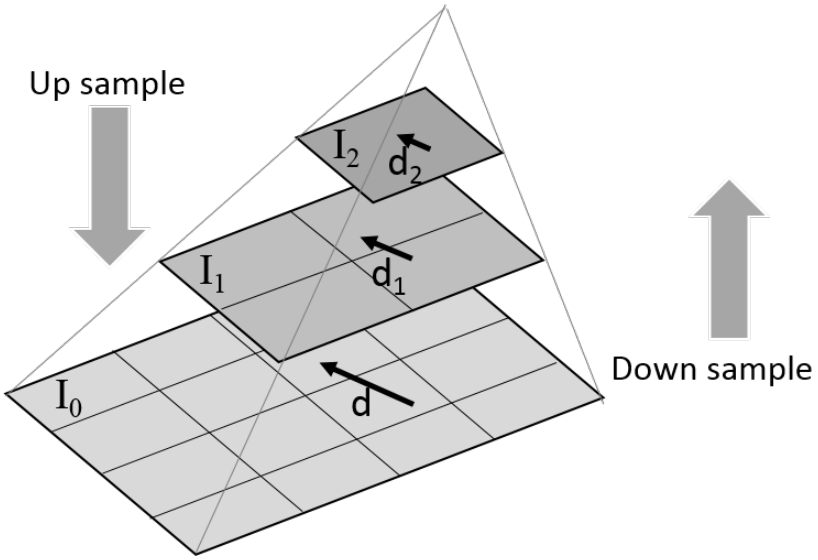

Figure 2.4: Pyramidal approach for KLT illustrated with a 2-level pyramid ($L_m = 2$)

known to suffer from high inaccuracies [78]. This is a severe limitation of KLT especially for surveillance UAVs as drastic frame-to-frame rotation is common in aerial videos when UAVs perform bank turns [78] as shown in Fig. 2.1. In [79], an affine formulation is proposed that replaces the translation model with an affine model in the KLT computations, allowing the feature patch to undergo rotation, scaling and skew, in addition to translation. However, this is highly compute-intensive and real-time performance is achieved only with GPU implementations [80]. In [78, 81], the measurements from on-board inertial measurement units (IMUs) are used to provide initial estimates for KLT, eliminating the rotation component of the motion. In [82], a fixed window size is replaced with an exhaustive sampling of window sizes is employed, to select an optimal window size, in order to improve the accuracy of KLT. In [83], it is shown that by increasing the window size of the KLT feature tracking step in the original high resolution frame (I_0), the benefits of a pyramidal approach can be obtained without the need for the computationally costly image pyramid generation. Therefore, window size selection is an important consideration for the accuracy of KLT feature tracker.

As the accuracy of the tracking algorithm is critical for the subsequent algorithms that use the feature correspondences as input, several works focus on evaluating the tracks obtained from KLT. Error estimates are proposed in [84, 85], that

are used to evaluate the feature correspondences reported by KLT allowing the end-user application to qualify the correspondences. In [86], theoretical error estimates for KLT tracks is presented, however it is reported that the error estimate computations are so high that they cannot be used in real-world applications in an online manner. When a feature patch is tracked over many frames, its appearance starts to change causing a drift error in KLT. To deal with errors during long-term tracking, a more sophisticated appearance model is incorporated in [87, 88]. A rejection rule that allows the tracker to automatically reject bad tracks was proposed in [89].

Global Motion Estimation

Videos captured by moving cameras contain *global* motion across frames due to the motion of the camera. This *global* motion can be modelled using a 2D parametric transformation model that defines the displacement of each pixel in the frame due to the global motion [90]. Global motion estimation (GME) is the process of estimating the parameters of this global motion model. The GME is illustrated in Fig 2.5 where the difference between successive frames in a video sequence is shown *before* and *after* global motion estimation has been employed. As is clearly seen, after the global motion due to the moving camera has been compensated, using the global motion model, the residuals are only the moving objects (e.g. the white car in Fig 2.5) and the new regions uncovered due to camera motion. Once the camera motion has been eliminated, further processing for mosaic-based compression or motion detection as described in Section 2.2 can be performed.

Global Motion Models

The global motion model uses a 2D parametric transformation model T to define the motion of each pixel from (x, y) in the current frame to (u, v) in the next

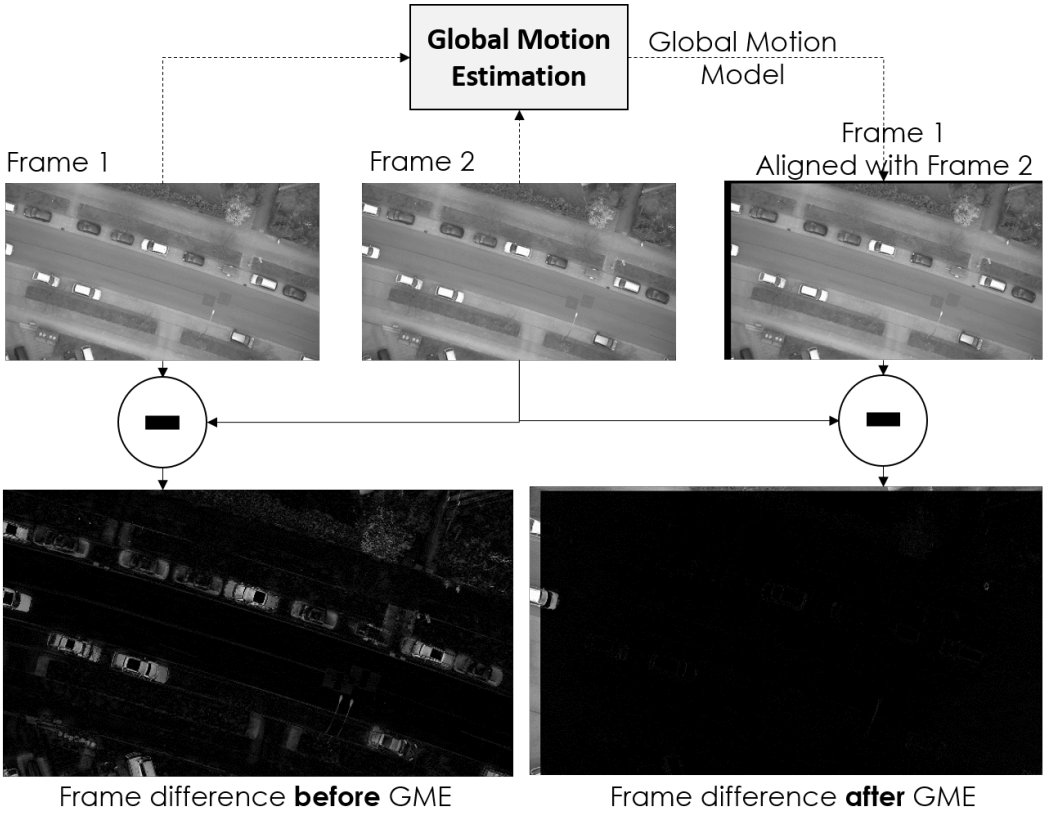

Figure 2.5: Illustration of global motion estimation on aerial video

frame, caused by global motion, as shown:

$$(u, v) = T[(x, y)] \tag{2.1}$$

The simplest motion model is a 2-parameter *translation* model:

$$\begin{aligned} u(x, y) &= x + t_1 \\ v(x, y) &= y + t_2 \end{aligned} \tag{2.2}$$

A more frequently used model [90, 91, 92] is the *affine* motion model with 6 parameters:

$$\begin{aligned} u(x, y) &= a_1 x + a_2 y + a_3 \\ v(x, y) &= a_4 y + a_5 y + a_6 \end{aligned} \tag{2.3}$$

The model represents the motion of a planar surface under orthographic projection, and therefore it assumes that the depth map of the scene is small compared to the distance of the camera from the scene. Hence, it is a very good approximation for the global motion of the image when the camera is viewing *distant* scenes e.g. aerial videos captured by on-board cameras on surveillance UAVs.

The 2D *projective* model with 8 parameters, also called a homography, [93, 94] relaxes the constraint of the affine model and allows for the representation of the motion of a planar surface under perspective projection:

$$\begin{aligned} u(x,y) &= \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} \\ v(x,y) &= \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1} \end{aligned} \tag{2.4}$$

A more complex motion model with 12 parameters that allows more freedom to the deformation of the observed 2D scene is a non-linear parabolic model [91].

The choice of the motion model depends on the type of the depth structure of the scene with respect to the location of the camera and the type of camera motion expected. For inter-frame estimation of distant scenes, a homography sufficiently describes the global motion. But over a longer period, i.e. if global motion needs to be estimated and compensated for a large number of frames, then more sophisticated models are necessary.

Parameter Estimation Approaches

The parameter estimation method used in the GME can be broadly classified into:

1. Pixel-based or direct methods [90]
2. Feature-based methods [95]

The main difference between the two approaches is that in direct methods, all the pixels in the image are used to derive the global motion parameters, whereas in

feature-based methods, only feature correspondences are used in the parameter estimation.

In pixel-based GME, the parameters are estimated by the minimization of the prediction error between the two frames, using the iterative gradient descent method. The advantage of a pixel based GME is that it provides highly accurate motion estimation parameters with sub-pixel accuracy. Accurate GME is essential for:

- *High coding gains*: When the GME is accurate, the residuals coded for each frame relative to the mosaic due to misalignment are minimized. Hence, the overall amount of data sent with each frame is lesser.
- *Accurate foreground extraction*: For foreground extraction, the residuals due to misalignment are noise and keeping them to a minimum with an accurate GME can improve the accuracy of the moving objects extracted.

The main limitations of the pixel-based GME are [95]:

- *High computational complexity*: The complexity of the pixel based GME is very high as it relies on iterative energy minimization algorithms that operate on the entire image.
- *Small pixel motion assumption*: Pixel based GME formulation involves a linearization step that assumes the pixel motion to be small between the two frames. Hence, in the absence of a good initial estimate, the direct methods can be stuck in local minima and/or never converge.
- *Bias of estimation due to outliers*: Outliers are created by local object motion, i.e. pixels belonging to moving objects, and they can bias the estimate of global motion parameters.

The feature-based GME methods [95] are developed on the fundamental basis that for reliable estimation of the parameters of the global motion model, only those regions in the image that can produce good correspondences should be used; point features or interest points [50] are examples of such regions.

The feature based GME [57] applies a robust model-fitting algorithm such as the Random Sample Consensus (RanSaC) [96] on the feature correspondences between two successive frames, classifying them into inliers and outliers and computing the global motion model that fits the inliers.

The advantages of the feature-based GME over the direct methods are [95]:

1. *Invariance*: The features have photometric and geometric invariance. That is, between frames, when there is change of illumination or viewpoint, the same 3D point is still detected as a feature in the images. This allows for obtaining reliable feature correspondences.

2. *Optimal estimation*: The feature-based approach lends itself well to GME over a long sequence, as this provides a means to compute a maximum likelihood estimate of the estimated quantities, i.e. the affine or perspective model parameters. For direct methods, it is not straightforward to write down a practical likelihood function for all pixels.

3. *Computational efficiency and convergence*: Features accelerate the process of estimating the global motion but with good accuracy. Use of all the pixels, as in direct methods, potentially introduces many outliers that not only increase the complexity of the estimation process but can also result in incorrect convergence of the minimization.

Hence, use of features leads to a solution of the GME with good (less noisy) data. The lower computational complexity of the feature-based GME compared to direct methods makes it attractive for use in low-resource platforms.

State-of-the-art GME techniques

The limitations of direct methods have been extensively addressed in recent research in GME [92, 97, 98]. The high complexity of direct GME methods, owing to the use of pixels from the entire image, has been addressed by selecting appropriate pixel subsets, and then performing the iterative error minimization. In [92],

it was shown that the pixels from the entire image need not be used for the prediction error. In this approach, only those pixels with the largest gradient magnitude are chosen for the application of the direct method GME. In order to ensure that the entire frame is well represented the image is divided into 100 sub regions and the top 10% of the pixels are chosen. In [97], a similar pixel sub-sampling is used. However, it is based on regularized patterns of pixel locations and hence is less complex than [92]. It was shown in a later work [98] that the regularized patterns as in [97] can have poor registration accuracy compared to gradient-based pixel subset selection as in [92].

Direct methods also assume very small displacement between frames due to global or camera motion. For large translations, direct methods can get stuck in local minima. In [93], a modified n-step search is used at the lowest resolution of a Gaussian image pyramid to get the initial estimate for the translation components. In [91], a feature-based GME is applied to get the initial estimates of the translational components. Once the translation components are found, more complex models are applied for direct error minimization, after generating the predicted image by applying the translation parameters.

There are attempts to reduce the complexity of the gradient descent step in direct methods. In [92], an interpolation free formulation of the gradient descent method is proposed that avoids intensity interpolation computations for non-integral pixel locations in the gradient descent iterations. In a recent work [98], the full precision images are replaced with 1-bit width images in the gradient descent. Hence, all the full precision arithmetic operations are replaced with logical operations reducing the complexity significantly.

Direct methods for GME are also easily biased by outliers due to local motion. In [93], a robust M-estimator is used with a truncated quadratic error function. The modification applies a threshold to the absolute error such that only those pixels with the error below the threshold are used in the iterative parameter estimation. The idea behind this is that the absolute error of a residual pixel that belongs to

a moving object will be much higher compared to the absolute error of a residual pixel due to misalignment. In [91], an M-estimator is used with the feature matching step to prevent the bias due to outliers.

With the widespread use of encoded MPEG video streams, global motion estimation using data in the compressed domain has seen a lot of research contributions. The motion vectors contain information about the motion of the various parts of the scene and are computed by the compression engine. Hence, they can be reused for the parameter estimation process. Also the GME can happen without the need to decode the MPEG video. In [99], a background mosaic is built by analysing the MPEG motion vectors directly. Recent work [100, 101] has focused on increasing the robustness of the parameter estimation using the MPEG motion vectors based on imaginary line tracking and Helmholtz principle. In [102] however, the motion-vector based GME was compared with direct methods, and it was shown that a simplified GME on down-sampled image pairs achieved better accuracy than the motion-vector based techniques. The drawback of motion vectors is that the motion vector represents the block with the least residual energy that can be used to predict the current block, which need not represent the true motion of the block [41].

Feature based methods have been adopted in GME due to their lower complexity as the features are sparse yet reliable representations of the scene [95, 103]. Compared to direct methods, they can better handle large motions and outliers. In [94], the Harris corners are used as features and the frame to frame alignment is obtained. In [57], an evaluation of a feature-based GME system is provided. A motion prediction step is introduced that exploits the fact that the motion between two frames will be mostly smooth which reduces the search range to find the feature correspondence and hence reduces complexity. In [104], point crossings of edges are used as features and parameters for the affine camera motion model is estimated using the feature correspondences.

The feature-based methods do not give sub-pixel accuracy compared to the direct methods [90]. In order to improve the accuracy, the feature matching step has

been replaced by a feature tracking step (as described in Section 2.3.2) using an optical flow based feature tracker such as the Kanade-Lucas-Tomasi(KLT) feature tracker [54]. The tracker is applied on each point feature detected by the feature selection step to find its correspondence in the reference frame, and it gives sub-pixel accurate local motion vectors for the point features. The use of the KLT feature tracker in combination with the Shi Tomasi feature selection has been explained in [105]. Harris corners and the KLT feature tracker is used in [39] for the global motion estimation. In a recent evaluation [106], it has been shown that the KLT feature-based image mosaicing gives superior results. In [107], such a KLT tracker is used to estimate the global motion parameters and this is shown to be more accurate than the method using the MPEG block-based motion vectors.

Tackling large displacements between frames is an important consideration for all GME algorithms. Almost all GME techniques use some form of hierarchical estimation. This involves the use of several levels of lower resolution (sub-sampled) versions of the frames called an image pyramid. The estimation starts at the lowest resolution and the results are propagated as initial estimates to the higher levels. The need for hierarchical estimation is to be able to handle large displacements due to fast motion [108]. In [93], a coarse-to-fine strategy is proposed for the direct GME method where the GME is performed at the lowest resolution of a Gaussian image pyramid and the results are propagated to higher resolution levels as initial estimates.

The intuitive argument for the use of image pyramids is computational efficiency. If large displacements can be retrieved from low resolution image information, then high degree of savings in computations can be achieved. The higher resolution images can then be used to refine the results obtained from the lower resolution images and improve the accuracy. However, another important consideration is that it is not just efficient but is necessary to remove the high resolution details in order to correctly retrieve the large displacements, in the absence of which there is a high likelihood of the estimation process getting stuck in local minima.

Inertial measurement units (IMUs) are an integral part of the on-board navigation systems in UAVs. In [109, 110], meta data is collected from external sensors on the moving vehicle that is fused with global motion estimation techniques. In [109], the aerial videos are of low resolution and hence a conventional image pyramid approach will fail to give good results since at lower resolutions, there will be very less features to detect and track. Meta data that contains additional information about the camera position is extracted using on-board sensors and is used as initial estimate for the GME. In [110], a sensor-assisted video encoding scheme is presented that uses low-power and low-cost sensors such as digital compass and accelerometer to provide camera motion inputs for the GME. The scheme has been integrated with the H.264/AVC encoder and implemented on mobile phones.

Feature-based GME with KLT Feature Tracker

A good compromise between sub-pixel accuracy and low complexity is the feature-based GME method that employs a KLT feature tracker for feature correspondences. Figure. 2.6 illustrates this scheme. As described in Sec. 2.3, a feature detection followed by feature tracking algorithm provides the local motion estimation. These feature correspondences are then fed into the robust estimation algorithm, random sample consensus (RanSaC) to generate the GME parameter values.

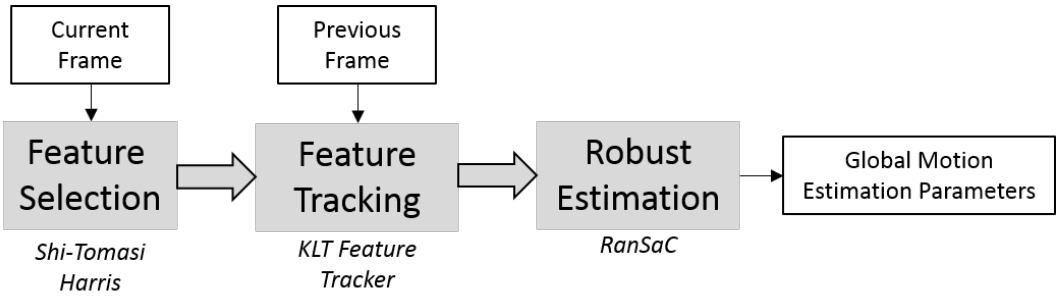

Figure 2.6: Feature-based GME pipeline with KLT feature tracker [39]

As GME needs to be performed on each frame of the video, it is critical that the underlying algorithms are low-complexity and real-time. In [39] it was recognized

that the number of features used for GME contribute to its complexity and sparse feature sets could reduce the GME complexity. It has also been recognized that the coverage of the frame by the feature set is a critical component to the accuracy of the GME [111].

As RanSaC [96] is a widely used robust estimation algorithm, several improvements have been proposed to reduce its complexity. RanSaC uses a hypothesize-and-verify framework in an iterative manner to arrive at the largest inlier set - set of feature correspondences that agree to a global motion model. In [112], a pre-evaluation stage is used to quickly discard bad hypotheses. While traditional RanSaC treats all feature correspondences equally, PROSAC (PRogressive SAmple Consensus) [113] orders the sampling process based on the quality of the feature correspondences, considering the highest quality correspondences first. In [114], many hypotheses are scored simultaneously, in a breadth-first manner as opposed to the depth first approach of the traditional RanSaC, thereby giving a fixed computation time which is considered suitable for real-time implementations. In all these works, it is still assumed that a large number of feature correspondences are available for robust estimation.

Summary

In this chapter, it is seen that modern surveillance systems are increasingly remote and mobile such as UAVs. Therefore they require on-board processing of several vision tasks, such as motion detection, mosaic-based compression and control/navigation for efficient and autonomous operation. However, these battery-powered systems are also highly resource constrained with strict payload limits and therefore the vision-based methods need to be compute-efficient to be deployed on-board. Global motion estimation was identified as the first step for the vision-based surveillance tasks.

The aerial videos captured on surveillance UAVs exhibit diverse scene content and illumination conditions. The typical trajectories of the UAVs suggest that

the camera motion also varies significantly. For the on-board vision tasks to be executed seamlessly without human intervention, the algorithms are traditionally setup to deal with the worst case scenarios. For example, even though the motion of the UAV is smooth - in order to cater to an occasional fast rotation, a feature descriptor and matching pipeline is used instead of a feature detector with a tracker. Such design decisions in favour of autonomy result in high computational complexity. There are several concerns with respect to the feature-based GME in the context of the low-resource platforms on the surveillance UAVs:

1. *Complexity of feature detection*: Feature detection is an essential first step for all the vision tasks on-board. However, it involves complex computations on every pixel of the image. Efforts to speed up the feature detector have involved hardware accelerations using GPU [69, 70]. However, they do not address the opportunity to reduce the feature detection computations based on the varying image content, which is a characteristic for surveillance UAVs.

2. *Prefixed threshold for feature detection*: The threshold used for determining "good" corner regions is typically set empirically. However, the illumination and scene content changes are drastic for aerial videos. In order to deal with this wide variation, the thresholds are set conservatively resulting in unnecessary computations when the scene contrast is favourable with a large number of patches with "good" candidates. Being adaptive to the scene content can not only make the detector work in a wide range of scenarios but also adapt the computations accordingly, resulting in low-complexity feature detection. However, existing work [74, 75, 76] on the selection of optimal thresholds involve exhaustive evaluations of the various parameters and are not suited for low-complexity platforms such as the surveillance UAVs.

3. *Robustness of KLT*: KLT is a preferred tracker because of its low computational complexity. However, it is not robust to the rotation/scaling motions that the UAVs incur during their flight. This has been tackled in literature using sensor fusion [78, 81] increasing the computational complexity. Using

the affine formulation [79] also brings in robustness at a high cost. Improving the robustness of KLT to rotations with a low computational complexity is critical in order to make it viable for aerial videos which undergo fast rotations/scale changes infrequently.

4. *Number of features*: Although the number of features used for GME has a direct impact on the computation complexity, in literature it has always been assumed that a dense feature set is needed for GME. However, as in all estimation problems, the feature-based GME is also an over-constrained system - i.e., a large number of feature correspondences are used to cope with inaccuracies and outliers (foreground objects). From a computational efficiency standpoint, this leads to wasted computations for surveillance videos, as the camera motion is predominantly simple and the number of outliers due to foreground is less. Therefore the ability to adapt the number of features to the scene, can result in large savings in computational complexity in aerial videos. In literature, there is no systematic method to determine the optimal feature set for GME for a given camera motion and number of moving objects.

In the following chapters of this thesis, these concerns are addressed one by one, developing low-complexity techniques for global motion estimation in aerial video processing.

3

Low-Complexity Pruning for Corner Detection

Introduction

Corners, also known as features or interest points, represent identifiable anchor points in the image. Corners are used for matching (e.g. image registration), tracking (e.g. object tracking) and as robust image representation when combined with feature descriptors for object recognition.

Several corner detectors have been proposed in the literature [49, 58] and comparative evaluations have shown that the Shi-Tomasi [50] and Harris [51] corner detectors achieve some of the best results [58, 61]. Also, wide range of recent real

time applications [25, 115, 116, 117, 118] have used Shi-Tomasi or Harris corner detectors.

However, both the algorithms require a complex corner measure computation for every pixel in the image. This step is highly compute intensive requiring floating point arithmetic and becomes a bottleneck for real time vision tasks. Reducing the computational complexity of these corner detection algorithms is essential in low cost and low power embedded systems, especially those that do not support an on-board floating point unit (FPU).

As mentioned in Section 2.3.1, architecture-dependent approaches [68, 69, 70] have been proposed for reducing the corner detection complexity and/or making it real-time. Algorithmic innovations to reduce complexity such as [71, 72] are independent of the underlying architecture. The proposed method belongs to this category and can potentially result in higher computation savings when used with the architecture specific solutions discussed earlier.

In this chapter, a low-complexity pruning technique for Shi-Tomasi and Harris corner detectors is presented, to efficiently discard non-corners, significantly reducing the selection and evaluation effort for the presence of corners to only corner-like regions. The approach proposed in [72] has a similar motivation as the work in this chapter. However, it uses the gradient magnitude of a pixel alone as an indicator of the corner and does not consider the intensity pattern of the pixel neighbourhood. This can potentially miss many good corners. Also, the use of integral image in [71] shows substantial savings only for large window sizes. Small window sizes, i.e. 3x3, have been shown to be sufficient for Shi-Tomasi/Harris detectors [77, 117]. In addition, while the work in [69] and [70] achieves speedup by exploiting parallelism on GPU, they are not well suited for low cost/power embedded systems. Moreover, compared to [68, 69, 70, 71], the proposed technique computes the corner response only on a small set of corner candidates instead of the entire image. As shown in the experimental results, the proposed method leads to significant computation savings without compromising the accuracy of corner detection.

The rest of the chapter is organized as follows. In the next section, the Shi-Tomasi and Harris corner detection algorithms are introduced. In Section 3.3, the proposed pruning technique for the Shi-Tomasi and Harris algorithms is described. Section 3.4 provides computational complexity analysis of the proposed method in comparison with other related work. Section 3.5 presents actual implementation results on the Nios-II processor to evaluate the accuracy and performance of the proposed technique with a set of image benchmarks. The chapter concludes in Section 3.6.

Shi-Tomasi and Harris Corner Detectors

Figure 3.1 shows the steps in the Shi-Tomasi and Harris corner detection algorithms. Both the corner detectors compute a corner measure C on all the pixels in the image, based on the local auto-correlation function that is approximated by matrix M over a small window W for each pixel *p(x,y)*:

$$M = \begin{bmatrix} \sum_W w(x) I_x^2 & \sum_W w(x) I_x I_y \\ \sum_W w(x) I_x I_y & \sum_W w(x) I_y^2 \end{bmatrix} = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \tag{3.1}$$

I_x and I_y are horizontal and vertical gradients respectively, and *w(x)* is an averaging filter that can be a box or a Gaussian filter. The eigenvalues λ_1 and λ_2 of M (where $\lambda_1 \geq \lambda_2$) indicate the type of intensity change in the window W around *p(x,y)*:

- If both λ_1 and λ_2 are small, *p(x,y)* is a point in a flat region.
- If λ_1 is large and λ_2 is small, *p(x,y)* is an edge point.
- If both λ_1 and λ_2 are large, *p(x,y)* represents a corner point.

Shi-Tomasi directly computes the smaller eigenvalue λ_2 as its corner measure C as shown in Eq. 3.2:

$$C = \lambda_2 = \frac{1}{2}\left((a+c) - \sqrt{(a-c)^2 + 4b^2}\right) \tag{3.2}$$

Harris combines the eigenvalues into a single corner measure R as shown in Eq. 3.3, which avoids the explicit computation of eigenvalues. In Eq. 3.3, k is an empirical constant ($k = 0.04$ to 0.06).

$$C = R = \lambda_1\lambda_2 - k\left(\lambda_1 + \lambda_2\right)^2 = det(M) - k.trace(M)^2 = (ac - b^2) - k(a+c)^2 \quad (3.3)$$

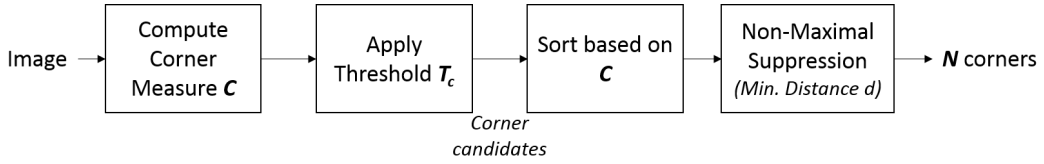

Figure 3.1: Block diagram for conventional Shi-Tomasi and Harris corner detectors

As shown in Fig. 3.1, once the corner measure C for every pixel is computed, a threshold T_c is applied on the corner measures to discard the obvious non-corners. The remaining pixels are the corner candidates, which are then ranked in the descending order of the corner measure C. Candidates with the highest corner measure are selected as final corners such that they are separated from other selected corners by a user specified distance d. This is referred to as non-maximal suppression (NMS). In this process, if a candidate is within the vicinity of an already selected corner, it is discarded. The maximum number of corners (N) that need to be extracted, the threshold T_c and the minimum distance of separation between corners (d) is specified by the user.

Pruning Technique for Corner Detection

The following observations are made on the Shi-Tomasi and Harris detectors:

- In most images, the obvious non-corners (i.e. the flat and edge regions) constitute a large majority of the image. Hence, the Shi-Tomasi and Harris detectors incur a lot of redundant computations as they evaluate the entire image for a high corner response.

- For Shi-Tomasi, expanding Eq. 3.2 gives

$$\lambda_2 = \frac{1}{2}\left((a+c) - \sqrt{(a-c)^2 + 4b^2}\right) = \frac{1}{2}\left((a+c) - \sqrt{(a+c)^2 - 4(ac-b^2)}\right) \tag{3.4}$$

  λ_2 is most influenced by the term $(ac - b^2)$ as the two $(a + c)$ terms cancel out. For a good corner, λ_2 needs to be a large value. Hence maximizing $(ac - b^2)$ which is also $det(M)$ can select good Shi-Tomasi corners without explicit eigenvalue computation.

- For the Harris corner measure as in Eq. 3.3, the $trace(M)$ term is introduced so that edges can also be detected. Ignoring the $trace(M)$ term, the $det(M)$ term alone is sufficient to select corner regions.

Based on the above observations, $det(M)$ is the key term in the corner measures for both Shi-Tomasi and Harris, and pixels that maximize $det(M)$ represent good corner candidates. Therefore, a low-complexity pruning technique is proposed, that employs simple approximations of the $det(M)$, to quickly remove the non-corners. The overall flow of the proposed technique is presented in Fig. 3.2. Note that $det(M) = ac - b^2$ consists of two terms and the proposed pruning technique is divided into two steps that handle each term successively: PP that selects pixels with a large value for ac, followed by ER that discards pixels with a large value for b. In effect, PP and ER applied together select pixels that maximize $det(M)$. Each of the pruning steps is described in detail next.

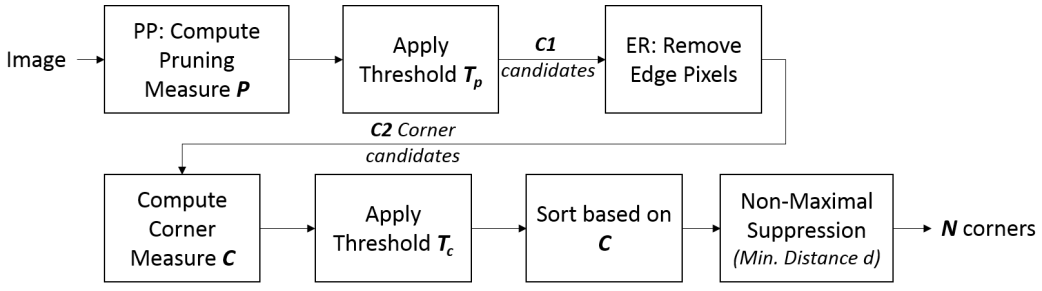

Figure 3.2: Block diagram for the proposed pruning technique for Shi-Tomasi and Harris corner detectors

Partial Pruning

As shown in Fig. 3.2, partial pruning (PP) [2] is the first step of the proposed pruning technique and it selects an initial set of corner candidate pixels $C1$ which have a large value for ac. Applying an appropriate threshold can discard pixels with low ac values.

Figure. 3.3 (b) shows the initial corner candidates selected by applying threshold $= 0.05 * Max(ac)$, and it is clear that this covers the final Shi-Tomasi corner regions in Fig. 3.3 (d) well. Instead of computing a and c for every pixel explicitly, the I_x^2 and I_y^2 terms in the expression for ac are approximated with the absolute values for I_x and I_y respectively and a pruning measure P is computed as follows:

$$P = a'c' = \sum |I_x| \,.\, \sum |I_y| \tag{3.5}$$

This eliminates the multiplication operations involved in the squared gradients. Figure. 3.3 (c) shows that the $a'c'$ map covers the ac map in Fig. 3.3 (b) and the final corner regions in Fig. 3.3 (d) well.

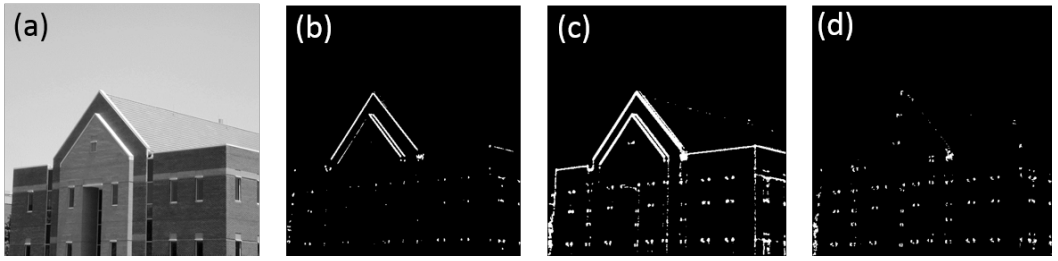

Figure 3.3: Selection of corner candidates (a) Original image (b) Corner candidates selected using ac at $threshold = 0.05 * Max(\mathbf{ac})$ (c) Corner candidates selected using $a'c'$ at $threshold = 0.05 * Max(\mathbf{a'c'})$ (d) Shi-Tomasi corner regions with λ_2 at $threshold = 0.05 * Max(\lambda_\mathbf{2})$

Figure. 3.4 shows that as the threshold for $a'c'$ map is reduced under uniform illumination, corners and slanted edges are released first. This is followed by vertical and horizontal lines, and finally faint textures and flat regions. This shows that when the threshold is applied to $a'c'$ map of the image, non-corner regions are likely to be removed, while retaining a significant amount of corners.

Hence, $a'c'$ can be used as an effective corner indicator measure as it elevates the corner regions above the non-corner regions. At the end of PP, an initial corner candidate set denoted as $C1$ is generated and further processing takes place on $C1$.

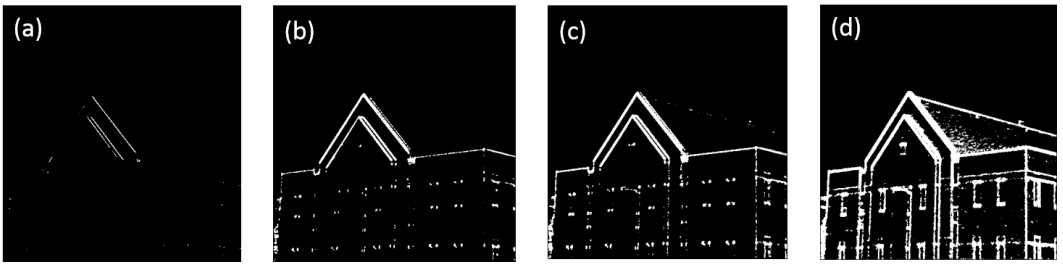

Figure 3.4: Corner candidates with partial pruning PP (denoted as $C1$) by thresholding the $\mathbf{a'c'}$ map at T_p = (a) 0.5 (b) 0.1 (c) 0.05 (d) 0.01

As seen in Fig. 3.4, $C1$ generated after PP contains many edge pixels, which are obvious non-corner pixels. This is because PP only looks for pixels with high values for ac and does not consider b^2, which is the second term in $det(M)$.

Removing Edge Pixels

As shown in Fig. 3.2, the next step is to remove the edge pixels (referred to as ER). In order to maximize $det(M)$, for all the candidates in $C1$, the second term b^2 needs to be substantially smaller than the first term ac. In other words, the pixels that have comparable values for ac and b^2 need to be eliminated. The $det(M)$ computed over a small window W of a pixel is given by:

$$det(M) = ac - b^2 = \sum_{i=1}^{W} I_{x_i}^2 . \sum_{i=1}^{W} I_{y_i}^2 - (\sum_{i=1}^{W} I_{x_i} I_{y_i})^2 \tag{3.6}$$

The gradient direction of the pixel is represented by $k_i = I_{x_i}/I_{y_i}$ and $det(M)$ can be rewritten in terms of the gradient direction k_i as:

$$det(M) = ac - b^2 = \sum_{i=1}^{W} I_{x_i}^2 . \sum_{i=1}^{W} (k_i I_{x_i})^2 - (\sum_{i=1}^{W} k_i I_{x_i}^2)^2 \tag{3.7}$$

When $k_i \approx k$ for all the neighbours of a pixel in window W, the two terms, ac and b^2, converge and $det(M) \approx 0$ as shown:

$$det(M) = ac - b^2 \approx \sum_{i=1}^{W} I_{x_i}^2 . \sum_{i=1}^{W} (kI_{x_i})^2 - (\sum_{i=1}^{W} kI_{x_i}^2)^2 = 0 \tag{3.8}$$

This happens when the pixel and its neighbours fall on an edge with an edge direction given by k. Hence the edge pixels in $C1$ need to be removed, as they do not maximize $det(M)$. A novel compute-efficient method to remove the edge pixels, by analysing the gradients I_x and I_y of the 3x3 neighbours of the candidate pixels, is presented. The proposed ER method avoids an explicit computation of the b^2 term.

The I_x-I_y space is divided into two sets of overlapping bins (bins 1-8) as shown in Fig. 3.5. Edge pixels will have all their neighbours in the same bin whereas corner pixels will show a spread and this allows quick detection of the edge pixels. Overlapping bins are necessary to capture edge pixels located along the bin boundaries in the I_x-I_y space. For instance, bin 8 captures edge pixels falling on the boundary of bins 2 and 3. Also, a 3x3 window is localized around the candidate pixel and hence is considered the best option for the proposed method.

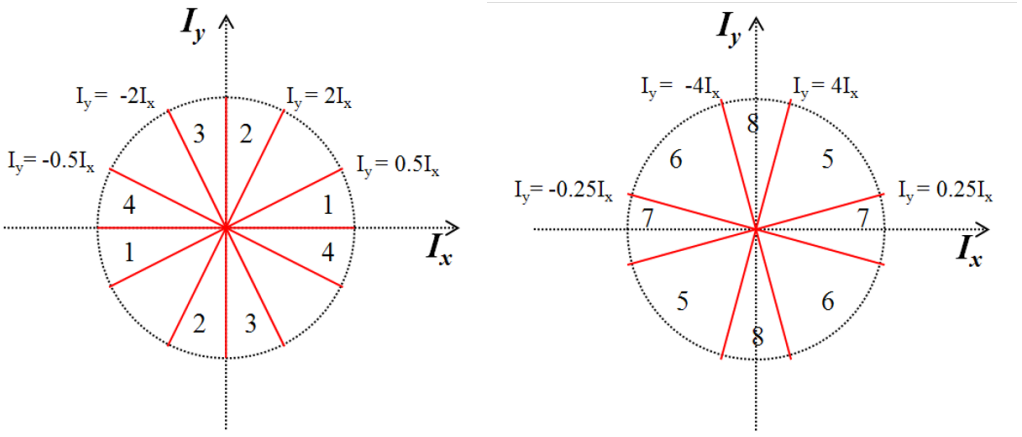

Figure 3.5: Edge removal (ER) in pruning: bin boundaries for I_x-I_y plots

Figure 3.6 illustrates the I_x-I_y plot of the 3x3 patches of various intensity patterns for an example image:

1. Pixels with a relatively flat 3x3 neighbourhood have all the points clustered together close to the origin.

2. Edge point and its neighbours fall along the edge direction.

3. Corner point and its neighbours are spread wider.

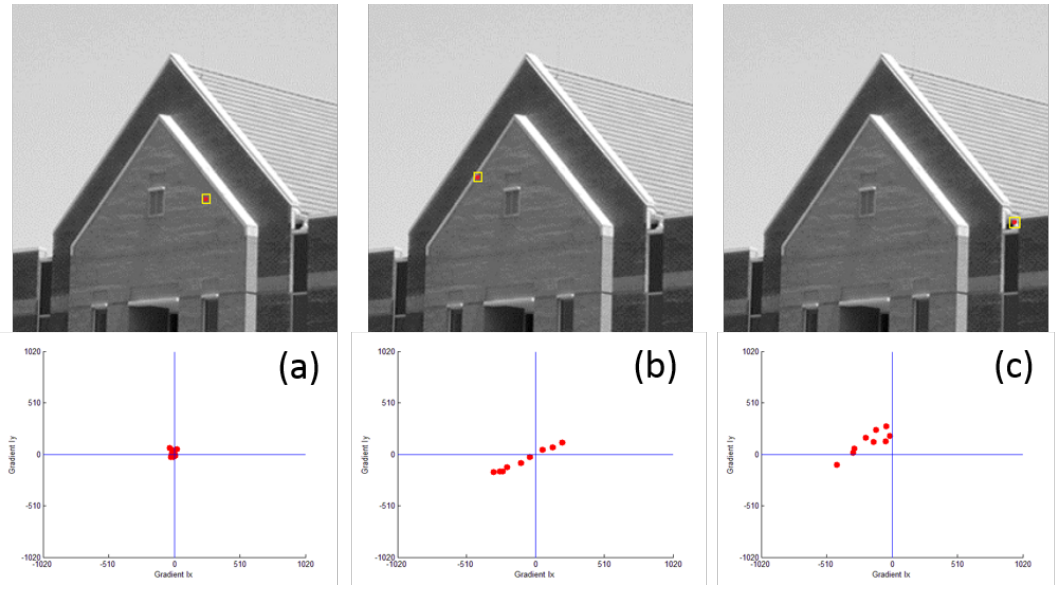

Figure 3.6: Examples of I_x-I_y plots for 3x3 neighbourhood of a pixel in various intensity patterns during the edge removal (ER) step (a) Pixel on a flat region (b) Pixel on an edge (c) Pixel on a corner

In order to identify an edge point, for each candidate pixel in $C1$, the number of neighbour pixels in each bin is counted, in the I_x-I_y space. Ideally, a candidate pixel and all its 3x3 neighbours should fall in the same bin for it to be considered an edge point. However, this is often not the case as, in practice, images are noisy and the gradient computation is a numerical approximation. In order to overcome this problem, a pixel is considered an edge point and is discarded when the number of neighbouring pixels in a particular bin exceeds a predefined threshold (possibly lower than 3x3=9).

During the bin allocation for a candidate pixel in the I_x-I_y space, if a neighbour pixel has a low gradient magnitude, i.e. below the thresholds (C_x, C_y) as computed in Eq. 3.9 and Eq. 3.10, it then contributes equally to all the bins:

$$C_x = mean(\mathbf{I_x}) + k.stdev(\mathbf{I_x}) \tag{3.9}$$

$$C_y = mean(\mathbf{I_y}) + k.stdev(\mathbf{I_y}) \tag{3.10}$$

Here, $\mathbf{I_x}$ and $\mathbf{I_y}$ represent the gradient maps along the x and y directions of the image respectively and $k = 0.25$ is set empirically.

The proposed method for removing edge pixels from the initial corner candidate set $C1$ relies on the extremely compute-efficient bin allocation strategy that is enabled by the use of bin boundaries, which are factors of 2 of the gradient magnitudes. This enables the bin boundary computation and bin allocation to be achieved using only comparisons and bit shifts. The corner candidate set generated at the end of ER is referred to as $C2$ and the baseline algorithm (Shi-Tomasi or Harris) is now applied only on $C2$.

As shown in Fig. 3.2, the corner measure computation, sorting and non-maximal suppression steps are applied only on the corner candidates remaining after the PP-ER pruning has been completed.

Nomenclature: The proposed pruning technique can be applied to both Shi-Tomasi (ST) and Harris (H) corner detectors. As the proposed pruning technique has two steps - PP that stands for *partial pruning* [2] followed by ER that stands for *edge removal* [1], it is referred to as PP-ER. To refer to the corresponding corner detectors with pruning, suffix of ST or H is added, leading to PP-ER_{ST} and PP-ER_{H} for the proposed pruning method for Shi-Tomasi and Harris respectively. For the cost analyses and evaluations, pruning with only PP is also considered (in order to demonstrate the impact of ER) and this is referred to as PP_{ST} and PP_{H} for partial pruning for Shi-Tomasi and Harris respectively. As shown in Eq. 3.1, when the averaging filter $w(x)$ is a box filter, the suffices for the corner detection algorithms are STB and HB for Shi-Tomasi (Box) and Harris (Box) respectively. When the averaging filter is a Gaussian filter, the corresponding suffices are STG and HG.

Listing 1 Pruning based corner detection (PP-ER$_{ST/H}$)

for Each Pixel in Image I **do** ▷ Partial Pruning: PP
    Compute $P = a'c' = \sum |I_x| . \sum |I_y|$
    Apply threshold $= T_p * Max(a'c')$ to the $a'c'$ map to get $C1$
end for
for Each Pixel in $C1$ **do** ▷ Edge Removal: ER
    Compute bin boundaries: $0.25I_x$, $0.5I_x$, $2I_x$, $4I_x$
    Allocate bins for each of the 3x3 neighbours in the I_x-I_y plot
    for Each Bin **do**
        if $nBinMembers > binThreshold$ **then**
            $DiscardFlag = True$
        end if
    end for
    if $DiscardFlag = false$ **then**
        Add pixel to the candidate set $C2$
    end if
end for
for Each pixel in $C2N$ ($C2$ and its neighbours) **do** ▷ Corner Response: B1a
    Compute I_x^2, I_y^2, I_xI_y
end for
for Each Pixel in $C2$ **do** ▷ Corner Response: B1b
    Compute $a = \sum w(x)I_x^2$, $c = \sum w(x)I_y^2$, $b = \sum w(x)I_xI_y$
end for
for Each Pixel in $C2$ **do** ▷ Corner Response: B2
    Compute corner response $C = R$ or λ_2
end for
Apply threshold $= T_c * Max(C)$ on pixels in $C2$ ▷ Corner Response: B3
Sort the corner candidates in $C2$ in descending order of C
Apply non-maximal suppression

Cost Analysis

Listing 1 describes the proposed low-complexity pruning technique applied to the baseline corner detector. The proposed PP-ER$_{ST/H}$ algorithm first applies partial pruning (PP) to generate a corner candidate set $C1$. It then removes edge pixels from $C1$ using ER to generate $C2$. The baseline algorithm is then applied only to the corner candidate set $C2$. It is noteworthy that PP is applied to the entire image, but subsequent steps of the pruning and baseline algorithm are only applied to smaller candidate sets (i.e. $C1$ and $C2$).

Steps	Variants	Multiplications	Additions	Square Root
PP		1 (16-bit)	2W (16-bit) + 2	
ER		1 (32-bit)	39 (16-bit)	
B1a: I_x^2, I_y^2, I_xI_y		3 (16-bit)		
B1b: $w(x)$ for M	Box		3W (32-bit)	
	Gaussian	3W (FP)	3W (FP)	
B2: C	Shi-Tomasi (Box)	2 (32-bit)	4 (32-bit)	1
	Shi-Tomasi (Gaussian)	2 (FP)	4 (FP)	1
	Harris (Box)	3(32-bit) + 1 (FP)	2 (32-bit) + 1 (FP)	
	Harris (Gaussian)	4 (FP)	3 (FP)	

Table 3.1: Operations per pixel for pruning and conventional Shi-Tomasi and Harris corner detectors

The Shi-Tomasi and Harris algorithms have been used as the baseline algorithms. They consist of the following key steps:

- $B1$: Compute auto-correlation matrix M for each pixel,
- $B2$: Compute corner measure (Shi-Tomasi or Harris) C for each pixel, and
- $B3$: Sort all the pixels based on the corner measure and apply non-maximal suppression to generate the final corners.

Table 3.1 shows the computations incurred by the pruning and the baseline algorithms per pixel. The pruning step PP only has 16-bit additions and multiplications and ER only has bit shifts and comparisons (which are reported as additions). It is evident that the computations of PP and ER are less complex compared to the steps $B1$ and $B2$ of the baseline algorithms, thereby potentially resulting in substantial savings.

Figure 3.7 shows the number of pixels on which each step of the PP-ER$_{\mathrm{ST/H}}$ is applied. The first step in pruning, PP results in a corner candidate set, $C1$. The number of candidate pixels in this set is represented as $|C1|$, and the second step in pruning, ER is applied only on $|C1|$ pixels. This results in a smaller candidate set $C2$. It should be noted that for the next step $B1a$, when the baseline algorithm is used without pruning, for each pixel, 3 (16-bit) multiplications are incurred for $B1a$ (computing the I_x^2, I_y^2, I_xI_y). When pruning is applied, these values need to be computed for the corner candidate as well as its neighbours in the window W.

Hence $B1a$ needs to be applied to all pixels in $C2$ and their neighbours (represented as $C2N$) for PP-ER. The steps $B1b$ to compute auto-correlation matrix M and $B2$ to compute the corner measure C are applied to the pixels in the candidate set $C2$. In the case of PP, $B1a$ is applied to all pixels in the candidate set $C1$ and their neighbours (represented by $C1N$) and the subsequent steps $B1b$ and $B2$ are applied to all pixels in candidate set $C1$.

Figure 3.7: Corner candidate sizes with PP-ER algorithm

The computations of PP-ER$_{\mathrm{ST/H}}$ is compared with the baseline algorithms (i.e. Shi-Tomasi and Harris), PP$_{\mathrm{ST/H}}$, and recently reported work that aim to reduce complexity of the baseline algorithms [68, 71] in Table 3.2 and 3.3. In these comparisons, it was assumed that a box filter i.e. $w(x) = 1$ and a 3x3 window (i.e. $W = 9$) is used for the auto-correlation matrix M. When a Gaussian window is used for $w(x)$, additional floating point (FP) multiplications are incurred on the corner measure computation step $B1b$. When compared to the box filter, the use of a Gaussian window will clearly lead to higher savings for PP-ER$_{\mathrm{ST/H}}$ as the overall number of multiplications increases significantly in the baseline algorithms. Note that, in both [68] and [71], the complexity of the baseline algorithm is reduced by removing the square root in Shi-Tomasi and reducing the number of additions for the window operations respectively. However, the computations are still applied to all pixels in the image. In contrast, the proposed pruning approach can potentially result in significant savings as the values for $|C1|$, $|C2|$, $|C1N|$ and $|C2N|$, which represent the size of the corner candidates at various stages in the algorithm (as shown in Fig. 3.7), are typically substantially smaller than the image size. In order to illustrate the savings in computations per pixel, the corner candidate sizes for $C1$, $C2$, $C1N$ and $C2N$ have been normalized with the image size and are represented as $pC1$, $pC2$, $pC1N$ and $pC2N$ in Tables 3.2 and 3.3.

Method	Multiplications	Additions	Square Roots
Shi Tomasi [50]	5	31	1
Perona [68]	5	30	
SLC-KLT [71]	5	19	1
PP_{ST} [2]	$1 + pC1N * 3 + pC1 * 2$	$20 + pC1 * 31$	$pC1$
PP-ER_{ST} [1]	$1 + pC1 + pC2N * 3 + pC2 * 2$	$20 + pC1 * 39 + pC2 * 31$	$pC2$

Table 3.2: Comparison of computations for each pixel for PP-ER_{ST}

Method	Multiplications	Additions
Harris [51]	6	30
SLC-Harris [71]	6	18
PP_H [2]	$1 + pC1N * 3 + pC1 * 4$	$20 + pC1 * 30$
PP-ER_H [1]	$1 + pC1 + pC2N * 3 + pC2 * 4$	$20 + pC1 * 39 + pC2 * 30$

Table 3.3: Comparison of computations for each pixel for PP-ER_H

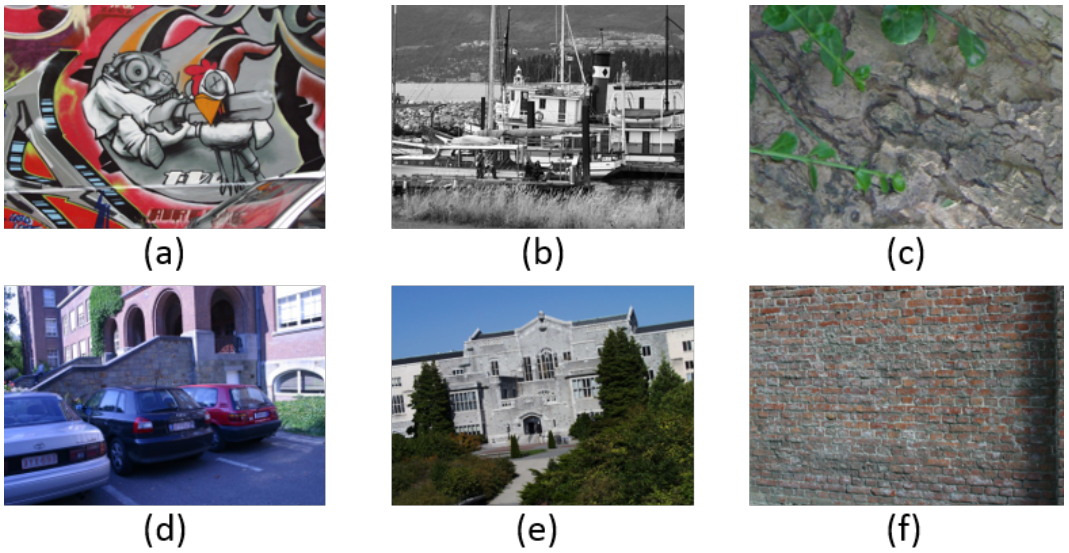

Figure 3.8: Image dataset [119] used for the evaluation (a) *graf* (b) *boat* (c) *bark* (d) *leuven* (e) *ubc* (f) *wall*

The experiments with the chosen image dataset in Fig. 3.8 show that the average values for the normalized corner candidate sizes of 300 corners for Shi-Tomasi and Harris are $pC1 = 0.083$, $pC2 = 0.052$, $pC1N = 0.173$ and $pC2N = 0.134$. The number of operations per pixel can be estimated by substituting these values of $pC1$, $pC2$, $pC1N$ and $pC2N$ in Tables 3.2 and 3.3. For example, the total number of multiplications for PP_{ST} in Table 3.2 is $1 + pC1N * 3 + pC1 * 2 \approx 1.6$. This can be compared to the corresponding baseline algorithm, Shi-Tomasi that needs 5 multiplications per pixel.

Method	Loads	Stores
Baseline Shi-Tomasi and Harris	$4W + 2$	5
PP-ER$_{\mathrm{ST/H}}$ [1]	$3W + pC1 * W + pC2N * 2 + pC2 * 3W$	$2 + pC1 + pC2N * 3 + pC2$

Table 3.4: Comparison of memory load/store for each pixel for PP-ER$_{\mathrm{ST/H}}$

Table 3.4 shows the comparison of the total number of estimated memory accesses per pixel. The number of memory accesses per pixel for pruning can be estimated by substituting these values of $pC1$, $pC2$, $pC1N$ and $pC2N$ in Table 3.4 as follows: total number of loads per pixel $\approx 3.24W + 0.26$ and stores per pixel ≈ 2.54. Therefore, compared to the baseline algorithms, there is a reduction in the number of memory accesses per pixel.

Tables 3.1, 3.2 and 3.3 show that the per-pixel complexity of the operations in the proposed algorithms is lower than the baseline algorithms. For example, the PP and ER computations only require 16-bit operations whereas the baseline corner measure computations require 32-bit and floating-point operations. The actual realization of these operations depends on the target processor. For example, if the target processor does not support a floating-point unit, multiple integer operations will be used to emulate the floating-point operations in the conventional corner measure computation. The floating point emulations will incur a high computational complexity. Similarly, processors that support only 16-bit memory transfers will result in a higher number of memory load/store operations for the conventional corner measure computation.

Performance Evaluations

In this section, the performance of the proposed PP-ER$_{\mathrm{ST/H}}$ methods is evaluated from the standpoint of accuracy and efficiency, when compared with the corresponding baseline algorithms. As the pruning technique is tightly correlated with the corner detection algorithms, it is expected that the accuracy of the pruning-based corner detection will be comparable to using the conventional corner detectors. At the same time, as the pruning measure is of much lower complexity than

the corner measures in Eq. 3.2 and Eq. 3.3, the efficiency of the proposed pruning based corner detectors is expected to be substantially improved.

Evaluation Setup

The proposed PP-ER$_{\mathrm{ST/H}}$ technique is compared with the following algorithms:

1. *Conventional corner detectors without pruning, Shi-Tomasi (ST) and Harris (H)*

   Two variants of each of the baseline algorithms have been used, which is based on the weights $w(x)$ of the filter for the autocorrelation matrix M in Eq. 3.1:

   - Simple averaging filter (also called Box) resulting in Shi-Tomasi-Box (STB) and Harris-Box (HB), and
   - Gaussian filter resulting in Shi-Tomasi-Gaussian (STG) and Harris-Gaussian (HG).

   The window size is set to W=3x3 and $\sigma = 0.5$ for the Gaussian filter, as in [117]. A normalized 3x3 discrete Gaussian kernel is used as shown below:

$$w(x) = \begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix} \tag{3.11}$$

   The following 3x3 Sobel filters are used for computing the horizontal and vertical gradient images I_x and I_y as shown below:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.12}$$

   The threshold for cornerness is set to $T_c = 0.05$ for Shi-Tomasi and $T_c = 0.005$ for Harris.

2. *Corner detection with partial pruning ($PP_{ST/H}$)*

   Partial pruning is used as a baseline to show the impact of edge response removal (ER) step in the proposed PP-$\text{ER}_{\text{ST/H}}$ technique. Threshold for the pruning measure in Eq. 3.5 is set to $T_p = 0.05$.

3. *Corner detection with cascaded candidate pruning ($CCP_{ST/H}$)*

   The cascaded candidate pruning (CCP) [72] selects corner candidates that are local maxima in gradient magnitude. It does not consider the gradient patterns in the pixel neighbourhood.

All the baseline algorithms mentioned above, and the proposed pruning techniques $\text{PP}_{\text{ST/H}}$ and PP-$\text{ER}_{\text{ST/H}}$, have been implemented in C, following the description of the algorithms in Section 3.2 and 3.3, and the detailed pseudocode in Listing 1.

The image data in Fig. 3.8 is used to evaluate the accuracy of the corner detectors. The dataset contains $n_S = 6$ sequences (*graf*, *boat*, *bark*, *leuven*, *ubc*, *wall*) of images with various image transformations such as changes in viewpoint, zoom, rotation and illumination [119], with each sequence containing $n_I = 6$ transformed images. The image resolutions are as follows: *graf*-800x640, *boat*-850x680, *bark*-765x512, *leuven*-900x600, *ubc*-800x640 and *wall*-880x680. A feature set of 300 corners was generated for all the algorithms considered.

Accuracy Evaluation

The following evaluation criteria for accuracy are used for the evaluation of the proposed methods ($\text{PP}_{\text{ST/H}}$ and PP-$\text{ER}_{\text{ST/H}}$):

1. *Feature matches*: The feature sets of the pruning based methods, $S_{pruning}$ and the baseline algorithms $S_{baseline}$ are compared. A feature C_i is considered a match if $m_i = 1$ as follows:

$$m_i = \begin{cases} 1 & c_i \in S_{pruning}, c_i \in S_{baseline} \\ 0 & Otherwise \end{cases} \tag{3.13}$$

In other words, if a corner found by the pruning based method is also found by the baseline algorithm, then it is considered a feature match. The feature matches for a given set of features, are counted and reported as a percentage of the total number of features N, as μ:

$$\mu = \frac{\sum_{i=1}^{N} m_i}{N} \tag{3.14}$$

For example, if 270 corners from the pruning based method matched with the corresponding baseline algorithm, out of the total of 300 corners detected, a match of $\mu = 90\%$ is reported. For each image pair (i, j) in the image sequence (i^{th} sequence, j^{th} transformed image), the feature match $\mu^{i,j}$ is computed. The range of values of the feature matches across all the image pairs is reported as μ_{min} and μ_{max}.

2. *Repeatability rate*: It is desirable for each corner to be *repeatable*, i.e. be detected in two or more images of the same scene. Such features can be reliably used for registering the images. Repeatability rate is defined as the percentage of features that are simultaneously present in two images and has been extensively used as a metric for corner detectors [58]. The two images represent the same scene but with some variations such as a viewpoint change as in [119].

   For each image pair (i, j) in the sequence (i^{th} sequence, j^{th} transformed image), the repeatability rate $\rho^{i,j}_{baseline}$, of the baseline Shi-Tomasi/Harris corner detectors is first computed. Then the repeatability rate $\rho^{i,j}_{proposed}$, of the pruning-based corner detectors is computed. It is expected that when pruning is applied, there is no loss in accuracy of the corner detection. Therefore, the difference in repeatability rate $\Delta\rho^{i,j}$ and its mean $\overline{\Delta\rho}$ are computed as follows:

$$\begin{aligned} \Delta\rho^{i,j} &= \rho^{i,j}_{proposed} - \rho^{i,j}_{baseline} \\ \overline{\Delta\rho} &= \frac{\sum_{i=1}^{n_S} \sum_{j=1}^{n_I} (\Delta\rho^{i,j})}{n_S.n_I} \end{aligned} \tag{3.15}$$

Proposed Methods	Feature Matches μ		Repeatability Rate		
	μ_{min} (%)	μ_{max}(%)	$\Delta\rho_{min}$	$\Delta\rho_{max}$	$\overline{\Delta\rho}$
CCP_{STB}	3.7	16.0	-28.91	1.04	-12.41
CCP_{STG}	2.7	20.3	-36.84	5.18	-9.66
CCP_{HB}	4.7	20.7	-30.46	0.01	-13.40
CCP_{HG}	10.7	38.0	-48.16	-0.51	-15.55
PP_{STB}	99.7	100.0	0.00	0.33	0.01
PP_{STG}	95.7	100.0	0.00	0.00	0.00
PP_{HB}	100.0	100.0	-0.33	0.00	-0.01
PP_{HG}	99.0	100.0	0.00	0.00	0.00
$PP\text{-}ER_{STB}$	98.0	100.0	-1.84	0.33	-0.13
$PP\text{-}ER_{STG}$	95.0	100.0	-3.07	0.51	-0.17
$PP\text{-}ER_{HB}$	96.7	100.0	-0.67	0.74	0.02
$PP\text{-}ER_{HG}$	98.0	100.0	-3.57	1.26	-0.10

Table 3.5: Accuracy results for PP and PP-ER in comparison to the baseline algorithms

When $\Delta\rho$ is negative it implies that the repeatability of the proposed method was lower than the baseline method indicating a loss in accuracy due to pruning, and vice versa. The range ($\Delta\rho_{min}$, $\Delta\rho_{max}$) along with its mean $\overline{\Delta\rho}$ is reported over all image sequences for each method.

In Table 3.5, the accuracy evaluation results are reported. In general, the cascaded candidate pruning (CCP) method proposed in [72] performs poorly, showing a substantial drop in repeatability rate of 29% and above, as well as substantially low feature matches at less than 38%, compared to the baseline corner detectors. This is attributed to how the candidate pixels are selected by CCP: pixels that are local gradient magnitude maxima are chosen, without considering the intensity or gradient patterns in the pixel neighbourhood. However, the Shi-Tomasi and Harris corner detectors rely on the variability of the gradient magnitudes in the pixel neighbourhood, which is quantified as the corner metric as defined in Eq. 3.2 and Eq. 3.3 respectively. Therefore, potential corner pixels may not necessarily be local maxima in gradient values and CCP eliminates many of these potential corners. In addition, CCP also selects noisy pixels as candidates, which also have high gradient values, leading to a lower quality corner set and therefore poor rate of repeatability.

The proposed pruning techniques, $\text{PP}_{\text{ST/H}}$ and $\text{PP-ER}_{\text{ST/H}}$, consider the pixel neighbourhood in both the PP and ER stages of pruning, achieving repeatability rates very close to their corresponding baselines (with the worst case drop being only 4%) and feature matches of 95% and above. Therefore the proposed pruning method is able to approximate the more complex Shi-Tomasi and Harris corner measure with a high degree of accuracy.

Efficiency Evaluation

Embedded platform: The computational efficiency achieved by employing the pruning techniques $\text{PP}_{\text{ST/H}}$ and $\text{PP-ER}_{\text{ST/H}}$ is demonstrated on the Nios-II embedded platform [120]. The Nios-II soft core processor is chosen for the evaluations as the on-chip configurations of cache and floating point unit (FPU) can be modified for this processor. This allows the analysis of the impact of FPU and cache on the performance of the proposed pruning based corner detection. Further details of the experimental setup with Nios-II and the software development flows have been provided in Appendix B. As FPU often leads to high cost and high energy consumption in embedded processors, two configurations of the Nios-II soft core are used: with FPU disabled and FPU enabled.

Evaluation criteria: The execution time is measured for $\text{PP}_{\text{ST/H}}$ and $\text{PP-ER}_{\text{ST/H}}$ pruning based corner detection (referred to as $t_{proposed}$), as well as their corresponding baseline corner detectors without pruning (referred to as $t_{baseline}$). To show the overall savings in computations, achieved by introducing pruning, the *relative speedup* ψ (%) in execution time of the proposed pruning techniques with respect to the corresponding baseline algorithms is reported, as given by:

$$\psi = \frac{(t_{baseline} - t_{proposed})}{t_{baseline}} \tag{3.16}$$

The images in Fig. 3.8 have varying sizes and therefore execution time needs to be normalized for comparison. This is achieved by the use of the *speedup* ψ in

execution time as the evaluation metric. Note that, the first image in each image sequence in Fig. 3.8 is used for the efficiency evaluations.

Figure 3.9 (a) shows the execution times (in seconds) as the bar charts and speedup ψ (%) as triangles/circles charts for Shi-Tomasi on Nios-II when FPU is disabled. The images have been arranged from left to right from smallest to the largest in terms of their image size. The execution time of the baseline algorithms ($t_{baseline}$ darker blue/orange) i.e., *STB* and *STG* depends on both the image size and the nature of the image content. Therefore, images of larger size have higher execution times - for example, *wall* has a higher execution time than *bark*.

The overall execution time for *STB* is in the range of (3.3-6.5) seconds. The execution time for *STG* is in the range of (11.6-21.5) seconds due to the more complex corner measure involving the Gaussian filter. The average speedup achieved by PP_{STB} is 38% and 76% for PP_{STG}. It is to be noted that the execution time of PP_{ST} for both the Box and Gaussian is almost the same. This shows how effective the pruning has been in discarding most of the non-corner regions resulting in very small corner candidate sets.

The nature of the image content determines the amount of savings that can be achieved. For images that have a combination of textures and homogeneous regions - such as *boat* - the savings are higher with pruning. In comparison, images with a comparable image size but with very rich textures and less homogeneous regions - such as *wall* - the benefits from PP_{ST} are lower. This can be attributed to the chosen threshold for T_p and T_c - while this threshold is optimal for the image *boat* releasing only the necessary candidates for selecting $N = 300$ corners, it is too low for the richer image *wall*, releasing many more candidates than required.

Figure 3.9 (b) shows the additional speedup that is achieved with PP-ER_{ST} when compared to PP_{ST}. It is evident that all the images benefit from the removal of edge pixels resulting in much smaller corner candidate sets that need the complex corner measure computation. *graf* has very distinct edges that PP_{ST} was unable to prune and therefore PP-ER_{ST} shows clear benefits. Similarly, *wall* does not benefit from PP_{ST}, but shows positive speedup with PP-ER_{ST}.

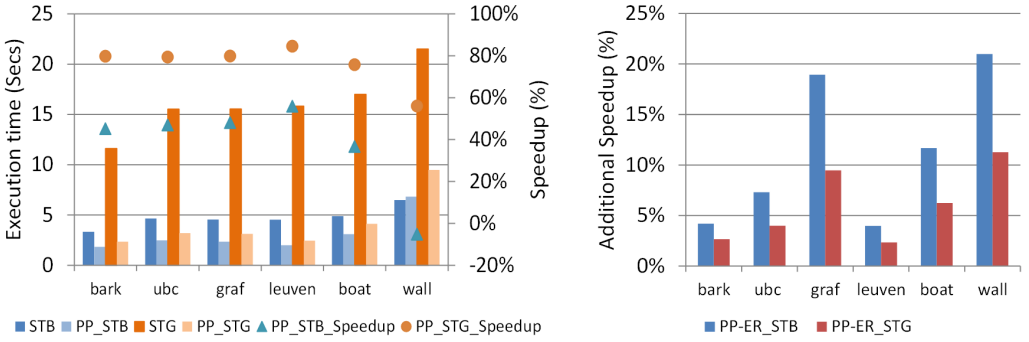

Figure 3.9: (a) Speedup in execution time on Nios-II with FPU disabled for Shi-Tomasi with PP_{ST} (b) Additional speedup achieved with PP-ER_{ST}

Figure 3.10 shows the corresponding timing results for Harris. The execution times of HB are in the range of (3-6) seconds. In comparison to STB, the corner measure of HB is less complex and hence, the execution times are lower. The average speedup achieved by PP_{HB} and PP_{HG} is 35% and 76% respectively. PP-ER_{H} achieves higher average speedup of 46% and 82% for PP-ER_{HB} and PP-ER_{HG} respectively.

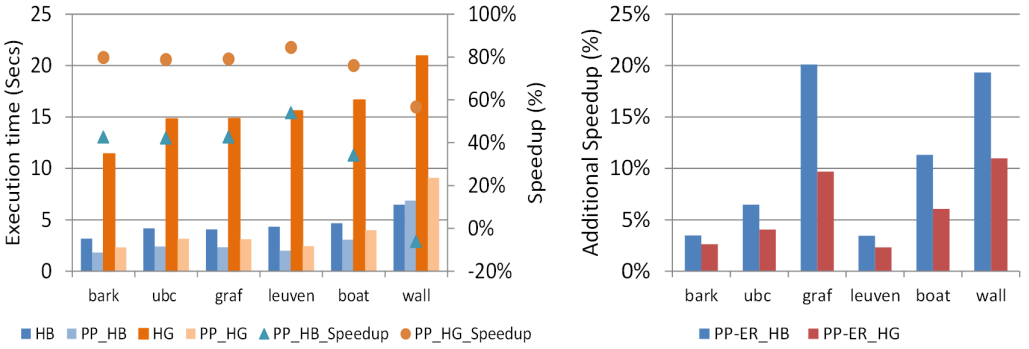

Figure 3.10: (a) Speedup in execution time on Nios-II with FPU disabled for Harris with PP_{H} (b) Additional speedup achieved with PP-ER_{H}

Figure 3.11 shows the timing results with FPU enabled on Nios-II soft core. When the FPU is available the floating point operations are handled by the FPU and hence the corner measure computation is executed much faster compared to the software emulation of floating-point arithmetic in the absence of an FPU. However, an average speedup with PP_{STB} of 43%, PP_{STG} of 69%, PP_{HB} of 8% and PP_{HG} of 64% is still achieved. PP-$ER_{ST/H}$ still shows significant improvements when

compared to $\text{PP}_{\text{ST/H}}$, for *graf* and *wall* images, which have a large number of edge pixels, that are only pruned away by the ER step.

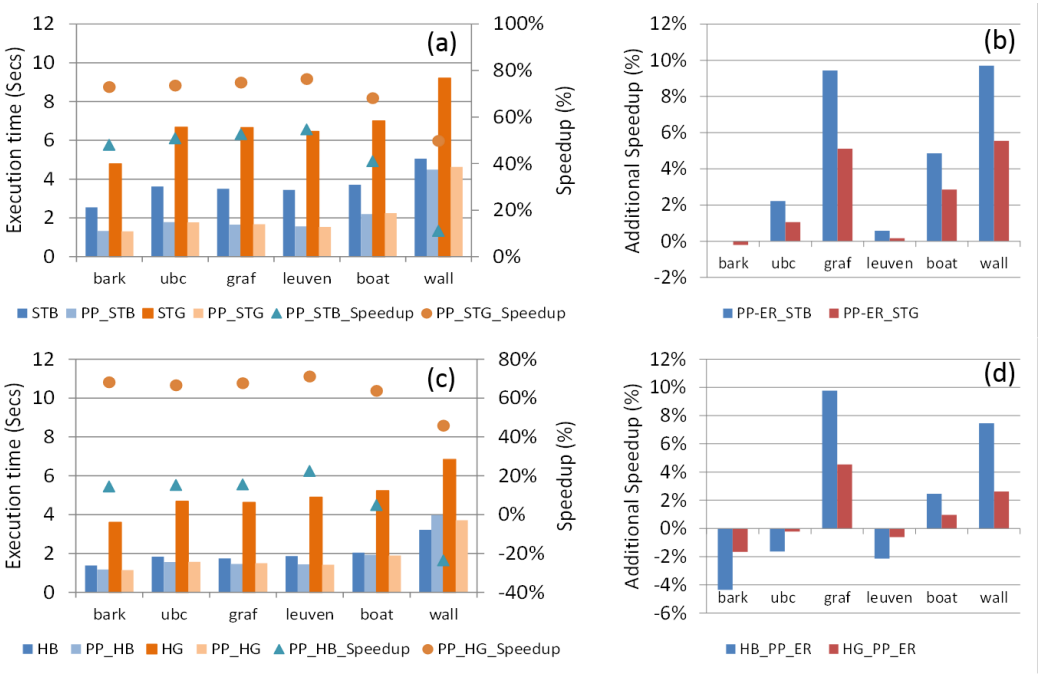

Figure 3.11: On Nios-II with FPU enabled (a) Speedup in execution time for Shi-Tomasi with PP_{ST} (b) Additional speedup achieved with PP-ER_{ST} (c) Speedup in execution time for Harris with PP_{H} (d) Additional speedup achieved with PP-ER_{H}

Impact of on-chip data cache: In order to analyse the impact of cache on the proposed method, the baseline and proposed algorithms were implemented on two configurations of the Nios-II processor: with and without cache (FPU was disabled in both cases). The speedup achieved by the $\text{PP-ER}_{\text{ST/H}}$ is reported in Fig. 3.12. When the cache is disabled, the memory access time becomes the major component of the total execution time, in comparison to the computation time. However, even in this case, a substantial speedup of 33% for $\text{PP-ER}_{\text{STB}}$ and 37% for PP-ER_{HB} is seen.

Selection of threshold T_c: As seen in the evaluation results, the speedup achieved with pruning is dependent on the image content. The same threshold of T_c and T_p was applied for all images, but depending on the content - presence of textures, edges and their contrast - the speedup due to pruning varied.

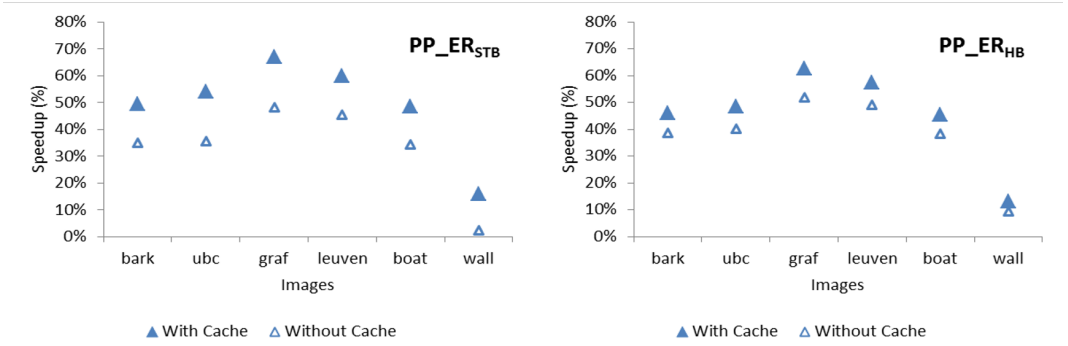

Figure 3.12: Efficiency evaluation with/without cache: speedup in execution time (a) Shi-Tomasi with PP-ER$_{\text{STB}}$ (b) Harris with PP-ER$_{\text{HB}}$

The impact of varying the threshold on the corner detection and the speedup achieved with the proposed pruning methods is analysed. Figure 3.13 shows the number of corners released when the threshold T_c is varied for the baseline Shi-Tomasi and Harris algorithms and it can be clearly seen that the number of corners released at a given threshold depends on the image content.

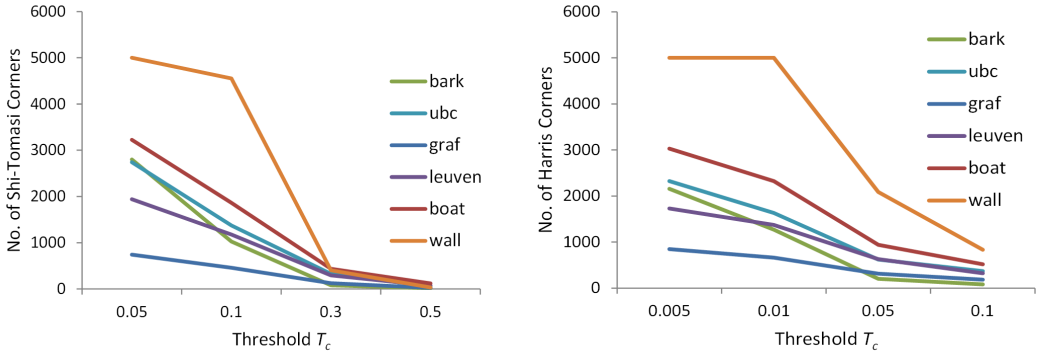

Figure 3.13: Number of corners vs. threshold T_c in baseline (a) Shi-Tomasi and (b) Harris algorithms

Figure 3.13 (b) shows that Harris requires much lower threshold values for T_c to select comparable number of corners with Shi-Tomasi. This is because Shi-Tomasi and Harris corner measure values have different ranges. Harris corner measure is directly proportional to the determinant of the autocorrelation matrix, $det(M)$ as in Eq. 3.3. Shi-Tomasi corner measure is proportional to the root of the determinant of the autocorrelation matrix $det(M)$ as shown in Eq. 3.2.

Figure 3.14 shows the impact of varying the threshold T_p ($T_c = 0.05$ for Shi-Tomasi and $T_c = 0.005$ for Harris to ensure the required number of $N = 300$ corners is found). A high value of T_p results in insufficient number of corners whereas a low value of T_p reduces the overall speedup achieved by pruning. The investigations show that when threshold T_p is set such that 300 corners is found, the repeatability ρ and feature matches μ do not vary much with the threshold and is similar to the results reported in Table 3.5.

In addition, in Fig 3.14, the benefit of PP-ER when compared to PP can be clearly seen when the thresholds are set lower. At lower thresholds such as when $T_p = 0.01$, partial pruning (PP) alone is unable to prune away many non-corner candidates and incurs an overhead in computations as seen in images *bark*, *ubc*, *boat* and *wall*. However due to the efficient pruning by the ER step, PP-ER achieves a speedup of an average of 59% for STB and 56% for HB at $T_p = 0.01$. This shows that PP-ER is a highly efficient pruning technique and the ER step is critical for higher computation savings.

Global Motion Estimation on Aerial Videos

As the main objective of reducing the complexity of the corner detection is to enable low-complexity global motion estimation, the proposed pruning techniques $\text{PP}_{\text{ST/HB}}$ and $\text{PP-ER}_{\text{ST/HB}}$ are employed in a feature-based GME pipeline as described in Fig. 2.6, on the VIRAT [121] aerial video dataset with 150 frames as shown in Fig. 3.15.

The GME is performed on every frame pair in the chosen video sequence. For every frame, a maximum of $N = 300$ corners is detected using the box variants of the baseline algorithms (STB and HB) and their corresponding proposed pruning based methods. The thresholds are set as follows: Shi-Tomasi Box ($T_c = 0.01$, $T_p = 0.01$) and Harris Box ($T_c = 0.001$, $T_p = 0.01$). Then, the Kanade-Lucas-Tomasi (KLT) feature tracking [54] (as in Section 2.3.2) is used to find the feature correspondences in the next frame. The Random Sample Consensus (RanSaC)

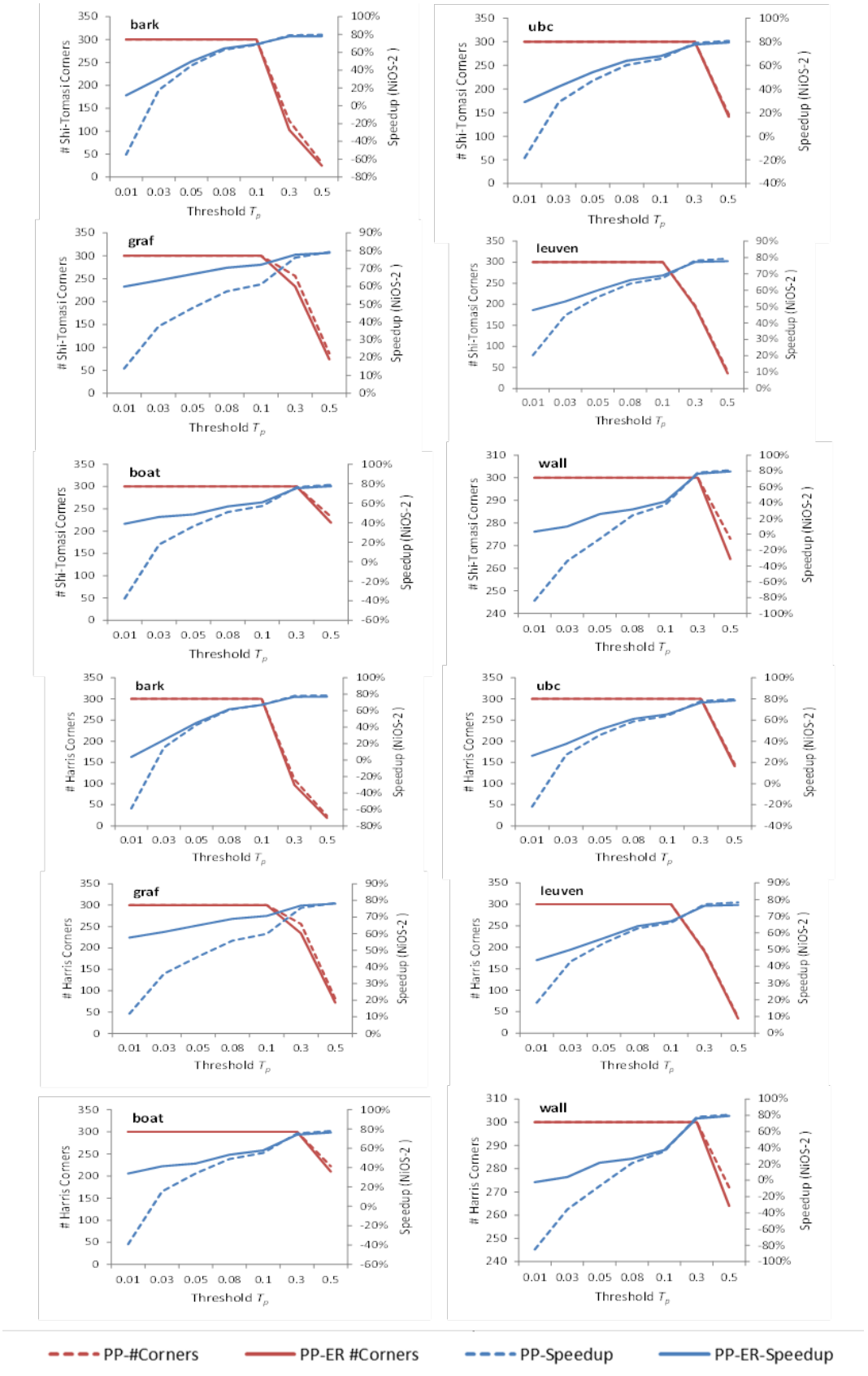

Figure 3.14: Number of corners and speedup in execution time vs. pruning threshold T_p on PP_{STB}, $\text{PP-ER}_{\text{STB}}$, PP_{HB} and PP-ER_{HB}

Figure 3.15: Sample frames from VIRAT [121] video dataset: evaluation of GME with pruning-based corner detection

algorithm [96] is then applied to the feature correspondences with a 2D affine motion model as in Eq. 2.3, for the GME.

The pruning-based corner detection used the C implementation in Section 3.5. The rest of the modules in feature-based GME, were implemented using standard libraries, in OpenCV, for the KLT feature tracker and RanSaC algorithm. The corner detection was executed on the Nios-II platform (as in Appendix B with the floating point unit enabled) and the detected corners were then fed into the GME pipeline, executed on a standard desktop.

For a frame pair (F_1, F_2), the parameters of the affine model T are estimated by the GME. This is used to align the frame F_1 with respect to frame F_2 by eliminating the global motion, resulting in an estimated frame $F_2' = T(F_1)$. The peak-signal-to-noise-ratio (PSNR) is computed for the frame difference between the actual frame F_2 and the estimated frame F_2' in order to measure the accuracy of the GME as shown below:

$$PSNR = 10 \times \log_{10}\left(\frac{255^2}{MSE}\right) \tag{3.17}$$

MSE is the mean of the squared-error between the two frames F_2 and F_2'. The relative error in PSNR when pruning based corner detection replaces the conventional corner detectors in the GME is represented as the error margin $\Delta PSNR$ computed as:

$$\Delta PSNR = \frac{PSNR_{proposed} - PSNR_{baseline}}{PSNR_{baseline}} \tag{3.18}$$

	Δ *PSNR*			
	PP_{STB}	PP_{HB}	PP-ER_{STB}	PP-ER_{HB}
Min	-0.3%	0.0%	-0.4%	-0.4%
Max	0.1%	0.0%	0.2%	0.2%

Table 3.6: Error margin in GME accuracy with pruning based corner detection

Table 3.6 shows the error margin in PSNR for the video frames when the $PP_{ST/H}$ and PP-$ER_{ST/H}$ are used in comparison to the corresponding baseline algorithms. The error margin is within 0.5% and shows that the pruning based corner detection achieves comparable accuracy in GME with the baseline corner detection methods.

As the image content does not change substantially in every frame, in order to demonstrate the computation savings with pruning, a key frame is chosen to represent a batch of 15 consecutive frames. Figure 3.16 shows that a speedup in execution time ψ (as in Eq. 3.16) of 50% with Shi-Tomasi and 20% with Harris corner detection is achieved when the proposed PP-$ER_{ST/H}$ pruning technique is employed.

As these frames contain many edges that need to be removed after $PP_{ST/H}$, PP-$ER_{ST/H}$ shows higher savings in computation time for both STB and HB. For the Gaussian variants of the baseline algorithms, even higher savings are expected in computations than the box variants. These results show how pruning lowers the per-frame computation complexity of the corner detection enabling GME on resource constrained embedded platforms.

Summary

In this chapter, a low complexity pruning technique [1, 2] is presented to accelerate corner detection with Shi-Tomasi and Harris algorithms. The proposed technique systematically selects corner candidates for Shi-Tomasi and Harris corner detectors by efficiently pruning away non-corner regions. Unlike existing pruning techniques

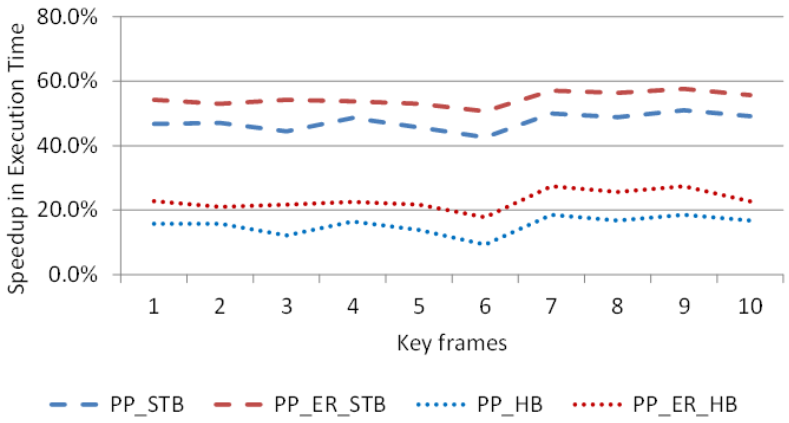

Figure 3.16: Speedup in execution time (ψ) on Nios-II for pruning based corner detection in GME

that operate only on the individual pixels, the proposed pruning steps closely approximate the corner measure of the Shi-Tomasi and Harris algorithms, by considering the gradient patterns in the neighbourhood of the pixel. This results in minimal loss of the potential corner candidates during pruning. In addition, the pruning also results in small candidate sets and the complex corner measure is applied only to this small pool of candidates. Finally, the pruning measure itself involves only simple computations compared to the more computationally complex corner measure, thereby leading to significant computation savings.

Evaluations on an embedded platform (Nios-II processor) with on-chip floating-point hardware, show that the proposed PP-ER pruning technique leads to a substantial speedup (in terms of execution time) of 47%-71% in Shi-Tomasi and 10%-65% in Harris for 300 corners. In the absence of the FPU which typifies low-cost/low-power embedded systems, both Shi-Tomasi and Harris benefit from PP-ER with computational savings of 48%-82% and 45%-81% respectively. When compared to the partial pruning (PP), the PP-ER pruning technique for 300 corners, shows an average additional speedup of up to 11% for Harris and 13% for Shi-Tomasi. However, for lower threshold settings when PP fails to achieve any speedup, PP-ER achieves a much higher average additional speedup of 56% for Harris and 59% for Shi-Tomasi. Also, PP-ER enables the use of the more robust but computationally complex Gaussian filter in corner detection especially

on systems that do not support FPU. Hence, the proposed low-complexity pruning technique, PP-ER, is highly suited for corner detection in real-time and low-power vision-based embedded systems, such as the UAVs.

4

Automating Threshold Selection for Corner Detection

Introduction

Video imagery captured on-board surveillance UAVs exhibits a wide range of image content and illumination conditions, resulting in wide variations in the quality and quantity of the corners across frames. Manual intervention to set the optimal parameters for corner detection is infeasible as these videos need to be processed in real-time and on-board. In order to increase the robustness of these platforms, there is a need for automated parameter selection for corner detection which can adapt to the scene conditions. In addition, methods employed to automate the

parameter selection need to be highly compute-efficient in order to cater to the real-time requirements of these applications.

As seen in Fig. 3.1, the Shi-Tomasi [50] and Harris [51] corner detectors necessitate the selection of a threshold parameter (T_c) manually in order to identify good quality corners. This threshold is set relative to the maximum observed corner measure value in the image - for example, 1% of the maximum corner measure value is a typical choice [58]. Figure 4.1 shows the result of a fixed threshold on 2 images - the same threshold ($T_c = 0.3$) used on *trees* image results in 300 corners whereas on the *bike* image extracts only 92 corners. The conservative low threshold of 1% ($T_c = 0.01$) is set up as in [58], by standard Matlab and OpenCV implementations, in order to mitigate this problem. However, a fixed global threshold has the following issues:

1. *Widely varying image content*: The contrast and spread of textures can vary drastically, especially for aerial video applications, and a fixed threshold cannot guarantee that the required number of corners will always be found. In addition, within a single frame, the presence of a small high contrast region can elevate the threshold so much that candidates from lower contrast regions are missed out completely resulting in a very small corner set [122].

2. *Real-time processing*: Setting a conservative low threshold results in unnecessary computations for images rich with a large number of corner regions. This is illustrated in Fig. 4.1 (e) and (f) where, for the *trees* image richer with corners, a low threshold results in a very large number of corner candidates compromising the real-time implementation of corner detection.

Therefore, there is a need for the threshold selection to be automated, so that corner detection can be robust to the varying scene conditions, experienced in video applications such as aerial videos.

Drastic illumination change between frames severely affects the corner detectors, as the fixed threshold results in varied set of corners when illumination changes. In [123], the drastic illumination changes are handled by operating on logarithmic

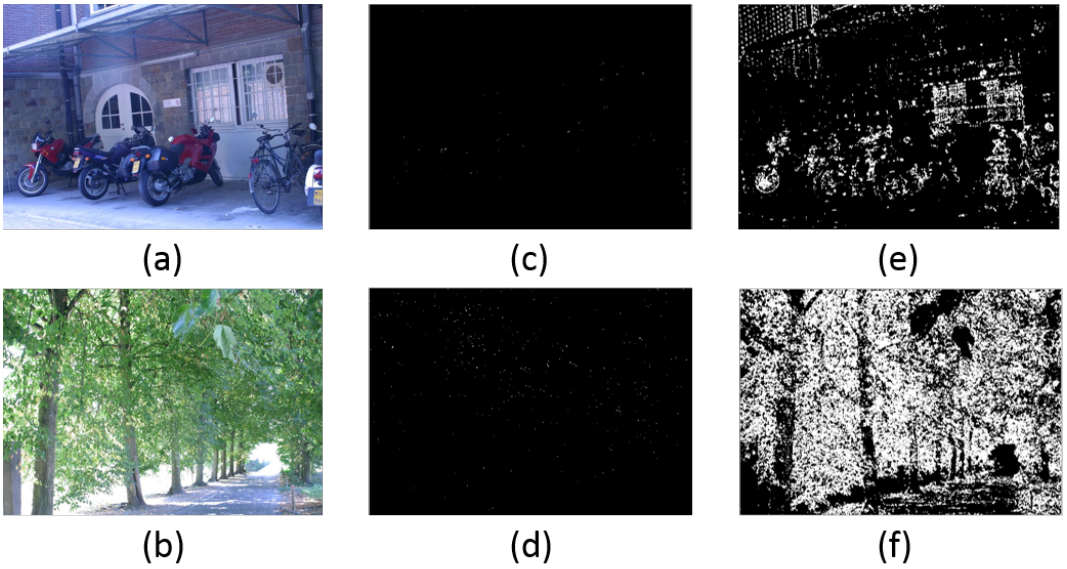

Figure 4.1: Example of inappropriate threshold settings (a) *bike* image (b) *trees* image, Corner regions at "too high" $threshold = 0.3$ for (c) *bike* resulting in only 92 corners and (d) *trees* resulting in 300 corners, and (e) and (f) Corner regions at "too low" $threshold = 0.01$.

images instead. While this approach achieves higher repeatability of the detected corners, in the face of a drastic illumination change between scenes, it does not guarantee the required number of corners when the image content varies over time. Also it increases the complexity of the corner detection by about 10% as reported in [123]. A similar approach in [124] uses moment images to counter illumination variation, but still requires a fixed threshold selection. Corner detection is applied in a block based manner such that thresholds are applied for each block independently, instead of using a global threshold for the entire frame in [122]. This can deal with situations when a very high contrast region elevates the global threshold. However, this can also lead to noisy corners for very low contrast regions [125]. The work in [125] aims to reduce the number of noisy corners when a conservative threshold is set with additional region-based checks. It still required a pair of thresholds to be specified by the user.

In this chapter, a novel iterative thresholding scheme is proposed, that eliminates the need for the user to specify the quality threshold (T_c) and guarantees the extraction of the specified number of (N) corners. As the evaluations show, the proposed method is able to adapt the computations to the image content. For

images rich with corner regions, the required number of corners can be found with lesser number of iterations. A novel non-maximum suppression (NMS) strategy is also proposed, that exploits the iterative release of corner candidates, to reduce the number of candidates processed in each iteration.

Iterative Thresholding

In the conventional Shi-Tomasi and Harris corner detectors, the corner measure C is computed on all pixels of the image (refer to Eq. 3.2 and Eq. 3.3). The parameters to threshold for quality of the corners T_c, the maximum number of corners to be detected N, and the minimum distance of separation between the extracted corners d, are specified by the user. The threshold T_c is applied on the corner measure to discard the non-corner regions: a pixel is a corner candidate if $(C > T_c * Max(C))$, where $Max(C)$ is the highest corner measure value in the image [58]. The corner candidates are then sorted based on the value of C, and non-maximal suppression is applied to extract the final corners as shown in Fig. 3.1. As seen in Section 4.1, the user-defined threshold T_c does not guarantee that the required number of N corners will be found for a given image.

Figure. 4.2 shows the proposed method for automating the threshold selection for corner detection, which guarantees the required number of N corners. It is proposed that the corner detection is started with a sufficiently high threshold T_c, to release the corner candidates. Non-maximal suppression is applied to these corner candidates and final corners are extracted. If the required number of N corners are found, the algorithm stops. If not, the threshold is lowered to release additional candidates and extract corners. This process of releasing candidates is continued until the required number of (N) corners is found.

The time complexity of such an iterative thresholding is a function of the number of sampling trials (k) and the number of candidates in each trial (n_c) as shown:

$$t = f(k, n_c) \tag{4.1}$$

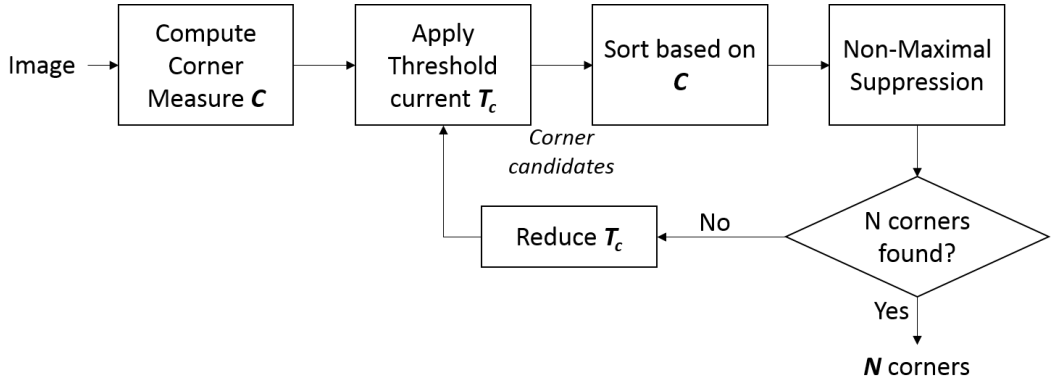

Figure 4.2: Automated thresholding for corner detection

For each trial, the image needs to be scanned to collect the corner candidates for the threshold considered. These corner candidates need to be sorted and non-maximal suppression is applied on them. Ideally, the required number of corners need to be extracted with the least number of trials and the minimum number of total candidates considered.

Now consider how the threshold T_c should be lowered. A brute force approach is an exhaustive sampling of thresholds using uniform steps between successive thresholds until N corners have been found. The OpenCV implementation (*DynamicAdaptedFeatureDetector* [76]) selects the optimal parameters for detecting corners by a similar exhaustive trial-and-error with various threshold values and involves naïve multiple executions of the detector, i.e. large number of trials (k).

In the proposed method, the following steps are used to reduce the number of trials (k) as well as the number of corner candidates n_c:

1. *Non-linear threshold steps*: The majority of the pixels in the image have very low corner measure values resulting in a non-linear distribution of pixels. The steps used to lower the threshold for corner measure T_c needs to account for this non-linear distribution of the pixels so that the number of trials k, needed to extract the required number of N corners, can be reduced.

2. *Mask-based non-maximal suppression (NMS)*: The corner candidates are processed in batches in each iteration of threshold lowering. This allows the

neighbours of already selected corners to be suppressed from being considered as candidates in future iterations. A mask-based NMS step is used to achieve this. This limits the number of candidates (n_c) processed in each trial, to only regions that are not already represented with corners.

Both these steps are discussed in detail below.

Non-linear Threshold Steps

In Fig. 4.3 (a), the histogram of pixels based on their (Shi-Tomasi) corner measure for a natural image is shown: the further in *x-axis*, the higher the corner measure. As can be seen, the distribution of pixels based on their corner measures is non-linear, as there are very few high quality corner regions compared to smooth regions. Using uniform threshold steps will release very few candidates for the higher range of corner measures needing a large number of trials (k) to extract the required number of corners. In contrast, in lower ranges of corner measures the uniform threshold step can potentially result in a very large number of candidates (n_c) being released at once, causing a sharp increase in the number of candidates (n_c) in a single trial. In order to prevent this, the threshold steps are also lowered in a non-linear manner as shown in Fig. 4.3 (b). Large steps are used for high values of corner measure that releases the candidates with less number of trials. But for lower ranges of corner measure, small steps are used so that in each trial, the number of candidates released does not surge. The initial threshold $T_c = 0.4$ and the minimum threshold is set as 0.0001. The threshold steps are chosen such that the step width falls non-linearly reducing by half each time. It was found that breaking this step width into another intermediate step with a 2:1 width further helped contain the corner candidates released in lower ranges of corner measures.

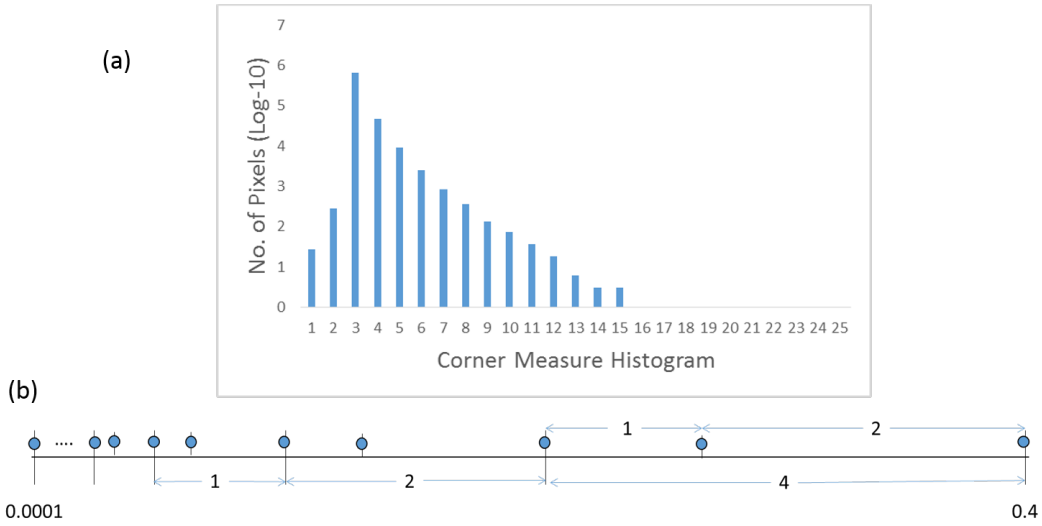

Figure 4.3: Threshold step selection: (a) Histogram of pixels based on their Shi-Tomasi corner measure for the *trees* image, (b) Threshold steps shown as circles with the step widths

Mask-based Non-Maximal Suppression

The pixels surrounding a corner also have a high corner response, and non-maximal suppression (NMS) is used to select one representative pixel in this region, which has the local maximum, as the corner. In [105] a simple and widely-used NMS scheme was proposed as shown in Listing 2. For each pixel considered as a potential corner candidate, the list of all the corners already selected is scanned to check if the current pixel is in the neighbourhood of any of the selected corners. If it is, then this pixel is suppressed; otherwise it is included as a corner.

For corner detection, the user specifies the number of corners N to be found and the minimum distance d that needs to be enforced with NMS between the corners as these parameter values are application dependent. The classic NMS scheme works well when:

1. Required number of corners N to be found is small, and/or

2. Minimum distance d between corners is small.

Listing 2 Conventional non-maximal suppression

```
1: Compute corner measure C for all pixels in image I
2: for each pixel p_i in I do
3:     if C_i > T_c * Max(C) then
4:         Add p_i to corner candidates
5:     end if
6: end for
7: for each corner candidate p do
8:     for each selected corner k do
9:         if distance(p, k) < d then
10:            Discard p
11:            Go to next p
12:        end if
13:    end for
14:    Add p to the list of selected corners
15: end for
```

In both these cases, the list of corner candidates that need to be scanned before the current corner is selected is small. However when either of these parameter values: N, in order to get a dense corner set, or d for a well-spread out corner set, the computations for NMS also increase because many more corner candidates need to be processed before the required number of corners is found.

In [69], a parallel NMS algorithm was proposed for GPU implementation that uses label maps to maintain the status of pixels during NMS. The proposed algorithm is well-suited for implementation on the parallel processing architecture for GPU, however it involves multiple scans of the image during non-maximal suppression and is not suitable for a CPU implementation.

Here, a mask-based NMS strategy is proposed using label maps similar to [69], which can be applied effectively to the automated thresholding scheme. Corners are selected in batches in the iterative thresholding method. Therefore, once a corner is selected, its neighbourhood can be suppressed from being processed as corner candidates in subsequent iterations, leading to a reduction in the corner candidates and hence, computations for sorting.

The proposed mask-based non-maximal suppression strategy is outlined in Listing 3. A binary label map, the same size as the image, is maintained. The label

Listing 3 Iterative thresholding with mask-based non-maximal suppression

```
1: Compute corner measure C for all pixels in image I
2: Initialize label map to not suppressed
3: while N corners not found do
4:     for each pixel p_i in I do
5:         if C_i > T_c * Max(C) and p_i is not suppressed then
6:             Add p_i to corner candidates
7:             C_i = 0
8:         end if
9:     end for
10:    Sort corner candidates based on C
11:    for each corner candidate p do
12:        if p is not suppressed then
13:            Add p to the list of selected corners
14:            Set label of all neighbours within d to suppressed
15:        end if
16:    end for
17:    Go to next lower T_c
18: end while
```

for each pixel can be either:

- *Not suppressed*: This is the initial state of every pixel. This means that the pixel is either not processed yet or it is not suppressed by an already chosen corner. For any pixel to be selected as a corner candidate, it needs to be *not suppressed*.
- *Suppressed*: This is the state of a pixel that is in the vicinity (determined by the minimum distance d) of an already selected corner.

In each iteration of processing corner candidates as in Fig. 4.2, the candidates are ranked based on their corner measure and selected as corners if they are *not suppressed*. Once a candidate is selected as a corner, the neighbourhood pixels within d are all updated to a *suppressed* status. The neighbourhood can be determined using precomputed masks based on the minimum distance parameter d as shown in Fig. 4.4. When releasing new corner candidates in a subsequent iteration, it is checked if the candidate is also *not suppressed*. If not, this corner candidate is discarded from further processing.

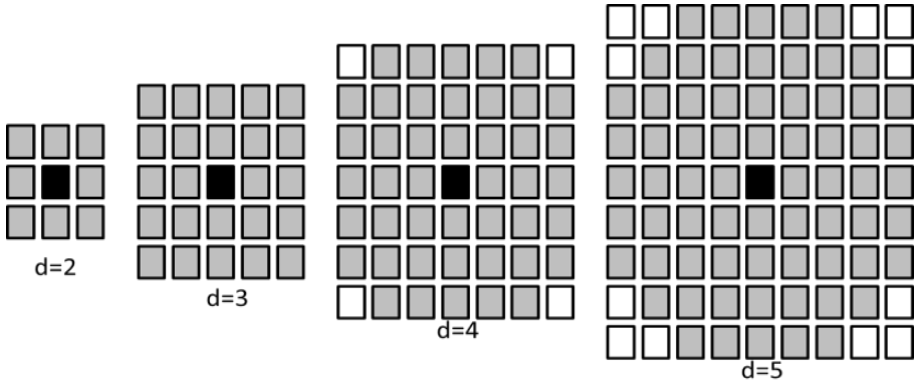

Figure 4.4: Non-maximal suppression masks for various minimum distance values, *d*. (Grey pixels are suppressed by the black pixel)

In this section, an automated thresholding scheme has been proposed, that iteratively releases corner candidates and performs a mask-based non-maximal suppression. This results in corner detection that is adaptive to the image content and eliminates the need for manually specifying the threshold parameter for corner quality.

Performance Evaluations

In this section, the proposed iterative thresholding method as outlined in Listing 3, is applied to Shi-Tomasi and Harris detectors, and its performance is evaluated.

Evaluation Setup: The baseline algorithms are the Shi-Tomasi (ST) and Harris (H) corner detectors with a *fixed* threshold and conventional NMS as in Listing 2. These parameters are used: the required number of corners $N = 300$, minimum distance between corners $d = 5$ and threshold $T_c = 0.01$. For the proposed iterative thresholding (IT) method, the threshold steps used are shown in Table 4.1 setup as in Fig. 4.3. The C implementation for the baseline corner detectors developed in Section 3.5 is extended to include the iterative thresholding and mask-based NMS as in Listing 3. The image data used is shown in Fig. 4.5.

Evaluation metric: As the main objective of the iterative thresholding is to automatically release *minimum* number of corner candidates for a given image, in

Iteration No.	Threshold T_p	Step Size
1	0.4000	-
2	0.2662	0.1338
3	0.1993	0.0669
4	0.1324	0.0669
5	0.0989	0.0335
6	0.0654	0.0335
7	0.0487	0.0167
8	0.0320	0.0167
9	0.0236	0.0084
10	0.0153	0.0084
11	0.0111	0.0042
12	0.0069	0.0042
13	0.0048	0.0021
14	0.0027	0.0021
15	0.0017	0.0010
16	0.0006	0.0010
17	0.0001	0.0005

Table 4.1: Threshold steps for automated thresholding technique

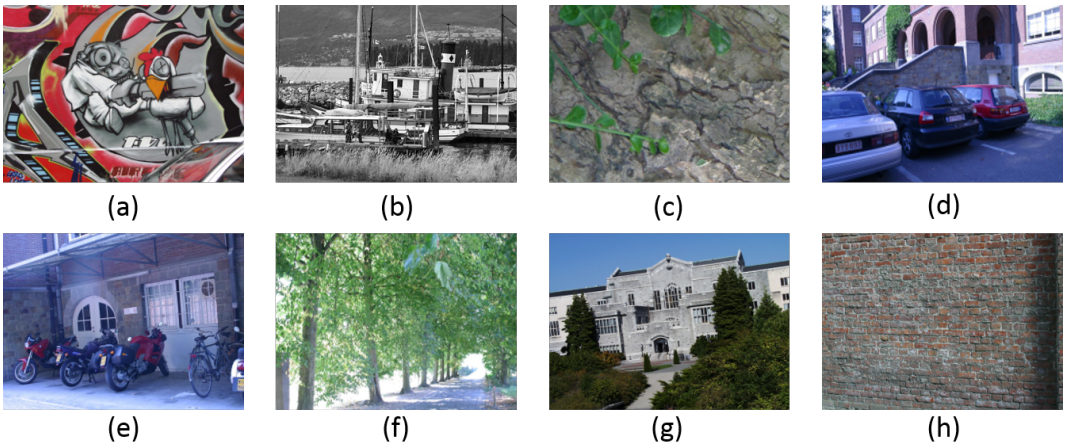

Figure 4.5: Image dataset [119] used for the evaluation (a) *graf* (800x640) (b) *boat* (850x680) (c) *bark* (765x512) (d) *leuven* (900x600) (e) *bike* (1000x700) (f) *trees* (1000x700) (g) *ubc* (800x640) (h) *wall* (1000x700)

order to extract the required N corners, the total number of corner candidates after thresholding N_c, is used as the evaluation metric for efficiency.

In the baseline detectors, the number of corner candidates after applying the *fixed* threshold T_c is computed, as N_c^{fixed}. For the proposed iterative thresholding method, if n_c corners are released in a single iteration, the total number of corner candidates released at the end of all the iterations given by $N_c^{iterative}$ is:

$$N_c^{iterative} = \sum_{Iter} n_c \tag{4.2}$$

The corner candidates size normalised by the image size I_{size} given by $\widetilde{N_c} = N_c/I_{size}$ is reported. Additionally, the *relative* reduction in the number of corner candidates when iterative thresholding replaces the fixed threshold, represented as ΔN_c, is also reported, given by:

$$\Delta N_c = \frac{N_c^{fixed} - N_c^{iterative}}{N_c^{fixed}} \tag{4.3}$$

Table 4.2 shows that the number of corner candidates with the *fixed* threshold for the baseline detectors (ST and H) has a wide variation among the various images. While *graf* releases only 5.8% with the fixed threshold of $T_c = 0.01$, *wall* releases 59.4% of the candidates. This confirms that a single fixed threshold is not appropriate for all image content. With iterative thresholding, the number of corner candidates falls to 0.2-0.7% for all images. This corresponds to 94.4-99.6% *relative* reduction in the total number of corner candidates for Shi-Tomasi and 71.3-98.4% for Harris corners, when iterative thresholding replaces a fixed threshold to detect $N = 300$ corners. Naturally, images with very distinct textures that released large number of candidates with the fixed threshold - such as *trees* and *wall* - see the maximum reduction in candidates when iterative thresholding is employed instead.

It is also worth noting that for a given image, the number of candidates released for a fixed threshold $T_c = 0.01$, is in general higher for Shi-Tomasi than Harris

	Shi-Tomasi			Harris		
	$\widetilde{N_c}$ %		ΔN_c	$\widetilde{N_c}$ %		ΔN_c
Images	ST	ST+IT		H	H+IT	
graf	5.8%	0.3%	94.4%	1.1%	0.3%	71.3%
boat	26.6%	0.4%	98.7%	4.4%	0.3%	94.0%
bike	8.0%	0.3%	96.4%	0.9%	0.3%	66.2%
bark	44.2%	0.7%	98.5%	2.4%	0.5%	79.3%
leuven	15.8%	0.3%	98.1%	3.8%	0.3%	91.5%
ubc	29.4%	0.2%	99.2%	3.0%	0.3%	90.0%
trees	49.8%	0.3%	99.5%	9.1%	0.1%	98.4%
wall	59.4%	0.3%	99.6%	12.5%	0.2%	98.1%

Table 4.2: Reduction in the total number of corner candidates with iterative thresholding (S+IT and H+IT) for 300 Shi-Tomasi and Harris corners: total number of corner candidates $\widetilde{N_c}$ and relative reduction when iterative thresholding replaces fixed threshold ΔN_c

- for example, *graf* releases 5.8% Shi-Tomasi candidates but only 1.1% Harris candidates. This shows that the same fixed threshold does not work across Shi-Tomasi and Harris as the corner measure distributions of the two methods differ. Therefore the *relative* reduction in corner candidates seen in Table 4.2 with the proposed iterative thresholding is lesser for Harris compared to Shi-Tomasi for a threshold of $T_c = 0.01$.

To further demonstrate the impact of IT, the results with a larger number of required corners, $N = 1000$, are shown in Table 4.3. For Shi-Tomasi, as the same threshold $T_c = 0.01$ can still be used to detect $N = 1000$ corners, the baseline releases the same number of corner candidates. Evidently, to extract 1000 corners, more candidates will need to be released and processed by the proposed method compared to the case of $N = 300$. This is also seen in the increase in the number of iterations/trials (k) that need to be done in iterative thresholding, when N is increased from an average of 2.75 for $N = 300$ to 5 for $N = 1000$. For Harris, the threshold of $T_c = 0.01$ is too high and does not extract sufficient corners for images *graf* and *bike* when $N = 1000$, highlighting the limitation of a fixed threshold. Therefore, a lower threshold of $T_c = 0.001$ is used for the baseline Harris (referred to as H(L) in Table 4.3) and the results in % *relative* reduction for both the threshold settings are shown.

	Shi-Tomasi			Harris				
	$\widetilde{N_c}$ %		ΔN_c	$\widetilde{N_c}$ %			ΔN_c	
Images	ST	ST+IT		H	H (L)	H+IT	Vs H	Vs H (L)
graf	5.8%	2.0%	65.3%	1.1%	3.0%	2.0%	-90.5%	32.1%
boat	26.6%	1.7%	93.8%	4.4%	14.0%	1.7%	61.7%	88.0%
bike	8.0%	1.5%	81.3%	0.9%	4.0%	1.6%	-72.3%	60.8%
bark	44.2%	1.4%	96.8%	2.4%	18.3%	2.1%	11.8%	88.4%
leuven	15.8%	2.5%	84.4%	3.8%	9.7%	2.8%	24.3%	70.7%
ubc	29.4%	1.9%	93.6%	3.0%	13.5%	2.2%	27.4%	83.8%
trees	49.8%	0.8%	98.4%	9.1%	32.4%	0.8%	91.6%	97.6%
wall	59.4%	0.8%	98.7%	12.5%	41.7%	1.1%	90.9%	97.3%

Table 4.3: Reduction in the total number of corner candidates with iterative thresholding (S+IT and H+IT) for 1000 Shi-Tomasi and Harris corners: ST and H use $T_c = 0.01$ whereas H(L) uses a lower threshold $T_c = 0.001$ so that $N = 1000$ can be found in all images

Next, the proposed iterative thresholding (ST + IT and H + IT) is compared with the following options for iteratively reducing the threshold where the sampling steps are equal:

- *Exhaustive iterative thresholding*: The thresholds are exhaustively sampled by dividing the threshold range into 100 uniform steps. This is referred to as EIT - for exhaustive iterative thresholding, specifically as ST + EIT and H + EIT.

- *Coarse iterative thresholding*: In order to show the effectiveness of non-uniform threshold steps, a uniform threshold step scheme is employed that uses the same number of steps as the proposed method (in this case, 20). This is referred to as ST + CIT and H + CIT.

Table 4.4 shows that using uniform steps for threshold lowering results in much higher number of trials (k) to release the required corner candidates, whereas with non-linear steps, the proposed IT method required fewer trials. The second advantage of using non-linear steps is seen in the % relative reduction of candidates (ΔN_c) for Harris. CIT uses uniform steps for all thresholds and releases many more candidates in lower threshold ranges resulting in poorer % relative reduction of candidates (ΔN_c). This is also seen in Table 4.5 when $N = 1000$ where for several

	% Relative reduction in candidates ΔN_c	No. of trials (k)
ST + IT	98.1%	2.8
ST + EIT	98.8%	34.5
ST + CIT	98.6%	7.8
H + IT	86.1%	5.6
H + EIT	88.5%	74.8
H + CIT	81.5%	15.6

Table 4.4: Comparison of iterative thresholding methods for 300 Shi-Tomasi and Harris corners

	% Relative reduction in candidates ΔN_c	No. of trials (k)
ST + IT	89%	5
ST + EIT	91.1%	66.6
ST + CIT	87.1%	14.1
H + IT	77.3%	9.8
H + EIT	60.8%	91.9
H + CIT	1.6%	18.6

Table 4.5: Comparison of Iterative Thresholding methods for 1000 Shi-Tomasi and Harris corners

images, the uniform steps for EIT and CIT result in far higher candidates being released than the proposed IT method. This shows that by reducing the step size in a non-uniform manner, the proposed iterative thresholding also reduces the number of candidates released in the lower ranges of corner measures.

The results thus far use the conventional NMS as in Listing 2. Table 4.6 shows that when the proposed mask-based NMS as in Listing 3 is applied, further reduction in the number of candidates can be achieved, with an average relative reduction (average of ΔN_c for all images) of more than 95%. The impact of the mask-based NMS is higher when N is increased because many of the candidates that are released in the trials with low thresholds have been already suppressed by the chosen corners in the earlier trials.

Summary

In this chapter, an iterative thresholding method was presented to automate the selection of quality threshold in the Shi-Tomasi and Harris corner detectors that

	Shi-Tomasi		Harris	
N	Conv NMS	Mask NMS	Conv NMS	Mask NMS
300	98.0%	99.1%	86.1%	95.5%
1000	89.0%	97.3%	77.3%	96.0%

Table 4.6: Relative reduction (%) in corner candidates (ΔN_c) of proposed mask-based NMS vs. conventional NMS with iterative thresholding S + IT and H + IT

leads to savings in computations for corner detection, by being adaptive to the image content. Unlike existing methods that involve exhaustive sampling of thresholds which cannot be realized in real-time processing, a non-linear threshold sampling scheme was proposed that employs varying threshold steps depending on the corner response. This leads to a faster release of the required number of corner candidates, at the same time, preventing a surge of candidates in lower ranges of corner measures. In effect, the proposed method is able to find a near-optimal threshold that releases just enough candidates for extracting the required number of corners. In addition, a novel mask-based non-maximal suppression technique was proposed [4], that complements the iterative release of corner candidates, by suppressing the neighbours of corners selected after the current release. This results in a substantial reduction in the number of corner candidates processed in subsequent iterations of sampling with lower thresholds.

The proposed iterative thresholding with mask-based NMS technique, results in an average reduction in the number of corner candidates of 98.2% for Shi-Tomasi and 95.7% for Harris compared to using fixed thresholds. Thus the proposed method enables the Shi-Tomasi and Harris corner detectors to be employed, without the need for manual intervention and with low-complexity, in applications where the image content varies drastically, as is the case in aerial videos captured on-board surveillance UAVs.

5

Accelerating Automated Thresholding with Pruning

In Chapter 4, an automated thresholding method was proposed, that eliminated the need to manually select the quality threshold parameter for Shi-Tomasi and Harris corner detectors. The method was able to find a near-optimum threshold that released just enough candidates for extracting the required number of corners. Earlier in Chapter 3, a compute-efficient pruning technique [1] was proposed that was used to rapidly extract corner candidates for Shi-Tomasi and Harris corner detectors. In this chapter, the automated thresholding method is combined with the low-complexity pruning technique resulting in further reduction in the computational complexity of Shi-Tomasi and Harris corner detectors at the same time making them robust under wide range of image content [3, 4].

Iterative Thresholding with Pruning

In Section 3.3.1, a pruning measure P (Eq. 3.5) was introduced that is computationally simpler than the corner measure C, to select corner candidates, and an edge removal step ER (as in Section 3.3.2) was employed to further enhance the pruning process. The complex corner measure C was then applied to only the small pool of corner candidates selected after the pruning process. In this section, the pruning technique is combined with the automated thresholding scheme proposed in Chapter 4.

As seen in Fig. 5.1, the distribution of the image pixels based on the pruning measure P, is non-linear, similar to the distribution of the corner measure C. This is expected because the pruning measure is derived from the corner measure itself. Therefore first, the pruning measure P is computed for all pixels in the image. Similar to the lowering of threshold T_c in Chapter 4, the pruning threshold T_p is lowered in each iteration using the threshold steps as shown in Fig. 4.3 (b) to release corner candidates in a controlled manner.

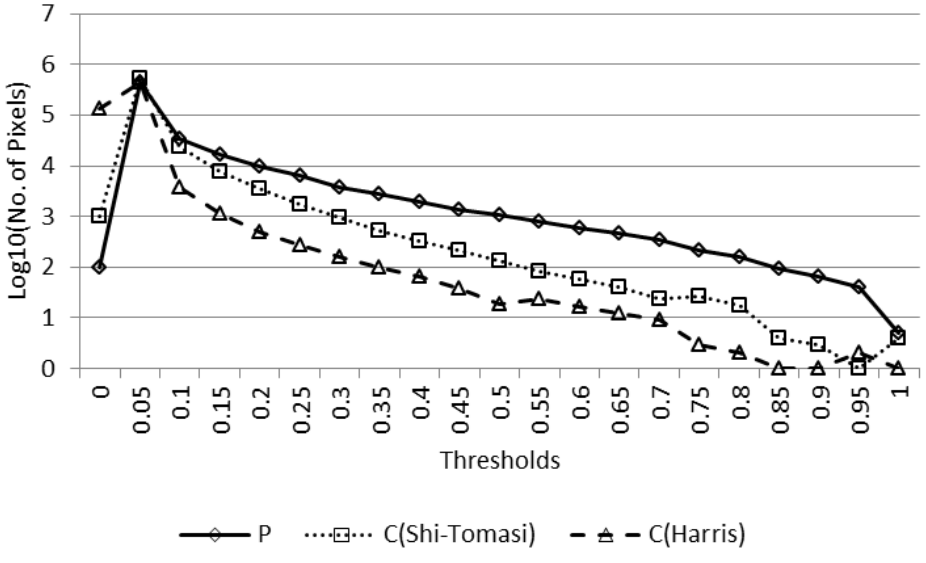

Figure 5.1: Histogram of pixels based on the pruning measure P, and corner measures C for Shi-Tomasi and Harris

Figure. 5.2 shows the block diagram for the proposed iterative thresholding with pruning technique. As described earlier, the release of the corner candidates is

controlled by the lowering of the threshold for pruning measure T_p. The corner measure C is computed only on these corner candidates. A novel bin-based approach is used to collect the corner candidates and select the best corners without the need to threshold with T_c. The iterative processing stops when the required number of N corners has been found.

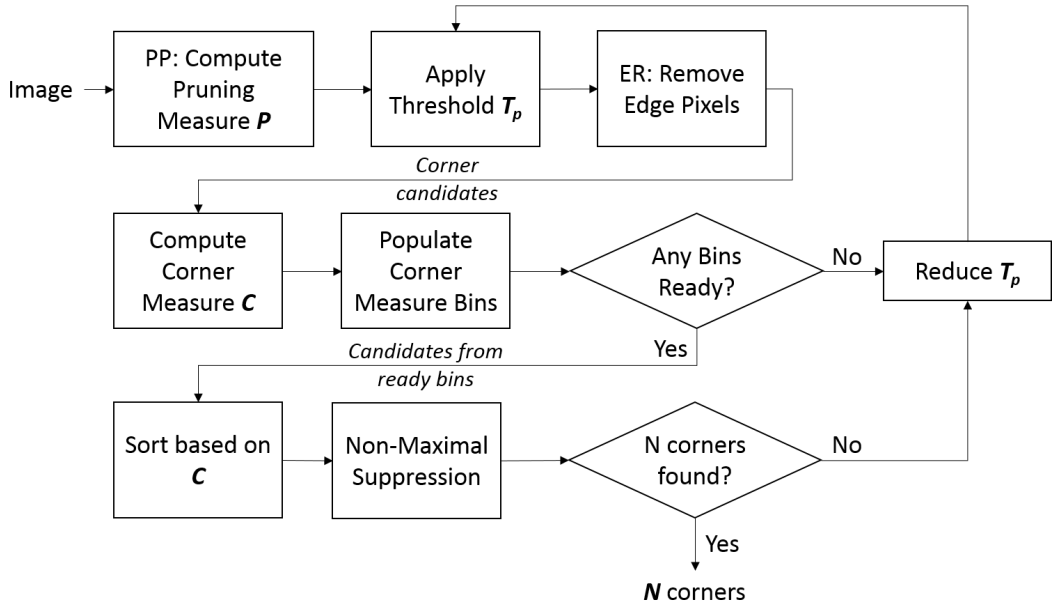

Figure 5.2: Proposed automated thresholding with pruning technique

The pruning threshold T_p is lowered in a controlled manner to release candidates on which the computationally complex corner measure C is computed. The process stops when the required number of N corners is found. For images that are rich with good corners, the proposed technique can detect these corners with a small number of iterations and a very small corner candidate set. Hence, the method becomes compute-efficient by being adaptive to the image content.

The bin-based processing of corner candidates is the critical step in the proposed method. The entire range of values of corner measure C that a corner candidate can take is divided into a predefined number of bins. Every candidate is then assigned a bin, based on its corner measure C. This results in a crude ordering of the corner candidates such that, the bins for the highest ranges of corner measure values contain the best corner candidates in the image.

In every iteration, as more corner candidates are classified into bins, the growth of each bin is monitored. If a bin stops growing, it is deemed to have *saturated* and the candidates in this bin are ready for further processing. Only the corner candidates from the *saturated* bins are then passed on for further sorting and non-maximal suppression. At the end of an iteration, if the required number of corners have been accumulated, the algorithm stops. If not, the threshold T_p is lowered in a systematic manner and the next iteration starts.

Figure. 5.3 shows the automated thresholding with pruning method at work for detecting 1000 Shi-Tomasi corners in the *boat* image. On the left, the histogram of the image pixels is shown based on the pruning measure P. The y-axis shows the number of pixels as $log_{10}(no.of pixels)$. On the right, the corner measure bins have been shown. The value of the corner measure increases with the bin number; for example, bin 18 holds the best quality corners. In each iteration, as the threshold on pruning measure T_p is lowered, the corner measure C is computed on the candidates released, sorted based on C and placed into the corresponding corner measure bins. In the 3rd iteration, bin 18 *saturates*, i.e. stops growing. The candidates from this bin are then sorted, NMS is applied and a total of 231 final corners is obtained from this bin. This process continues, and corners are accumulated in each iteration. The technique stops after the 6th iteration as the required number of 1000 corners has been found.

The proposed automated thresholding method is outlined in Listing 4. The key elements of the automated thresholding technique are:

1. Selection of pruning threshold steps and the bin boundaries of the corner measure bins,
2. Detection of saturating bins,
3. Efficient non-maximal suppression, and
4. Deterministic and compute-efficient convergence of the iterations.

Next, each of these elements is discussed in greater detail.

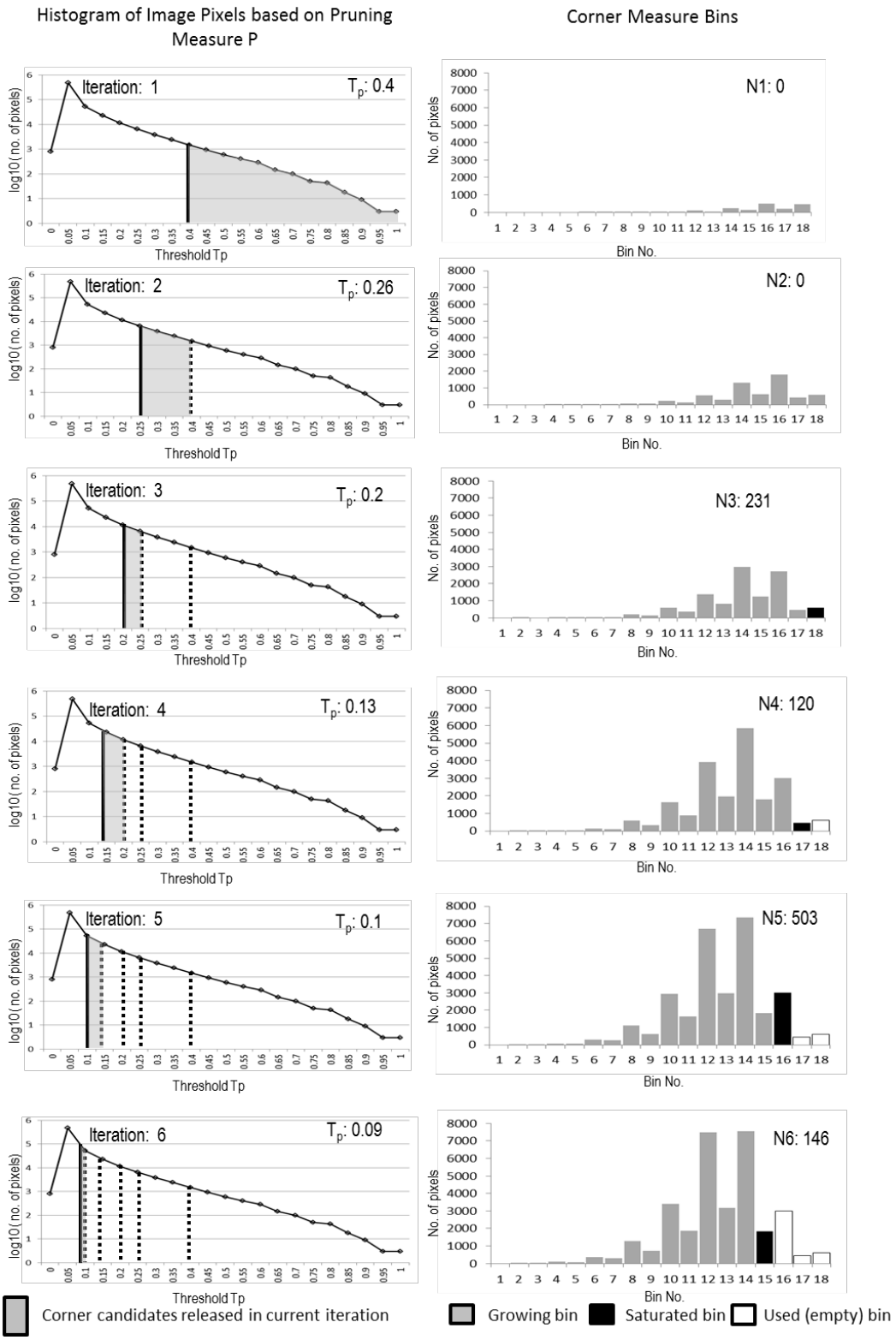

Figure 5.3: Automated thresholding with pruning at work: Lowering of pruning threshold T_p and *saturation* of corner measure bins in automated thresholding technique used to detect 1000 corners

Listing 4 Iterative thresholding with pruning for corner detection

```
1: Compute pruning measure P for all pixels in image I
2: Initialize NMS label map to not suppressed
3: while N corners not found do
4:     for each pixel p_i in I do
5:         if P_i > T_p * Max(P) and p_i is not suppressed then      ▷ Pruning (PP)
6:             if p_i passes the Edge Removal step then             ▷ Pruning (ER)
7:                 Add p_i to corner candidates C_i
8:                 Compute corner measure C
9:             end if
10:        end if
11:    end for
12:    Classify candidates in C_i into corner measure bins b_k
13:    for each bin b_k starting from the highest quality bin do    ▷ Scan the bins
14:        if b_k has saturated then
15:            Sort candidates in bin b_k in descending order of C
16:            for each candidate p_i in sorted bin b_k do
17:                if p_i is not suppressed then
18:                    Select p_i as final corner                   ▷ Select corner
19:                    Update NMS label map of neighbours of p_i within distance d
20:                end if
21:            end for
22:        else
23:            Exit the bin scan
24:        end if
25:    end for
26:    Select the next lower T_p
27: end while
```

Selection of Pruning Threshold Steps and Bin Boundaries

As the pruning measure P has been derived from the corner measure C, the threshold steps shown in Fig. 4.3 (b) are applicable to the threshold T_p as well. Therefore, the threshold T_p is reduced in the same manner as threshold T_c as described in 4.2.1.

The boundaries of the corner measure bins used to collect the corner candidates, are also setup in a non-linear manner. This ensures that each bin has roughly similar number of pixels; this property of the bins is necessary for them to *saturate*. The bin boundaries have been chosen as shown in Fig. 4.3. First the major bins are setup such that the range of thresholds for every bin is half that of its higher

neighbour major bin. Every major bin is then sub-divided into a 2:1 distribution to ensure that there is not a huge surge of pixels for a new major bin. When assigning every corner candidate to a bin, first its major bin is found, followed by the sub-bin. Note that the bin boundaries are specified as a factor of the highest corner measure value $max(C)$ found for the image.

Detection of Saturating Bins

After the corner candidates have been classified into the corner measure bins, in each iteration, the growth of each bin is monitored, for detecting *saturation* in the bins. The bins are scanned from the highest corner measure. If the current bin has *saturated*, its corner candidates are released for further processing. The scan is stopped at the first bin that is found to be growing. This way the best corners in the image are extracted from bins that have *saturated* and are not likely to grow any more.

The reason for bins to *saturate* is the high degree of correlation between the pruning measure P and the corner measure C. Pixels with high values of P are also likely to have high values of C. Therefore, after a couple of iterations the higher quality bins stop growing and *saturate* as can be observed in Fig. 5.3. The candidates in these bins are released, sorted and NMS is applied to them, thereby extracting final corners.

Efficient Non-Maximal Suppression

The mask-based non-maximal suppression (NMS) strategy in Section 4.2.2 when applied to the proposed iterative thresholding with pruning method, leads to further reduction in computations. This is because the size of corner candidate sets selected in each iteration that undergo the complex corner measure is also further reduced. The pseudo code for the mask-based NMS is shown in Listing 4.

	Max. growth for saturation (%)
Bins 1-5	1
Bins 6-11	5
Bins 12-18	10

Table 5.1: Criteria for saturation of corner measure bins

Deterministic and Compute-Efficient Convergence of Iterations

In order to ensure that the proposed iterative thresholding with pruning method conserves computations as much as possible, in spite of the non-linear nature of the image, several checks are applied.

1. *Saturation criterion for corner measure bins*: It was observed that due to the non-linear nature of the corner measure distribution, in the lower quality ranges, the corner measure bins continue to grow, with every iterative release of corner candidates. Therefore, a stringent *saturation* criterion prevents corner candidates in the lower quality corner measure bins from being processed, as these bins do not *saturate*. Therefore, the saturation criterion is relaxed for lower quality bins as shown in Table 5.1.

2. *Lowering of pruning threshold T_p*: At the end of every iteration, it is determined if the proposed method is close to finding the required N number of features or not. If it is close, the predefined steps to lower the threshold T_p are replaced by a conservative step (that is much smaller than the predefined step). This ensures that the candidates are released in a controlled manner depending on the number of corners that are remaining to be found.

   This estimation is based on a measure of the *yield* of corners from the *saturating* bins, in the current iteration. The *yield* is a crude indicator of the density of the corner candidates - for example, if the candidates are densely located, then the non-maximal suppression leads to many of the corner candidates being suppressed, resulting in a lower yield.

First, the number of corners remaining to be found ($n_{remainingCorners}$) is determined as shown:

$$n_{remainingCorners} = N - n_{chosenCorners} \tag{5.1}$$

Where $n_{chosenCorners}$ is the number of corners found thus far and N is the total number of corners required to be found.

The current yield, represented as $r_{current}$, refers to the yield at the end of the current iteration of corner candidates processing. Given the total number of pixels in the *saturating* bins, $n_{currentSaturating}$ and the total number of corners selected in the current iteration, $n_{currentCorners}$, $r_{current}$ is computed as:

$$r_{current} = \frac{n_{currentSaturating}}{n_{currentCorners}} \tag{5.2}$$

Similarly, the yield needed in the next iteration for the $n_{remainingCorners}$ to be selected is represented as r_{next} and computed as:

$$r_{next} = \frac{n_{nextSaturating}}{n_{remainingCorners}} \tag{5.3}$$

Where $n_{nextSaturating}$ refers to the number of pixels in the bin that is next in line to saturate and $n_{remainingCorners}$ refers to the number of corners remaining to be found.

When $r_{current} > r_{next}$, it indicates there are not enough candidates in the next bin to extract the remaining number of corners and therefore more candidates need to be released. This is an indication to go for a regular iteration with the predefined steps for threshold lowering. When $r_{current} < r_{next}$, it indicates that the method is likely to find the required number of corners with the next bin when it *saturates*. Therefore, a controlled release of candidates can be done, to ensure that the next bin indeed *saturates*. In the implementation, the following condition is used: $r_{next} > r_{current} + 2$ to trigger a controlled release of candidates.

Performance Evaluations

In this section, the combined iterative thresholding with pruning method (referred to as Automated Thresholding or AT) is evaluated in comparison to fixed/manual threshold techniques for Shi-Tomasi and Harris corner detectors. Following this, the additional reduction in complexity achieved by employing the mask-based NMS with the automated thresholding is demonstrated.

Evaluation Setup

For the evaluation of the proposed automated thresholding with pruning technique, the following *fixed* threshold techniques are used:

1. *Conventional Shi-Tomasi (ST) and Harris (H) detectors*: Fixed threshold of $T_c = 0.01$ for Shi-Tomasi and $T_c = 0.001$ for Harris is used.

2. *Pruning based Shi-Tomasi (ST+P) and Harris detectors (H+P)*: The pruning technique PP-$\text{ER}_{\text{ST/H}}$ [1], as described in Section 3.3, is applied to both the ST/H detectors. A *fixed* threshold for the pruning measure of $T_p = 0.01$ is used. The threshold T_c is the same as that of the conventional ST/H.

The proposed automated thresholding with pruning technique is referred to ST+AT and H+AT when applied to Shi-Tomasi and Harris corner detectors respectively. The threshold steps and corner measure bin boundaries, in the proposed automated thresholding technique, are setup to release corner candidates that have a non-linearly increasing distribution as shown in Table 4.1 and 5.2 respectively. Table 4.1 shows that as iterations increase, the step size by which the pruning threshold T_p is lowered reduces and this ensures that corner candidates released in a given iteration are not very large in number compared to the previous iteration. Table 5.2 shows the bin boundaries for the corner measure bins used to collect the corner candidates, are also setup such that the width of bins for lower threshold

	Threshold Factors		
Bin No.	Min	Max	Bin Width
1	0.4	1	0.6
2	0.2662	0.4	0.1338
3	0.1993	0.2662	0.0669
4	0.1324	0.1993	0.0669
5	0.0989	0.1324	0.0335
6	0.0654	0.0989	0.0335
7	0.0487	0.0654	0.0167
8	0.032	0.0487	0.0167
9	0.0236	0.032	0.0084
10	0.0153	0.0236	0.0084
11	0.0111	0.0153	0.0042
12	0.0069	0.0111	0.0042
13	0.0048	0.0069	0.0021
14	0.0027	0.0048	0.0021
15	0.0017	0.0027	0.001
16	0.0006	0.0017	0.001
17	0.0001	0.0006	0.0005

Table 5.2: Corner measure bin boundaries

values are much smaller than the higher threshold values. A maximum of $N = 300$ corners is extracted in all cases.

The following images are used - *bike*, *trees* and images of Mars from the navigation camera of the Curiosity rover [126] - as shown in Fig. 5.4 for the evaluations. Images *bike* and *trees* have an image size of 1000x700 pixels. Images *mars1-5* have an image size of 1024x1024.

The C implementations of the baseline algorithms have been used from Section 3.5. The proposed automated thresholding (AT) method has also been implemented in C as outlined in Listing 4.

The compute-efficiency of the proposed automated thresholding technique is demonstrated by executing all the algorithms on the Nios-II embedded platform (as described in Appendix B) and measuring the execution time. All the algorithms are executed as applications on the Nios-II/f fast core with the on-board cache and floating-point units enabled.

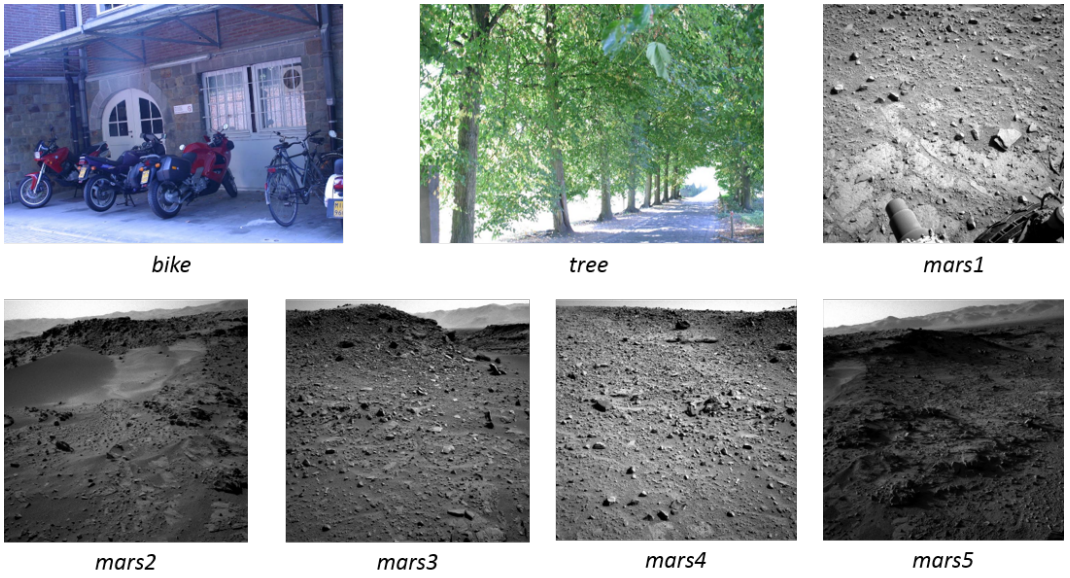

Figure 5.4: Image data [119, 126] used for evaluation of automated thresholding with pruning (AT) technique

Accuracy Evaluation

The *feature matches*, μ as described in Section 3.5.2, and computed using Eq. 3.13 and Eq. 3.14, is used as the evaluation criterion for the accuracy. This metric shows how close the corner sets generated by the pruning based method is, compared to the conventional detectors without pruning.

Table 5.3 shows that the feature matches, μ for the automated thresholding technique (ST+AT, H+AT) is the same as that of fixed thresholding pruning (ST+P, H+P) and is an average of 98% for Shi-Tomasi and 97.3% for Harris. This shows that the proposed method (AT) is able to extract corner sets which are a very close match of the corresponding conventional detector, without needing to specify the threshold T_c or T_p manually. This can be attributed to the novel strategy, to wait for the corner measure bins to *saturate* before the corners are selected, in each iteration of pruning. This ensures, that the loss of any high quality corners during iterative pruning is minimal.

Feature matches μ (%)				
Images	ST+P	ST+AT	H+P	H+AT
bike	93.1	93.1	90.7	90.7
trees	97.9	97.9	95.1	95.1
mars1	100	100	100	100
mars2	99.7	99.3	99.7	99.3
mars3	100	100	99.3	99.3
mars4	100	100	99.7	99.7
mars5	97	96.7	97	97

Table 5.3: Accuracy evaluation for proposed automated thresholding with pruning technique

Efficiency Evaluation

Figure 5.5 shows the execution time on Nios-II, for all the corner detection methods considered. Although images *bike* and *trees* have the same image size, both the conventional detectors (ST and H) have a longer execution time for *trees* compared to *bike* as it has more textures and hence more corner candidate regions to process before the required number of corners can be extracted. A similar variation in execution times is also observed for the Mars images e.g. *mars1* which has a higher contrast and hence more corner candidates has a higher execution time compared to *mars5*.

When pruning with the *fixed* threshold is applied (ST+P and H+P), *bike* shows computation savings because the corner measure is applied to a small pool of corner candidates selected by the pruning process. However, *trees* does not show substantial improvement in the execution time in ST+P and incurs an overhead in H+P. This is because the threshold for pruning, T_p and corner measure T_c is too low for this image and the pool of corner candidates is very large. In such cases, pruning leads to additional computations. Similar observations were observed in Section 3.5.3 in the case of the *wall* image. These results show that, if the image content is unknown, then setting a fixed threshold in a conservative manner is not computationally efficient as the number of corner regions that are selected by the fixed threshold is dependent on the nature of the image content and therefore, unnecessary computations are incurred for images rich with corner regions.

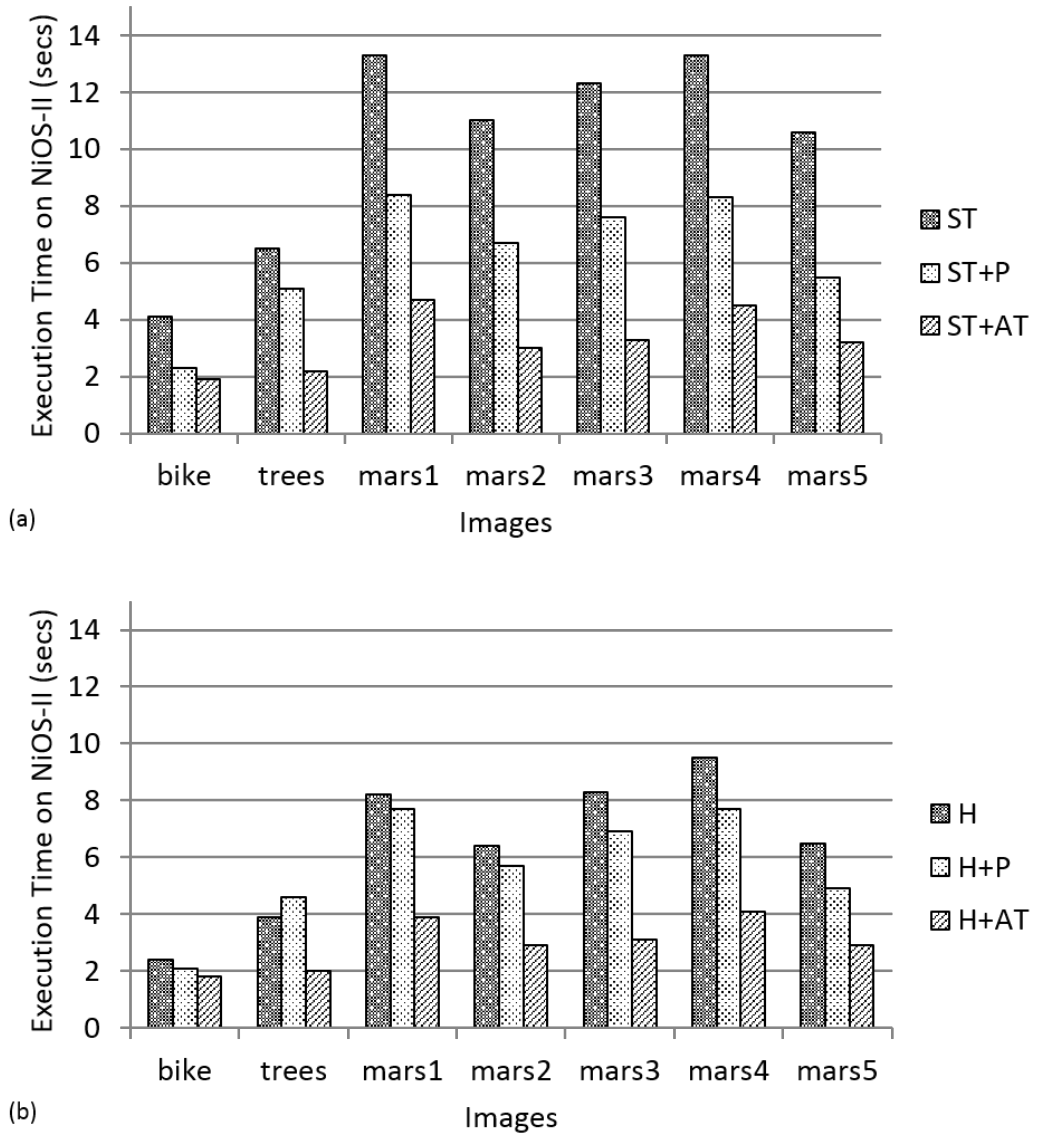

Figure 5.5: Efficiency evaluation with Nios-II for the proposed automated thresholding with pruning technique for (a) Shi-Tomasi and (b) Harris corner detectors

The automated thresholding technique (ST+AT, H+AT) alleviates this problem and exploits the nature of the image content to achieve high computation savings. For ST+AT, an average speedup in execution time, ψ (as in Eq. 3.16) of 67% is achieved compared to the conventional algorithm, ST, while HT+AT achieves an average speedup of 51% compared to the conventional algorithm, H. Rich images such as *trees* have a low speedup with fixed threshold pruning, specifically, -22% with ST+P and -18% with H+P (i.e. 22% and 18% overhead respectively). But

with automated thresholding, the pruning is adaptive to the image content and the required number of corners is found at a higher threshold resulting in a speedup in execution time ψ of 66% with ST+AT and 49% with H+AT. Note that the Nios-II platform has the floating point unit (FPU) enabled for all these evaluations. As seen in Section 3.5.3, even higher speedup in execution time can be expected when the FPU is disabled. The evaluations show that it is possible to automate the thresholding process for corner detection and guarantee the required number of N corners, and also achieve this with lower computational complexity.

Automated Thresholding with Mask-based NMS

The evaluations thus far used the conventional non-maximal suppression strategy as outlined in Listing 2. In this section, the additional benefits, of incorporating the mask-based NMS strategy as outlined in Listing 3 with the automated thresholding (AT) technique, are demonstrated.

The images in Fig 4.5 have been used for the evaluations. The proposed mask-based NMS strategy is applied with automated thresholding (AT) and is referred to as AT + maskNMS. It is compared with automated thresholding with conventional NMS strategy, referred to as AT + convNMS.

The main objective of the mask-based NMS strategy is to reduce the total number of candidates after pruning, that undergo the complex corner measure. Let the number of corner candidates for the automated thresholding with conventional NMS (AT + convNMS) be $N_c^{AT+convNMS}$ and for the mask-based NMS (AT + maskNMS) be $N_c^{AT+maskNMS}$, counted by summing up the number of candidates released in each iteration as described in Eq. 4.2. The additional reduction (denoted as $\widetilde{\delta_c}$), achieved by applying the mask-based NMS instead of the conventional NMS strategy, normalized with the image size I_{size}, is computed as shown:

$$\widetilde{\delta_c} = \frac{N_c^{AT+maskNMS} - N_c^{AT+convNMS}}{I_s} \tag{5.4}$$

	N=300			*N=1000*		
Methods	d=5	d=10	d=15	d=5	d=10	d=15
ST + AT + maskNMS	3%	12%	28%	10%	23%	37%
H + AT + maskNMS	9%	27%	38%	28%	39%	52%

Table 5.4: Additional reduction in the corner candidates ($\widetilde{\delta_c}$) with mask-based NMS for automated thresholding technique

In Table 5.4, the additional reduction in candidates ($\widetilde{\delta_c}$) by using the mask-based NMS with the automated thresholding is shown. It is clear that it is always beneficial to apply the mask-based NMS in place of the conventional NMS as $\widetilde{\delta_c}$ is positive. Clearly, when the minimum distance d between the corners is higher, the overall reduction in the number of candidates also increases. A larger value for d implies that more number of corner candidates need to be processed to get the required number of N corners. In this scenario, the proposed mask-based NMS strategy is able to avoid the processing of more number of candidates that are already in the vicinity of corners chosen in earlier iterations. Similarly when the number of corners to be detected, N, is increased, for many images, this results in higher reduction of the number of corner candidates when the proposed mask-based NMS strategy is applied.

The performance improvement achieved due to the savings in computations with the mask-based NMS, is demonstrated by running the algorithms on the Nios-II embedded platform (as described in Appendix B). Figure 5.6 shows the results in terms of the relative speedup in execution time (ψ_{NMS}[1]) of the automated thresholding with pruning technique (AT) when the mask-based NMS is used instead of the conventional NMS, and is computed as:

$$\psi_{NMS} = \frac{t_{AT+convNMS} - t_{AT+maskNMS}}{t_{AT+convNMS}} \tag{5.5}$$

[1]It is to be noted that the speedup in execution time (denoted as ψ), reported earlier in this thesis for pruning and automated thresholding techniques, is with respect to the conventional Shi-Tomasi and Harris corner detectors as is shown in Eq. 3.16. This relative speedup ψ_{NMS}, in contrast, is with respect to the automated thresholding technique itself, when the conventional NMS strategy is used, as is shown in Eq. 5.5.

In Fig. 5.6, the execution time of the AT + convNMS method is shown as bar charts for varying minimum distance d values for NMS and number of corners N. It is clear that the overall execution time increases with the minimum distance, as more number of candidates are suppressed with a larger d. Also, the execution time for $N = 300$ is smaller than that for $N = 1000$. The relative speedup in execution time (ψ_{NMS}) is shown overlaid as pointers in Fig. 5.6. The mask-based NMS (AT+maskNMS) for a minimum distance of $d = 15$ pixels, leads to a relative speedup (ψ_{NMS}) of 14% and 42% with 300 and 1000 Shi-Tomasi corners respectively. For Harris, a speedup (ψ_{NMS}) of 11% and 47% is achieved for 300 and 1000 corners respectively. Also, as seen in Fig. 5.6 (a),(c) the relative speedup (ψ_{NMS}) is higher when the minimum distance d for non-maximal suppression; this is similar to the analysis in Table 5.4.

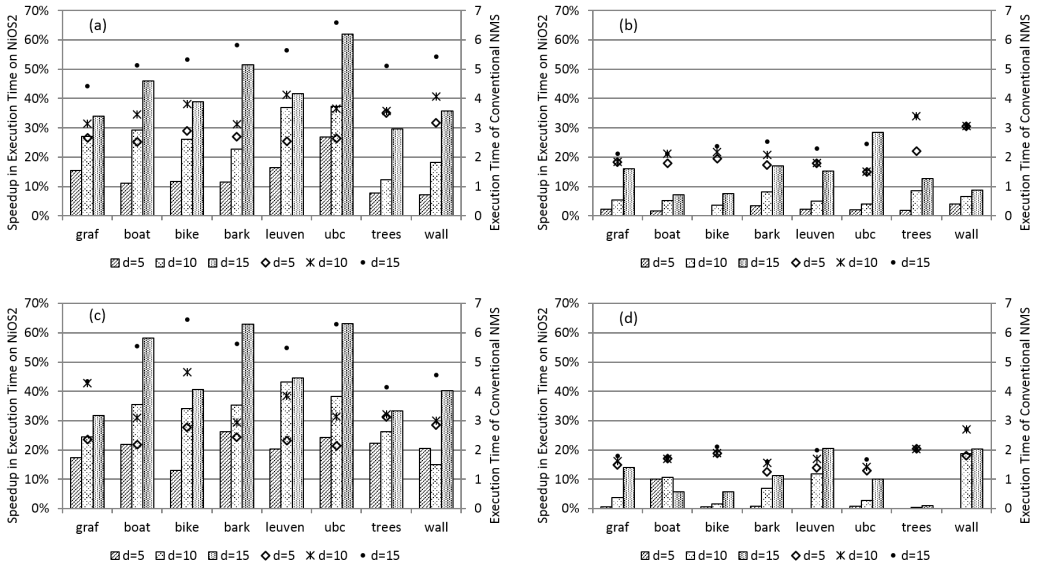

Figure 5.6: Evaluation results: Relative speedup ψ_{NMS} achieved by the proposed mask-based NMS strategy compared to the conventional NMS in the iterative pruning based Shi-Tomasi/Harris corner detectors (a) Shi-Tomasi $N = 1000$ (b) Shi-Tomasi $N = 300$ (c) Harris $N = 1000$ (d) Harris $N = 300$

It is to be noted that applying the automated thresholding with conventional NMS (AT + convNMS) already led to a speedup in computation time compared to the conventional detectors as reported in Section 5.2.3. By replacing the conventional NMS with a mask-based NMS in the automated thresholding further reduction

in complexity has been achieved as shown in this section, especially when larger number of candidates need to be processed in the lower quality corner measure bins.

Summary

In this chapter, the iterative thresholding method for automating the threshold selection for corner detection, presented in Chapter 4, is combined with the low-complexity pruning scheme introduced earlier in Chapter 3, in order to substantially reduce the complexity of corner detection with Shi-Tomasi and Harris algorithms, by adapting the computations to the nature of image content [3, 4]. This is enabled by employing the iterative thresholding with the lower complexity pruning measure instead of the original corner measure. A novel bin-based mechanism has been proposed to collect corner candidates released, such that all the potential corner candidates within a given range of corner measure values have been collected before the final corners are selected. This ensures that the pruning process that extracts the corner candidates has not left out any potential corners. In addition, the issue of a surge of candidates in lower ranges of corner measure has been mitigated systematically. First, the mask-based non-maximal suppression scheme removes corner candidates in the vicinity of already chosen corners from further consideration. In addition, a heuristic measure of the yield of corners from each bin is used to adapt the threshold steps for the subsequent release of candidates.

Evaluations show an average feature match of 98% with the baseline Shi-Tomasi and Harris detectors demonstrating the effectiveness of the bin mechanism in collecting all the potential corner candidates. In addition, an average speedup in execution time of 67% with Shi-Tomasi and 51% with Harris algorithms has been shown with the evaluations on the embedded Nios-II platform. In effect, by the controlled release of the corner candidates that undergo the complex corner measure, the proposed method adapts the computations needed for corner detection to the image content resulting in low complexity. This makes the proposed scheme

suitable for low-resource platforms that need to perform corner detection on a wide range of image content such as the surveillance UAVs with minimal human intervention.

6

Low-Complexity Global Motion Estimation with Sparse Features

Introduction

Estimating the motion of the camera by employing global motion estimation (GME) is a primary step for processing videos captured by on-board cameras on UAVs. However, as described in Chapter 2, existing methods for GME have high computational complexity and are therefore not suitable for real-time and on-board processing on low-complexity platforms. In Chapters 3– 5, the complexity of the corner detection step has been reduced by adapting, the computations for detecting corner regions, to the varying image content seen in aerial videos. In this chapter, a novel low complexity GME method is proposed that exploits

the characteristics of aerial videos to adapt the computations needed for GME with the complexity of the scene, by employing the minimum number of sparse well-distributed features.

As has been described in Section 2.4, GME can be performed using direct methods that operate on all the pixels in the frame for estimation. Although they provide high accuracy, feature-based methods have been preferred for their lower complexity and robustness to outliers. As shown in Fig. 2.6, the feature-based GME pipeline first detects and tracks features in a pair of frames to compute the feature correspondences. The robust estimator then fits a global motion model to these correspondences to compute the global motion between the frames. As videos have small baselines, a feature tracking step - with corner detectors such as Harris, Shi-Tomasi or FAST [52] combined with a tracking algorithm such as KLT [54] - is preferred as in [39], to a more computationally complex feature descriptor-matching step - using SIFT [55] or SURF [56].

Although the feature-based GME methods have lower computational complexity compared to the direct methods (as they operate only on features instead of all the pixels for GME), the number of features used for the estimation still has a substantial impact on the computational complexity of GME. Table 6.1 shows the average execution time for the feature detection, tracking and estimation steps in the GME for a typical aerial video sequence in [39] for various number of features [1]. Clearly, the number of features has a direct impact on the execution time of the feature tracking step. At 3000 features, as is used in [39], the execution time of the feature tracking step dominates the GME execution time.

The background PSNR[2] (peak-signal-to-noise-ratio) that quantifies the accuracy of the GME remains similar for all the densities of features considered in Table 6.1, showing that even a low density of features can achieve reasonable quality of GME

[1]The OpenCV v2.4.9 implementation of Shi-Tomasi feature detector and KLT tracker was used. The multi-threading in OpenCV was turned off to simulate a low-resource embedded platform.

[2]Background PSNR has been computed by removing the moving objects from the frames and then calculating the PSNR of the successive frames after GME as described in Eq. 3.17 in Section 3.5.4

Number of Features	$PSNR$ (dB)	T_{detect} (ms)	T_{track} (ms)	$T_{estimate}$ (ms)
500	38.2	265	**208**	3
1000	38.2	254	**371**	4
2000	38.1	262	**689**	6
3000	38.2	269	**1042**	8

Table 6.1: Accuracy (PSNR) and execution time (T_{detect}: time for feature detection, T_{track}: time for feature tracking, $T_{estimate}$: time for robust estimation) of feature-based GME for various densities of features for *TNT Aerial 350m* video sequence

for most of this video. This motivates us to question the choice of the number of features to be used for GME in order to substantially reduce the computations.

As described in Section 2.4, existing methods to achieve low-complexity GME have been proposed for direct GME methods. They use either lower-precision images [98] or a subset of the image pixels [97, 92]. There is also substantial work in literature [112, 113, 114] to reduce the complexity of the robust parameter estimation algorithm, Random Sample Consensus (RanSaC). However, all these methods still assume that a large number of feature correspondences is available, and therefore incur the associated computation overhead of tracking for a large feature set.

In this chapter, a systematic selection of features is proposed such that a sparser feature set can be employed by removing redundancies. In this context, spatial distribution has been addressed as an important criterion for the usefulness of feature detectors. In [111, 127] a metric to measure the coverage of the feature detector output is provided. Genetic algorithms are used to improve the coverage of feature detectors in [128]. All these approaches measure and improve coverage of the corner detectors for *any* end application. In contrast, in this chapter, the spatial distribution of the feature set and its coverage of the frame, is considered in the context of reducing the number of features needed for global motion estimation (GME). Specifically, during the feature selection, the usefulness of a feature is determined by not just the feature quality as corner detectors do, but also its location in the frame.

It is proposed that the number of features should be *adaptive* to the scene content and camera motion, such that when scene conditions are conducive, the GME can be performed with sparse features and only when needed, higher density of features is employed. To this end, a low complexity GME method, referred to as sparse-GME, is proposed that uses spatial distribution constraints to carefully select features, monitors the performance of the sparse features in the robust estimation to detect failures and ramps up the density of features only when needed.

The rest of the chapter is organized as follows. In Section 6.2, the feature-based GME pipeline is analysed to identify the assumptions that lead to a requirement of a high density of features for GME. In Section 6.3, the proposed sparse-GME method is presented. Section 6.4 discusses the computation cost of sparse-GME. The proposed method is evaluated with baseline algorithms in Section 6.5. The chapter concludes with Section 6.6.

Feature-based Global Motion Estimation

In this section, the feature-based GME is considered to analyse the need for a high density of features for the GME.

Robust Estimation

As described in Section 2.4, the goal of the GME algorithm is to determine the parameters of a global (camera) motion model that defines the camera motion between two frames. The 2D projective motion model, or homography [93, 94] is commonly used to represent the global motion in aerial videos [32].

As seen in Eq. 2.4, for successive video frames I and J, the displacement of every pixel at (x, y) in the current frame I to the new location (u, v) in the next frame

J can be represented using a homography matrix H as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6.1}$$

Parameter estimation involves computing the parameter values of the homography matrix H. Let $X^I = (x, y)$ be the feature in frame I, and let its corresponding location in frame J be $X^J = (x', y')$. The correspondence (X^I, X^J) is determined by a feature tracking/matching algorithm. If H represents the global motion of each feature correspondence, then from Eq. 6.1, a pair of equations for each correspondence can be derived as follows:

$$\begin{aligned} x'_i &= h_{11}x_i + h_{12}y_i + h_{13} - h_{31}x_i x'_i - h_{32}y_i x'_i \\ y'_i &= h_{21}x_i + h_{22}y_i + h_{23} - h_{31}x_i y'_i - h_{32}y_i y'_i \end{aligned} \tag{6.2}$$

Since one correspondence results in a pair of equations, in order to solve for 8 unknowns in H, at least 4 correspondences are needed. However, the feature correspondences could be noisy, due to localisation and tracking errors, or belong to a moving object, in which case, they cannot be used.

Noisy correspondences can be dealt with by using a large number N of correspondences and solving an over-determined system of equations as shown below:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ & & & & \vdots & & & \\ x_N & y_N & 1 & 0 & 0 & 0 & -x_Nx'_N & -y_Ny'_N \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ & & & & \vdots & & & \\ 0 & 0 & 0 & x_N & y_N & 1 & -x_Ny'_N & -y_Ny'_N \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x'_1 \\ \vdots \\ x'_N \\ y'_1 \\ \vdots \\ y'_N \end{bmatrix} \tag{6.3}$$

The set of N feature correspondences (X^I, X^J) are used to solve for the parameters of H in a least-squares manner where the error E, between the measured correspondence (x'_i, y'_i) and the estimated correspondence (u_i, v_i), is minimized:

$$E = \sum_{i}^{N} (x'_i - u_i)^2 + (y'_i - v_i)^2 \tag{6.4}$$

As mentioned earlier, the feature correspondence can also belong to a moving object and in this case, it cannot be used for estimating the global motion, as the feature moves independent of the global motion. These are the *outliers* in the parameter estimation process.

The classical least-squares estimation, described thus far, fails in the presence of such outliers, and *robust* estimation techniques are employed to deal with such cases. Figure 6.1 shows the problem of fitting a straight line *model* to *data* that contains a large number of outliers. In the context of global motion estimation, the *data* corresponds to the feature correspondences and the *model* is homography (in place of the straight line) and the problem is to *robustly* fit the homography model to this data. In Fig. 6.1 (a), the data points closest to the line represent feature correspondences adhering to the global motion. The data points further away are either inaccurate (noisy) background feature correspondences or, belong to moving objects, and are therefore outliers.

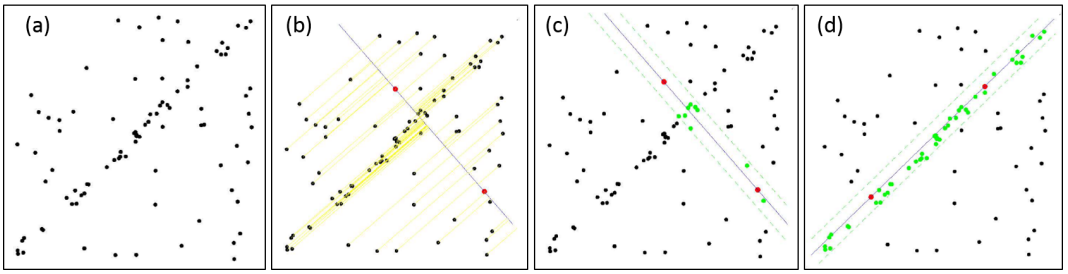

Figure 6.1: Illustration of Random Sample Consensus (RanSaC) algorithm [129] (a) Data points with noise (b) Compute error for a hypothesis set (c) Determine the corresponding consensus set (d) Find and report the largest consensus set after several iterations

The Random Sample Consensus (RanSaC) [96] is a classic robust estimation algorithm. It is an iterative hypothesize-and-test algorithm. It selects a random

sample set and estimates the model for this sample. It then computes the error, also known as reprojection error, of all data points to this model. By applying a threshold on the reprojection error as in Fig. 6.1 (b)(c), it generates a consensus set containing inliers. This process is repeated iteratively, until the probability of finding a consensus set larger than what has been found is very less. The largest consensus set is reported as the *inliers*, as shown in Fig. 6.1 (d).

With a dense feature set, it is expected that there are sufficient feature correspondences adhering to the global motion (in other words, belonging to the background) and they become *inliers* during RanSaC, thereby making all features that belong to the foreground or are too noisy, as *outliers*.

Density of Feature Correspondences

In practice, a dense set of feature correspondences is used for the estimation for the following reasons:

1. *Spatial Distribution*: If the sparse set of features are not well spread out in the frame, then they form a *degenerate set*, and cannot estimate the homography parameters. Spatial distribution is therefore critical for estimation. Allowing a large number of features to be selected, ensures that there is representation for every region in the image during estimation. However, as [111, 128, 127] show, depending on the feature detector, the spatial distribution may vary significantly, and having a high density does not necessarily guarantee good spatial distribution.

2. *Background features*: For a feature correspondence to be useful for estimation, its motion needs to adhere to global motion, i.e. belong to the background in the scene. In the presence of moving objects (foreground), a high density of features ensures that a sufficient number of background feature correspondences is still available for the robust estimation, to declare the foreground features as outliers and estimate global motion accurately.

3. *Accuracy of feature correspondences*: Feature tracking is prone to errors due to occlusions, repetitive textures (resulting in features landing on similar looking neighbouring features) and distortions due to rotations and scaling. It is known that KLT cannot handle large distortions [78]. Having a high density of features ensures that at least a small number of accurate feature correspondences are available to estimate the global motion accurately.

Now, consider the case of global motion estimation for aerial videos. The video captured by on-board cameras on UAVs have the following characteristics that are of interest during GME:

1. *Predominantly simple camera motion*: The camera motion experienced by successive frames in aerial surveillance videos is predominantly translational. Large rotations are only experienced when the vehicle is doing bank turns as shown in Fig. 2.1. This implies that the feature tracker can be setup for a smooth translation motion, and be expected to provide highly accurate feature correspondences for most of the video sequence. This in turn implies that the noise in the data used for GME can be expected to be low for most of the video frames in an aerial video.

2. *Small foreground*: In the surveillance application scenario, the background occupies the majority of the scene and moving objects are sparse and present only occasionally. This implies that the outlier ratio can be expected to be fairly small for most of the video frames in the aerial video.

It is clear that, the majority of the frames in the aerial surveillance videos experience simple motions, with very small number of moving objects, resulting in feature correspondences that are both useful (background features) and accurate. Therefore, when the overall noise in the data is expected to be small and the outlier ratio is also small, a dense feature set is not necessary for GME. By a careful selection of features, even a sparse set can achieve high quality GME for most of the video frames. This is confirmed by the results in Table 6.1.

In the next section, a low-complexity GME method is presented that employs minimum number of features for estimation, which is enabled by an evaluation strategy capable of detecting successful estimations followed by a systematic repopulation method to ramp up features in the event of a failure in estimation.

GME with sparse features

The objective of the proposed sparse-GME method is to adapt, the number of features used for GME, to the complexity of the scene. The idea is to stay lean when scene conditions are conducive - i.e. moving objects (foreground) are very sparse, camera motion is predominantly translational and the image has a natural spread of content. When any of these conditions is violated - i.e. features are lost due to the limitations of the tracker or occlusions, or majority of the features fall on moving objects or the features are not naturally spread out in the frame - only then, the density of the features shall be increased.

The sparse-GME starts with a sparse feature set which is well-distributed in the frame. The frame is processed in a block-based manner, extracting n_1 features in each block of a $k * k$ grid (for the evaluations $n_1 = 1$ and $k = 4$, resulting in 16 features has been considered). An *evaluation* method is proposed to determine if the GME with the sparse feature set is successful or not. This is followed by a systematic *repopulation* strategy that injects additional features to improve the accuracy of the GME when the current estimation has failed. These steps are described in detail next.

Evaluation of estimation

A simple low-complexity strategy is proposed to evaluate the current estimation, which can be used to determine whether additional features are needed to complete the GME.

Figure. 6.2 visualizes the various outcome scenarios in parameter estimation with sparse features by illustrating the feature correspondences in the model parameter space. Figure. 6.2 (a) is an example of a successful estimation: the inlier feature correspondences (black circles) are close to the model and the outliers (empty circles) are distinctly far from the inliers set. In this case, RanSaC is able to correctly classify the inliers and outliers and the model fitted by the inliers represents global motion.

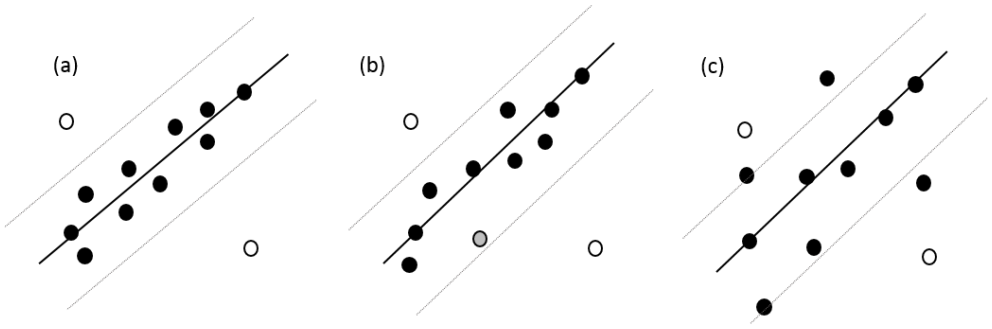

Figure 6.2: RanSaC failure cases with sparse features: (a) Successful estimation with clear classification of inliers and outliers (b) Failed estimation with a single feature correspondence (grey) skewing the estimation (c) Failed estimation with very poor agreement between the inliers

Failure in estimation, i.e. the estimated model is far from the actual global motion experienced, is due to the following reasons:

1. Inaccurate correspondences or foreground features participate and *skew* the estimation. This means that the inlier/outlier classification itself is faulty. Two possible scenarios arise:

   (a) As shown in Fig. 6.2 (b), majority of the correspondences are useful, however a very small number of inaccurate correspondences or features on moving objects (represented by grey circles) are responsible for erroneous estimation. Removing or replacing these feature correspondences could still lead to correct estimation.

   (b) As shown in Fig. 6.2 (c), majority of the correspondences are not agreeing well to the model. Therefore a higher density of features is needed

so that the camera motion can be captured with sufficient number of features representing the majority motion.

2. Inlier/outlier classification is correct, however the inliers form a *degenerate* set due to insufficient coverage of the frame.

The aim of the evaluation strategy is to distinguish between all these scenarios in order to complete the GME by employing the *minimum* number of features required. The evaluation strategy uses two criteria: (1) Inlier agreement (2) Spatial distribution constraint, which are described next.

Inlier Agreement

In order to evaluate the accuracy of the estimation, first it is determined if the set of feature correspondences agree with the estimated model well. In RanSaC, the reprojection threshold, used to separate the inliers from the outliers, determines how tightly the inlier set agrees with the estimated model. As shown in Fig. 6.3, the RanSaC is invoked with two reprojection thresholds T_1 and T_2 called *2-RanSaC* such that $T_2 < T_1$. If the inlier/outlier classification remains the same for both T_1 and T_2, then the inliers can be relied upon for further evaluation. The correspondences that are inliers for both the estimations are considered as *high-confidence* inliers. (The thresholds of $T_1 = 3$ and $T_2 = 1$ are used for the experiments.)

Next, it is ensured that there is a high degree of agreement to the model in the inlier set. The location error, LE of the inliers is computed and checked if it is below a threshold as given below:

$$LE = |H_{est}(x_i, y_i) - tracker(x_i, y_i)| \tag{6.5}$$
$$LE < T_{LE}$$

Where for feature at (x_i, y_i), $H_{est}(x_i, y_i)$ represents the GM estimated location by applying the homography H_{est} and $tracker(x_i, y_i)$ represents the tracked location.

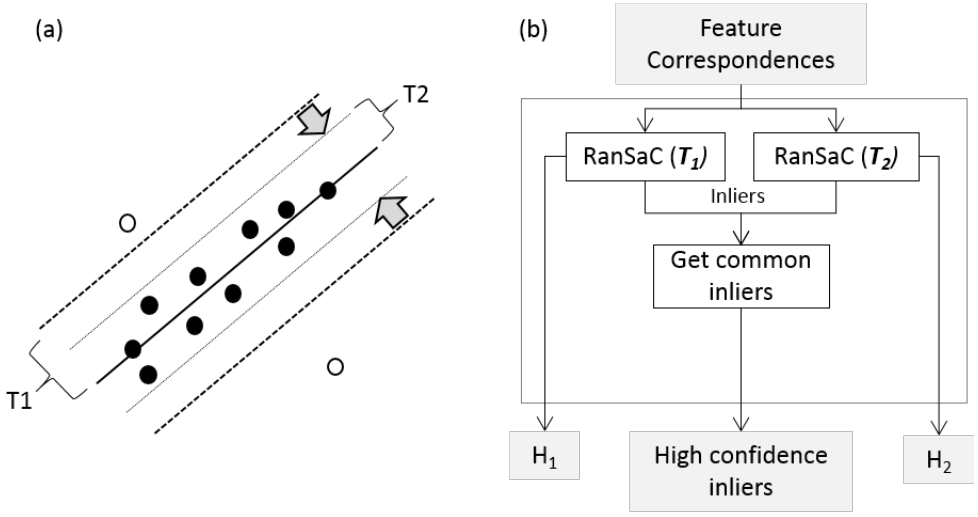

Figure 6.3: RanSaC with two reprojection thresholds (*2-RanSaC*)

All the *high-confidence inliers* need to have a low location error. For the evaluations, $T_{LE} = 0.5$ was used. Successful estimation as in Fig. 6.2 (a) should meet the above inlier agreement criterion.

Spatial Distribution Constraint

An additional check for the spatial distribution is required to rule out the possibility of a *degenerate* set - i.e. a situation where a there is a small number of inliers with high degree of agreement but not representing global motion due to lack of coverage of the frame. It is required that the *high-confidence inliers* cover at least a majority of the image content (for the evaluations, at least 75% of the $k * k$ grid was used).

The evaluation of the estimation is performed as follows:

- The estimation is considered successful as in Fig. 6.2 (a) if there are sufficient number of *high-confidence inliers* satisfying the spatial distribution constraint, with low location errors.

- Failure due to a small number of incorrect feature correspondences *skewing* the estimation as in Fig. 6.2 (b) is detected by the presence of *shaky* correspondences, i.e. features that are inliers for RanSaC with threshold T_1 but become outliers for the tighter reprojection threshold T_2.
- For the case of Fig. 6.2 (c) both the checks for inlier agreement as well as spatial distribution constraint fail.

Finally, for a successful estimation, the homography model corresponding to the estimation with a better inlier agreement is reported. First, the estimation with the tighter threshold T_2 is considered. If it fails the inlier agreement, then the estimation with T_1 is considered. When either of them are not "good", i.e. they fail the inlier agreement check, the next repopulation option is considered.

Figure 6.4 illustrates the flow of the evaluation steps described thus far, for the GME with the sparse set. A sparse set of feature correspondences are selected in phase 1 (P1). RanSaC is applied with two thresholds T_1 and T_2 as shown in Fig. 6.3 in the step: *2-RanSaC*, providing the estimated homographies H_1 and H_2 respectively and the *high-confidence inliers*. Next, the inlier agreement is checked for the *high-confidence inliers* with H_1 and H_2. It is also checked if *enough high-confidence inliers* meet the spatial distribution constraint. If both these checks are passed, the model among H_1 and H_2 that had better inlier agreement, is reported. When either of these checks fails, a systematic repopulation as shown in phases P2-P4 is undertaken, as described in the next section.

Repopulation

Repopulation with additional features is done in a phased manner to improve the accuracy of the GME when a failure is detected by the above evaluation method. Phase 1 (P1) represents the first pass of the estimation with a sparse well-distributed feature set with n_1 features per block in a $k * k$ grid ($n_1 = 1$ and $k = 4$ was used). When the estimation in P1 fails, subsequent phases of repopulation are performed, as detailed below:

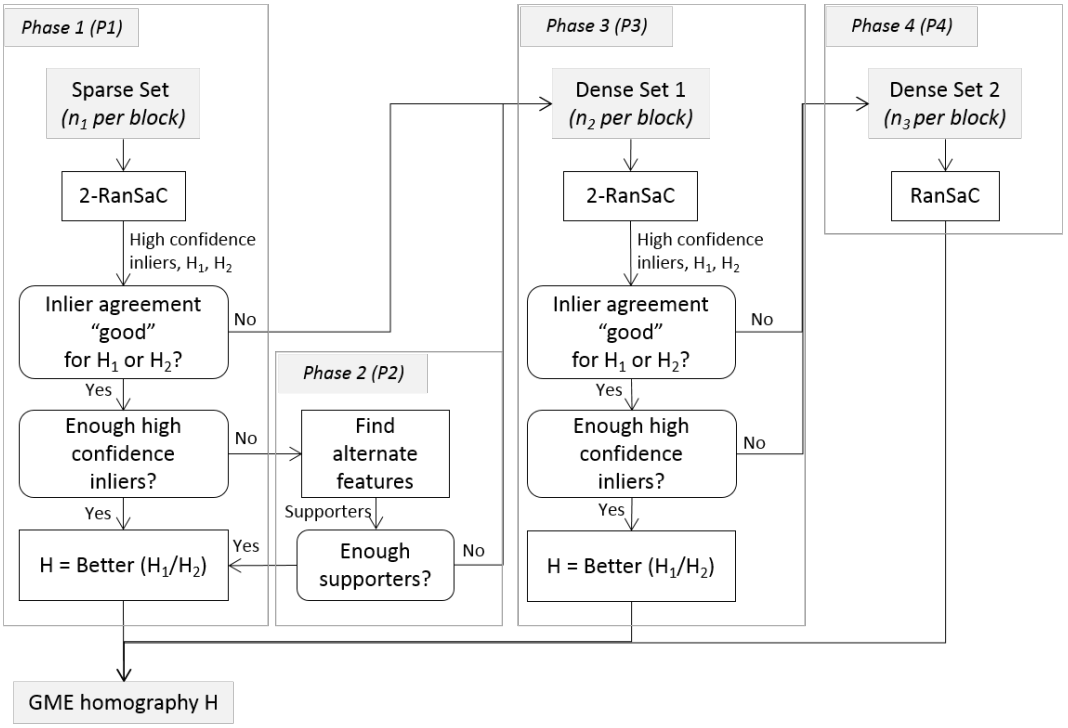

Figure 6.4: Flow of proposed sparse-GME method

1. *Selective injection of alternate features*: When the feature correspondences are lost during estimation (*2-RanSaC* step) - as either outliers or *shaky* features, the spatial distribution check fails, as there are not sufficient well-spread *high-confidence* inliers. This leads to a selective injection of features represented as phase P2 in Fig. 6.4.

   It is to be noted that the check for sufficient *high-confidence inliers* is performed after the check for inlier agreement and therefore the *high-confidence inliers* are in high degree of agreement. Additional features are needed only to "support" the current estimation. Therefore, in the blocks that have lost representation, additional feature correspondences are generated. For each such alternate feature, it is determined if it supports the current estimation by computing its location error as in Eq. 6.5 and checking if it is small. If sufficient "supporters" with these alternate features are found, the corresponding homography H is reported. This process is undertaken for H_2 first, if it passes the inlier agreement check and if not, then H_1 is considered.

2. *Uniform increase in density of features*: If neither of the estimates H_1 and

H_2 pass the inlier agreement check (i.e. have low location errors for all their high-confidence inliers), then it indicates that the confidence in the estimation by the sparse set in P1 is low. A higher density of features of n_2 is then applied and the inlier agreement is determined as shown in phase P3. If this fails as well, then the estimation is performed with the worst case dense set of features n_3 in phase P4. For the evaluations, $n_2 = 5$ and $n_3 = 10$ is used.

The flow of the proposed sparse-GME through all the phases, as described in this section, is shown in Fig. 6.4.

Cost Analysis

The proposed sparse-GME method works with *minimum* number of features for GME compared to conventional approaches that employ a dense set. This leads to savings in computations in the feature correspondence step. However, in order to determine whether the GME was successful with this sparse set of correspondences, additional computations are needed. The motivation for the proposed method is that assuming most of the frames in a video undergo simple global motion with a very small number of moving objects, the GME has a high chance of being successful with a sparse but well-covered feature set. Therefore, the only overhead for each frame is in checking if GME was successful or not and since these computations use a very sparse set of features, this overhead is substantially less compared to the computation cost of GME with a very dense set of features.

In this section, the computation cost of the proposed method is considered by looking at the cost of each of the phases (P1, P2, P3, P4) of iterative repopulation, employed by the proposed method as shown in Fig. 6.4. It is expected that most of the frames in a video sequence go through only phase P1, and only when GME fails at P1 are phases P2, P3 and P4 taken. The complexity of each individual step in these phases is discussed below:

1. *2-RanSaC*: This step includes computations for generating the feature correspondence - i.e. feature detection and tracking and the robust estimation using these correspondences. In a conventional GME, tracking with KLT and estimation with RanSaC is performed on a dense feature set.

   - KLT tracking [77]: KLT has two major steps - (1) Building an image pyramid for each frame (2) Finding the corresponding location for each feature. Step 1 is needed irrespective of the number of features. When the dense feature set is replaced by a systematically selected sparse feature set, the computations in Step 2 are reduced, proportional to the reduction in the number of features.
   - RanSaC: The time complexity t of RanSaC [129] is given by:

   $$t = k(t_M + \bar{m_s}N) \tag{6.6}$$

   Where t_M is the time to compute a single model, $\bar{m_s}$ is the average number of models per sample, k is the number of samples drawn and N is the number of data points. When the sparse feature set is used instead of a dense set, the number of data points N is reduced.

   It is clear that the computation cost of *2-RanSaC* is directly proportional to the number of data points, i.e. feature correspondences. Therefore depending on which phase uses this step - whether the sparse set in P1 or the dense set in P3 are used - the complexity is proportional to the size of the corresponding feature set.

2. *Inlier agreement check*: This involves computing the location error for the set of feature correspondences as in Eq. 6.5 and therefore is directly proportional to the number of feature correspondences.

3. *Find alternate features*: For each additional feature considered, this step finds the feature correspondence using KLT tracking algorithm and then computes the location error as in Eq. 6.5. The number of such additional features considered determines the computation cost of this step.

The steps that check for enough high-confidence inliers or supporters are simple checks with a predefined threshold. In summary, the computation cost of phases P1 and P2 are substantially less, compared to that of P3 and P4, as the number of data points used in these phases is much lesser. The more complex P3 and P4 phases are used only when the GME fails and as will be seen in Section 6.5, the number of frames that use these phases is very small in a video sequence thereby substantially reducing the overall GME complexity.

Performance Evaluations

In this section, the performance of the proposed sparse-GME method is evaluated on a variety of aerial video datasets. First, the experimental setup is described: specifically, the video datasets and the evaluation criteria. Then, the performance results of the proposed method are shown.

Evaluation Setup

Aerial Video Datasets: The aerial video datasets *TNT Aerial* [39] and *CLIF* [130] as shown in Fig. 6.5 have been chosen for the evaluation. The *TNT Aerial* sequences were recorded on a UAV with a downward looking moving camera, with full HDTV resolution (1920x1080, 30 fps). Radial distortion compensation was applied manually and the resulting cropped frames are of resolution 1904x1072. The names of the video sequences represent the height of the camera - for e.g. *TNT-350m* was recorded by the UAV at a height of 350m.

Simulated frames: As the video datasets above do not contain challenging scenarios with camera motion and moving objects, simulated data is also generated to evaluate the proposed method, when the camera motion is drastic and/or number of objects is large. The images in Fig. 6.6 are used. In Fig. 6.7, the applied simulations are illustrated. The simulations involve:

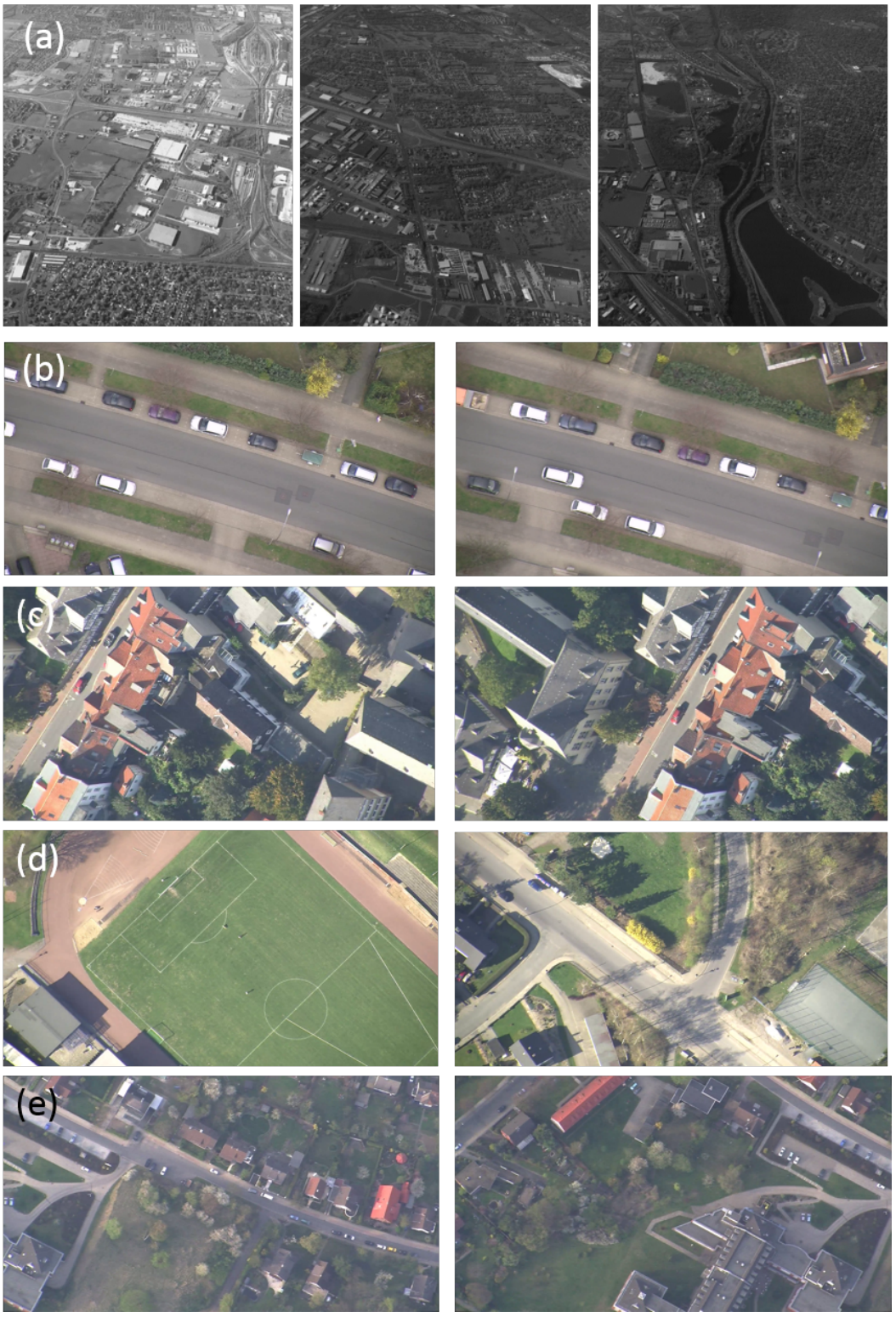

Figure 6.5: Aerial video datasets for evaluation (a) *CLIF* (b) *TNT-350m* (c) *TNT-500m* (d) *TNT-1000m* (e) *TNT-1500m*

- *Moving objects*: Figure. 6.7 (a) shows the original image (current frame) with the simulated objects, black square patches, overlaid. The objects are positioned in a spread out manner such that only one moving object appears in one block in a 4x4 grid. This is to ensure that the worst case scenario for GME in the presence of moving objects is simulated: i.e. the feature chosen within a block falls on the moving object and therefore is not useful for GME. Figure. 6.7 (b) shows the simulated next frame with both the object and camera motion applied to the current frame. Random local motion is applied to the simulated objects within ± 10 pixels.

- *Camera motion*: The simulated frames contain 0-10 moving objects. The camera motion is simulated as 1-10 degrees of in-plane rotation and scale factor of 1-0.8x.

The feature correspondences shown as optical flow vectors in Fig. 6.7 (c) show how the object motion differs from the background motion. All the frames have a resolution of 352x288.

Figure 6.6: Images used for the simulated data for evaluation of the proposed sparse-GME method

Baseline: A feature-based GME (as shown in Fig. 2.6) with dense features called dense-GME is used as the baseline, with the following parameter values for the various algorithms:

- *Shi-Tomasi feature detector*: Threshold for quality $T_c = 0.001$, Minimum distance for non-maximal suppression $d = 10$, maximum no. of features $N = 1000$ (as described in Section 3.2)

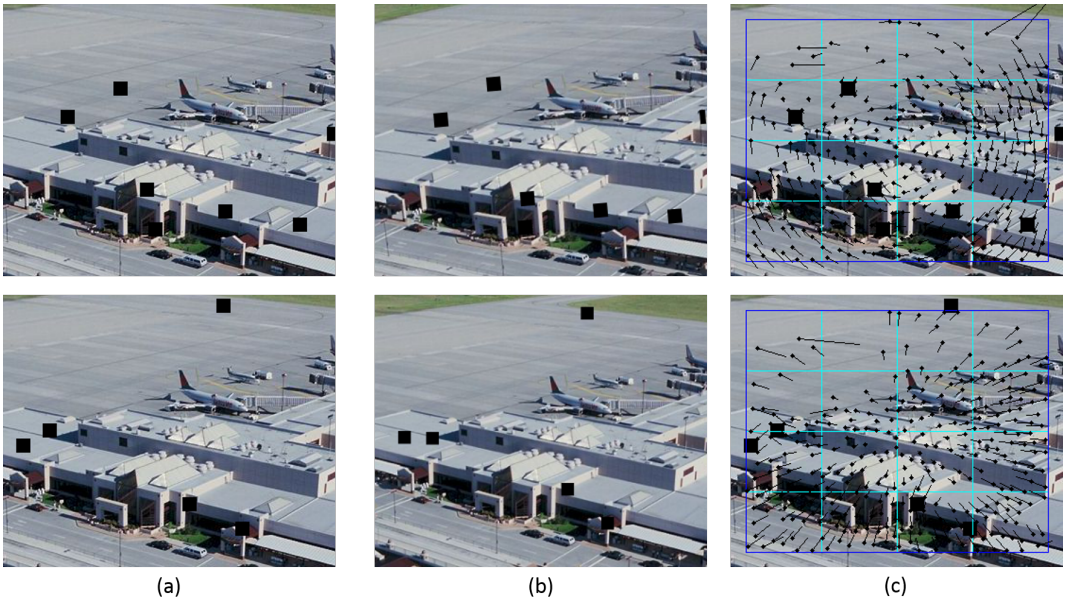

Figure 6.7: Simulated Frames: (a) Frame1 (b) Frame 2 (c) Optical flow generated by the KLT tracker on Frame 1; Row 1 shows 7 moving objects with in-plane rotation of 5 degrees; Row 2 shows 5 moving objects with scale factor of 0.9

- *KLT feature tracker*: Window size $W = 31$, Maximum no. of iterations $k = 20$, Number of pyramid levels $L_{max} = 2$ (Simulated Data) and $L_{max} = 4$ (Video Data), Minimum eigen threshold $\lambda_{th} = 0.0001$
- *RanSaC estimation*: Reprojection threshold $T_{reproj} = 3.0$

In addition, a naive approach of only using sparse features is also compared so that the significance of the repopulation stage in the proposed sparse-GME method can be demonstrated. This involves the sparse feature set used in the phase P1 of the proposed sparse-GME method, without any of the subsequent *evaluation* and *repopulation* steps. This approach is referred to as 4x4x1.

For the proposed sparse-GME method, a 4x4 grid ($k = 4$) as described in Fig. 6.4 is used for all feature selections. For the check for inlier agreement in P1, the threshold for LE, $T_{LE} = 0.5$ is used and *all high confidence inliers* need to have a location error lower than this threshold. The same threshold for location error is used in P2, while selecting alternate features. For the similar check in P3, 80% of *high confidence inliers* are required to have a location error lower than

the $T_{LE} = 0.5$. For the check on enough number of *high confidence inliers* and *supporters*, it is required that 75% of the total number of feature correspondences used for the estimation are *high confidence inliers* or *supporters*. The proposed sparse-GME method, as shown in Fig. 6.4, was implemented in OpenCV (v2.9) using standard libraries for Shi-Tomasi corner detection, KLT feature tracking and RanSaC algorithms.

Evaluation Criteria: To evaluate the GME accuracy, the background peak signal-to-noise ratio (PSNR) is calculated between the actual frame F_1 and the estimated frame $F_1' = H(F_2)$ where H is the homography reported by the GME as in [98].

The moving objects (foreground) are excluded from the PSNR computation so that the differences are related to only GME errors and non-foreground changes between the frames. The PSNR is computed similar to 3.17 as shown:

$$PSNR = 10 \times \log_{10}\left(\frac{255^2}{MSE}\right) \tag{6.7}$$

Here MSE is the mean of the squared-error of the pixel intensities between the actual frame F_1 and estimated frame F_1' after the foreground has been removed.

For the video dataset, the difference in the background PSNR when the proposed sparse-GME method is employed, in comparison to the baseline dense-GME method is represented as $\delta PSNR$ and computed as:

$$\delta PSNR = PSNR_{sparse} - PSNR_{dense} \tag{6.8}$$

The number of frames which have a PSNR difference smaller than $T_{PSNR} = 1$ dB between the proposed and baseline methods is also counted and reported as

successful cases of GME, ϕ in Table 6.2, computed as:

$$\phi_i = \begin{cases} 1 & \delta PSNR_i < T_{PSNR} \\ 0 & Otherwise \end{cases}$$
$$\phi = \frac{\sum_{i=1}^{n_F} \phi_i}{n_F} \tag{6.9}$$

For the simulated frames, the actual background PSNR values are reported.

The efficiency of the proposed method is evaluated by considering the repopulation phase taken by GME for that frame pair. As seen in Section 6.4, phase P1 has the lowest computation cost. For the video dataset, the % of frames that take each phase P1-P4 are reported in Table 6.3. For the simulated frames, the % of frame pairs that take each phase P1-P4 for a given camera and object motion are shown.

Performance Results

Aerial Video Datasets: Table 6.2 shows the accuracy evaluation results for the video sequences. The sparse-GME achieves background PSNR similar (within 1 dB) to the baseline dense-GME for 97% and above cases in all the video sequences. This is represented as the % of successful frames, ϕ. The average drop in background PSNR ($\delta PSNR$) when the sparse-GME is used is in the range of only [-0.08,-0.2] dB. This shows that the sparse-GME method is able to achieve comparable accuracy as the dense GME.

Table 6.3 shows that in 94% and above cases for all the video sequences, the sparse feature phase P1 is used by sparse-GME. The dense phases P3 and P4 are chosen only by at most 2.6% of the frames. This shows that the proposed method is effective in detecting conditions conducive for sparse features to be used for GME, and it stays lean in such situations.

Simulated frames: Figure. 6.8 shows the evaluation results for simulated frames with 0-10 moving objects in the frames (with a camera motion of in-plane rotation of 2 degrees). The sparse-GME is able to match the accuracy of the baseline

	% Successful Frames (ϕ)	Difference in PSNR ($\delta PSNR$) dB
TNT 350m	100	-0.1
TNT 750m	97.4	-0.21
TNT 1000m	99.7	-0.14
TNT 1500m	99.6	-0.08
CLIF	99	-0.11

Table 6.2: Accuracy evaluations of sparse-GME with aerial video datasets

	Repopulation Phases			
	P1 (%)	P2 (%)	P3 (%)	P4 (%)
TNT 350m	94.9	5.1	0	0
TNT 750m	94.9	2.6	2.6	0
TNT 1000m	98.5	0.2	1.4	0
TNT 1500m	95.7	2.4	2.0	0
CLIF	98.3	0.4	1.1	0.2

Table 6.3: Efficiency evaluations of sparse-GME with aerial video datasets

dense-GME even with a large number of moving objects. Clearly, as the number of moving objects increases, the naive sparse feature set in 4x4x1, fails while the sparse-GME successfully detects the failed evaluations and ramps up the required number of features achieving good GME accuracy.

As shown in Fig. 6.8, when the number of moving objects is between 1-3, sparse GME stops at phase P1 for 100% of the frames. As the number of moving objects is increased, the sparse-GME needs to employ the next phases of P2-P4. Note that when the phase P1 fails, selective injection of features by looking for an alternative feature in blocks that contain outliers or *shaky* inliers, i.e. phase P2 results in accurate GME. This phase is able to avoid a uniform increase in density in all blocks when P1 fails.

The proposed method is also evaluated when the inter-frame camera motion is drastic - in-plane rotations of 1-10 degrees and scale changes of 1-0.8x. As the global motions are all simulated, the background PSNR with the ground truth GME parameters is shown in Fig. 6.9 and 6.10. As can be seen, the conventional dense-GME is able to achieve comparable accuracy as that with the ground truth, only for small global motions - rotation angles less than 5 degrees and scale factors

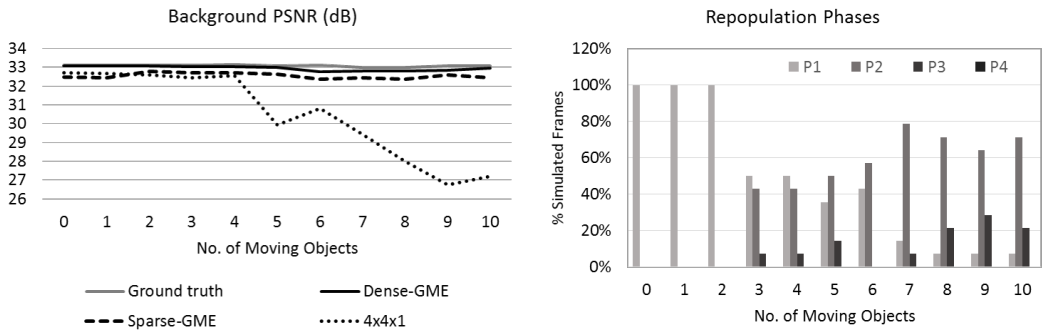

Figure 6.8: Accuracy and efficiency evaluation of sparse-GME with simulated moving objects

less than 0.9x. Beyond this, having a dense set alone is not helpful. This is because the KLT feature tracker is unable to deal with the drastic distortions caused by the rotation and scale changes resulting in inaccurate feature correspondences. Therefore, the proposed method also deteriorates in performance as it is using a subset of the KLT feature correspondences used by the dense-GME. The repopulation phases of P2-P4 are invoked but do not contribute to any improvement in GME accuracy as the newly injected feature correspondences are also inaccurate. In Chapter 7, the robustness of the KLT feature tracker is addressed, and when applied to GME shows a substantial improvement in the performance of the sparse-GME (refer to results in Fig. 7.12).

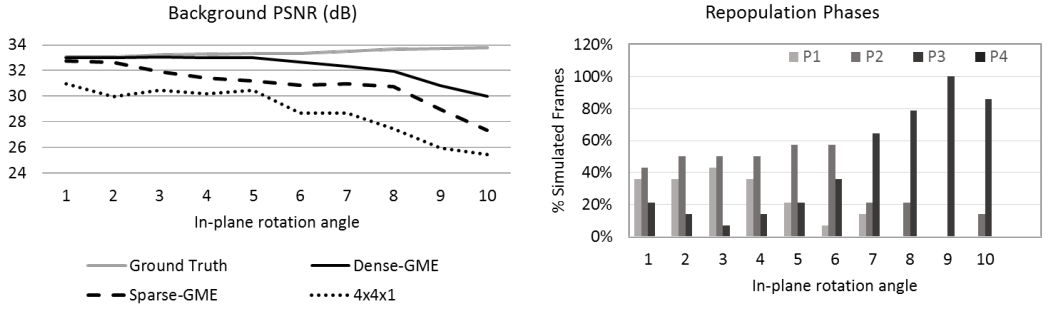

Figure 6.9: Accuracy and efficiency evaluation of sparse-GME with simulated camera motion (in-plane rotation)

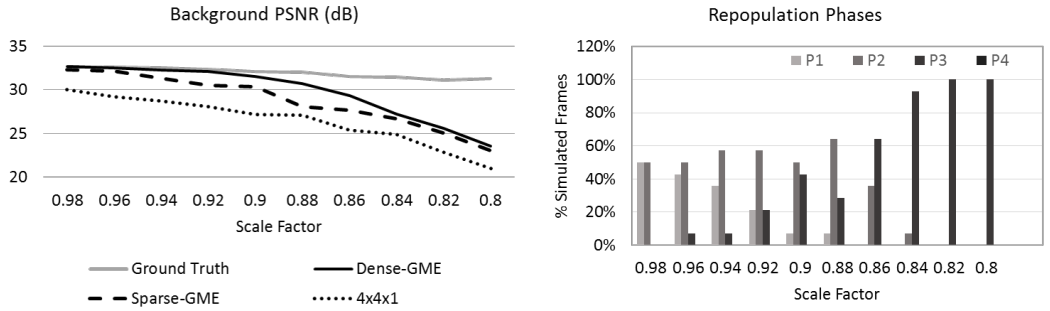

Figure 6.10: Accuracy and efficiency evaluation of sparse-GME with simulated camera motion (scale change)

Summary

In this chapter, a strategy for low-complexity global motion estimation with very sparse feature sets, is proposed that adapts the number of features needed for GME with the complexity of the scene and the camera motion, leading to substantial reduction in computations for GME in aerial surveillance videos. A sparse feature set with sufficient spatial distribution is selected and a computationally simple approach is proposed to determine the quality of GME using this set of features. A systematic repopulation with additional features is performed only if the GME fails with the sparse set. Unlike existing techniques for reducing complexity of the robust estimation that assume a dense feature set is always available, the proposed method detects the need for higher density and only then ramps up the number of features. This is enabled by the rapid evaluation of the GME with the sparse features.

For aerial video sequences, the accuracy of the GME with the proposed method is similar to that of GME with a dense feature set for 97% and above cases, while the computations are limited to GME with a very sparse feature set of around 16 features for around 95% of the frames. On simulated data, it is shown that the proposed method is able to match the accuracy of the dense feature set GME when the number of moving objects increases, by systematically moving to higher densities.

The evaluations with drastic camera motion show that when the tracker is itself unable to handle the motion and results in high noise in the feature correspondences, both the dense feature set and the proposed method suffer. It should be noted that such situations of drastic camera motion are infrequent in aerial videos, and a more robust tracker is needed to improve the performance of GME in these situations.

The proposed method enables low-complexity global motion estimation in aerial surveillance videos by, staying computationally lean when the global motion is simple and manageable, and ramping up the number of features, only when GME fails.

7

Adaptive Windowing for Robust and High-Speed KLT Tracker

Introduction

Feature tracking is an essential step in global motion estimation (GME) as well as many computer vision applications, such as image registration and object tracking that extract higher level information about camera and/or object motion from the local optical flow vectors of each feature. The Kanade-Lucas-Tomasi (KLT) feature tracking algorithm [54] has been extensively used, as it is very effective for small frame-to-frame displacements of features (which is common in video frames), and its low computational complexity enables it to be deployed on resource constrained platforms. In order to handle larger displacements, a multi-resolution approach

is presented in [77], which is based on a pyramidal implementation of KLT. The KLT tracker is used to find the feature correspondences for the Shi-Tomasi/Harris corners that can be used for estimating the global motion for aerial videos as shown in Fig. 2.6.

Despite using a pyramidal implementation to handle large displacements, the window size used by KLT is still set up *large enough* to capture a wide range of displacements (default value in OpenCV is $W = 21$ and Matlab is $W = 31$). However when the frame rate of the video is high leading to small displacement between frames, this setup wastes computations. Also, KLT assumes that the patch (defined by the window size) around the feature undergoes only translation motion. Therefore, in the presence of rotation and scaling, with a large window size, KLT is known to suffer from high inaccuracies [78]. This leads to errors in the global motion estimation (GME) that employs these KLT feature correspondences as shown in Fig. 6.9 and 6.10 in Chapter 6. This is a severe limitation of KLT as drastic frame-to-frame rotation is common in many applications, for example videos captured on-board UAVs that perform bank turns [78].

Several efforts to handle non-translation motion by KLT have been proposed in literature. One common method is to switch to a more complex feature detector-descriptor and employ feature matching [117]; however the higher computational complexity is prohibitive for their use on low-resource platforms. Inertial measurement units on-board the aerial vehicle provide the relative 3D motion of the vehicle and this data is used to arrive at initial estimates for KLT leading to increased robustness under drastic motions [78, 81]. In this chapter, a technique that is not dependent on IMUs is proposed. An affine formulation for KLT [79] employs an affine model instead of the translation model for the motion of the feature patch in the search window, allowing the feature patch to undergo rotation, scaling and skew. However, this is highly compute-intensive and real-time performance is achieved only with GPU implementations [80]. In [82] an uncertainty estimate for the KLT feature correspondence is computed using the intensity and disparity variations within the patch. This is used in an iterative window sampling method that arrives at the optimal window size for each feature. However, this

iterative sampling needs to be performed for each core KLT operation to estimate the displacement and is infeasible for real-time applications.

As KLT is susceptible to errors, evaluating the feature correspondence is critical for ensuring the accuracy of the global motion estimation and other algorithms that rely on this data. A good survey on such online evaluation methods is presented in [84]. Such evaluations associate an error/uncertainty estimate with each track, allowing the applications that use the feature tracks, to detect situations when KLT fails to find the correct correspondence. In [85], the time-reversibility constraint of trackers has been applied to compute a forward-backward error for tracks, and this metric is used to discard potentially erroneous tracks. In [86], theoretical error estimates for KLT tracks is presented, however it is reported that the error estimate computations are so high that they cannot be used for real-time applications.

In this chapter, it is shown that it is necessary for the KLT window size to adapt to the presence of distortions around the feature points in order to increase the quality of the tracks as is confirmed in [82]. A novel adaptive windowing method [5] is proposed for KLT that monitors the KLT *performance* for a given window size as an indicator of successful tracking, and does not require explicit error/uncertainty estimate computations as in existing methods. In particular, it is shown for the first time that the iterations needed to converge within KLT, can be used as a crude indicator of the quality of the feature track, thereby providing a simple and compute-efficient means to arrive at a suitable near-optimal window size. The proposed strategy is applied to each level of the pyramid in the hierarchical implementation of KLT to significantly improve the robustness to large displacements and distortions. The extensive evaluations with simulated as well as annotated tracking datasets show that the proposed strategy significantly outperforms the translation model-based KLT in terms of robustness. In addition, the proposed strategy achieves comparable robustness as the affine model based KLT at 7x run-time speedup.

The chapter is organized as follows. Section 7.2 presents an overview of the KLT algorithm and an analysis of the relationship between the search window size, the motion experienced by the feature and the accuracy of the KLT tracker and motivates the need for adapting the window size of KLT in order to enhance its robustness to distortions. Section 7.3 presents the proposed adaptive window strategy for KLT that can be applied to pyramidal KLT implementation. It is shown that the iterations needed to converge within KLT can be used as a crude indicator of the window size being too small, and this can guide the adaptive strategy to land at a suitable window size. In Section 7.4, using simulated as well as a real annotated tracking dataset [117], it is demonstrated that the proposed adaptive strategy outperforms the conventional fixed-window KLT in terms of robustness, and achieves significantly lower runtime without sacrificing robustness when compared to the well-known affine KLT. Chapter concludes with Section 7.5.

KLT Feature Tracker

The goal of the KLT tracking algorithm is to find the displacement $d = [d_x, d_y]^\top$ for a feature x in two images *I(x)* and *J(x)* such that the residual error $E(d)$ defined in Eq. 7.1 is minimized as shown:

$$E(d) = \sum_{W} [I(x) - J(x+d)]^2 \tag{7.1}$$

The residual $E\,(d)$ is the intensity difference computed for a region of support defined by the search window W around the feature x. Given a starting position for d (initialized to $d = 0$), Eq. 7.1 solves the least squares optimization problem by iteratively modifying d as $d \leftarrow d + \Delta d$, through a Newton-Raphson method, such that $E\,(d)$ is minimized. A detailed derivation of the KLT feature tracker is presented in Appendix A.

Eq. 7.1 assumes a translation model and is sufficient when the motion between the video frames is predominantly translational. In the presence of fast rotation or scale changes, this translation model fails. In order to track patches undergoing

rotation and scaling with KLT, an affine model was proposed in [79]. The residual error of the affine model is defined as in Eq. 7.2:

$$E(d) = \sum_{W} [I(x) - J(Ax + d)]^2 \tag{7.2}$$

Here, A is the affine transformation matrix given by:

$$A = \begin{bmatrix} 1 + d_{xx} & d_{xy} \\ d_{xy} & 1 + d_{yy} \end{bmatrix} \tag{7.3}$$

The six parameters $(d_x, d_y, d_{xx}, d_{yy}, d_{xy}, d_{yx})$ allow rotation, anisotropic scaling, skew and translation. However, this robustness comes with a computation cost trade-off. Therefore, the translation model is used as the basis of the proposed method.

Effect of Search Window Size for Rotation/Scaling

As shown in Fig. 7.1 (a), if a pixel P_1 in the current frame moves to P_2 in the next frame, with a maximum inter-frame displacement d_{max}, it is advisable to setup the search window size $W = 2 * d_{max} + 1$, such that the displacement of the feature is captured within the KLT search window [77]. Figure 7.1 (b) illustrates the translation model assumed by KLT as in Eq. 7.1 in which all the pixels within the search window W around P_1 have the same displacement given by (d_x, d_y). This assumption is valid only when the feature itself undergoes translation motion. As shown in Fig. 7.1 (c) when there is rotation, the displacement of each of the pixels in W varies and this violates the underlying assumption of a translation motion model by KLT.

When the feature undergoes translation motion alone, having a larger window size is preferable because any noise in the intensity patterns within the window is averaged out. However, when the feature undergoes rotation or scaling, then the pixels within the neighbourhood defined by the window, have varying displacement. As we move further away from the centre of this window, these variations increase

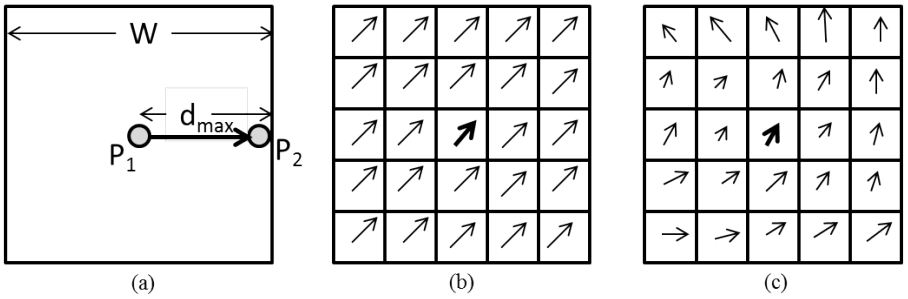

Figure 7.1: Search window size and feature patch displacement (a) Window size W needs to capture maximum displacement d_{max} (b) Displacements of neighbouring pixels for translation (c) Displacements of neighbouring pixels for rotation

– therefore, relying on the farther away pixels results in higher error in the KLT estimates. In such cases, it is to be noted that, typically the centre pixel undergoes a small displacement and therefore, this can still be captured with a small window size without including potentially erroneous estimates from far away pixels.

Hence, for images undergoing rotation and scaling, the window size needs to be *large enough* to capture the displacement of the feature but *small enough* to not invite potentially erroneous estimates from pixels far from the centre of the feature patch. Similar to the conclusions in [82], there exists an optimum window size that results in the most accurate track for KLT for a given feature patch and inter-frame motion.

When the frame undergoes a global rotation or scaling motion, as shown in Fig. 7.2 (b) and (c), then the local displacement and distortion experienced by different features depends on their location in the frame. Figure 7.2 shows the distribution of the displacement reported by KLT for the various features. It is clear that an optimum window size needs to be determined for each individual feature in the presence of global rotation and/or scaling in order to deal with the wide range of displacements and distortion.

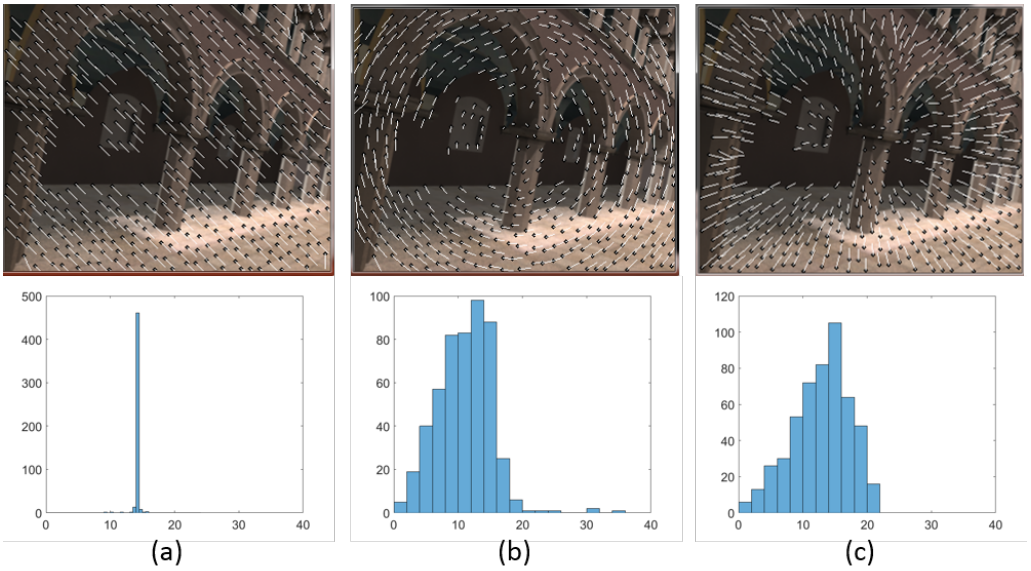

Figure 7.2: Illustration of local displacement in the presence of various global motions. Row 1: Local feature motions for various global motion (a) translation (b) rotation (c) scaling, Row 2: Histogram of the local displacements

Implications of Fixed Search Window Size with Pyramidal KLT

As seen in Section 2.3.2, a pyramidal approach [77] is employed in order to satisfy the linearity assumption made in the core step of KLT (refer to A.5 in Appendix A) which works only if the displacement Δd is very small. As shown in Fig. 2.4, image pyramids with L_m levels are constructed for both the frames I and J. KLT optical flow (d_{L_m}) is computed at the highest pyramid level L_m, which corresponds to the lowest resolution sub-sampled image of the frame. The result is propagated to the next level ($L_m - 1$) as the initial guess for the pixel displacement in this level. This process continues until level L_0 (original frame I_0) is reached. The main advantage of this hierarchical approach is that the displacement d_L to be found at each level, can be kept very small such that the linear approximation in the KLT core step remains valid. The initial guess g_{L-1} at any level $L-1$ can be defined as shown:

$$g_{L-1} = 2(g_L + d_L) \tag{7.4}$$

where d_L is the displacement and g_L is the initial guess at the higher level L. At the start of the algorithm, since there is no initial guess, g_{L_m} (the initial guess for level L_m) is set to 0. By adding the initial guess (g_L) with the KLT estimate d_L across levels, the total displacement d that can be captured by the pyramidal KLT with L_m levels is derived as:

$$d = \sum_{L=0}^{L_m} 2^L d_L \tag{7.5}$$

If the elementary KLT step at any level can capture a displacement of d_{max}, then the final maximum displacement d is obtained by substituting $d_L = d_{max}$ in Eq. 7.5 and computing the sum of the geometric series as $d = (2^{L_m+1} - 1).d_{max}$. For example, for a pyramid with $L_m = 2$ levels, this means a gain of 7 times the pixel displacement that can be captured at each level; in other words, with a window size $W = 3$, that can capture $d_{max} = 1$ pixel and a 2-level pyramid, a maximum displacement $d = 7$ pixels can be captured.

The above formulation however, assumes that the displacement that is captured at each pyramid level is the same. In practice, the displacement captured depends on the pyramid level. Consider this example: with a 2-level pyramid, if a maximum displacement of $d = 16$ pixels needs to be captured, what should the window size at each level be? At the highest level L_2, without any initial guess, the maximum displacement to be captured translates to $d_{max} = d/2^L_m = 16/2^2 = 4$. Therefore a window size $W_{min} = 9$ needs to be used at L_2. But once the displacement has been successfully captured at L_2 then the subsequent levels of pyramid only "refine" this result. For instance, if the result at L_2 is $d_2 = 3.75$, and the actual displacement is 4 pixels, then at level L_1 only a displacement corresponding to the error of $0.5 = (4 - 3.75) * 2$ pixels needs to be captured.

Figure 7.3 shows the displacement reported by KLT for 500 features at each level of a 2-level pyramid for the camera motions - translation, rotation and scaling. It is clearly seen that for the majority of the features, the larger displacement is captured at L_2 - for instance in Fig. 7.3 (a), a translation of $(10, 10)$ was applied to the image, resulting in a displacement of 14 pixels. At L_2, this translates to 3.5 pixels, corresponding to the peak of the histogram at L_2. Note that the peaks for

lower levels of the pyramid for all motions are closer to 0 compared to the level L_2. This confirms that the largest displacement is captured at the highest level of the pyramid, and subsequent levels refine this estimate.

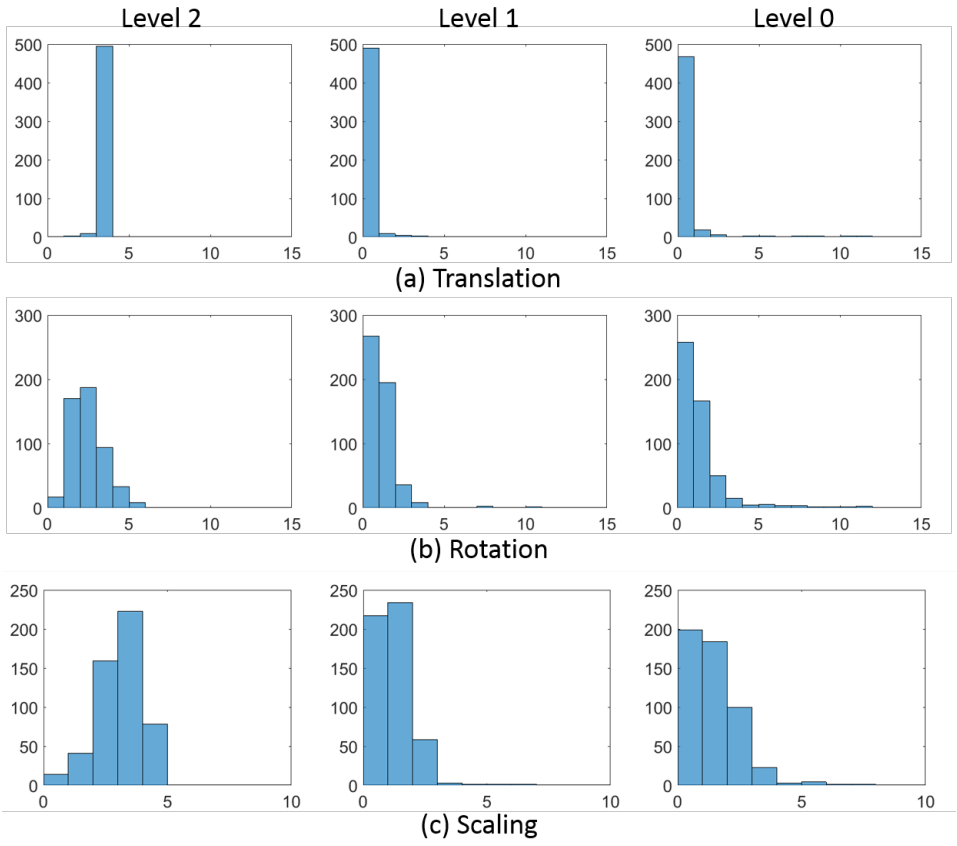

Figure 7.3: Histogram of displacements across pyramid levels for a 2-level pyramid with 500 features in the presence of (a) translation (b) rotation (c) scaling

The conventional KLT uses the same window size on all levels of the pyramid. Therefore, with $W = 9$ and a 2-level pyramid, theoretically, a maximum displacement of $d_{max} = 4$ can be captured at each level, adding up to $d = (2^{L_m+1} - 1).d_{max} = (2^{(2+1)} - 1).4 = 28$ pixels should be captured. However, in reality as Fig. 7.3 shows, the largest displacement is captured at the highest level L_2; therefore, if indeed a maximum displacement of 28 pixels needs to be captured the window size needs to be larger at $W = 15$, in order to capture the displacement of $28/2^2 = 7$ pixels at L_2.

Returning to the example, if the same window size as L_2 of $W = 9$ is applied at the lower level L_1 (that needs to capture a displacement of only 0.5 pixels), the accuracy of the tracker will fall in the presence of distortions, as this window size will be "too large".

In the analysis so far, it is assumed that for all features, the highest level successfully (i.e. with a very low error) captures the displacement. Although this assumption is valid for high quality features, when the quality is lower, the error in tracking also increases for the highest pyramid level, and the actual large displacement may be captured at a subsequent lower level. Therefore, when the displacement itself has to be captured at the higher levels, the window size needs to be large enough. But once it has been captured, and this estimate needs to be only "refined" with higher resolution in the lower levels of the pyramid, then a smaller window size is more appropriate to capture the errors in the initial guess, which were propagated from the higher level of the pyramid.

Based on the analysis in Section 7.2.1 and 7.2.2, it is clear that the conventional KLT using a translation model is susceptible to errors when a fixed window size is used, in the presence of rotation and scaling. Given a feature with an associated local motion, there exists an optimal window size that is *large enough* to capture the displacement of the feature but *small enough* to not be affected by the distortion that the neighbourhood of the feature undergoes. Therefore:

- In the presence of rotation and scaling, a fixed window size cannot be applied to all features in the frame. It needs to adapt to the motion experienced by each individual feature.
- Within the pyramidal implementation, the displacement that needs to be captured at the higher levels is typically larger compared to the subsequent lower levels, and therefore the window size needs to adapt to the displacement that needs to be captured at each level.

Adapting KLT Window Size

In this section, a novel adaptive windowing method is proposed for KLT, to improve its robustness in the presence of distortions due to rotations and scale changes. The proposed method finds a near-optimal window size for each feature to improve the accuracy of the feature correspondence.

Types of Tracking Errors

Since it is established that an optimal window size will result in the highest accuracy for a feature correspondence, it is critical to understand the kind of tracking errors that occur when this window size is sub-optimal - i.e. "too small" or "too large", in order to detect the failure due to sub-optimal window size. This is illustrated in Fig. 7.4 that shows the KLT tracks for a global in-plane rotation of 10 degrees. A global rotation has been chosen as this global motion involves various degrees of displacement as well as distortion in the local motion for the features. For a feature at (x, y) in current frame, the ground truth correspondence (x_{GT}, y_{GT}) in next frame is known and the KLT estimate is represented as (x_{KLT}, y_{KLT}). The tracking error ε_t for each correspondence is computed as follows:

$$\varepsilon_t = \sqrt{(x_{GT} - x_{KLT})^2 + (y_{GT} - y_{KLT})^2} \tag{7.6}$$

The feature correspondences are classified into 3 categories based on their tracking errors ε_t: green dots represent tracks with $\varepsilon_t < 1$ pixel; yellow dots represent tracks with $1 \leq \varepsilon_t \leq 5$ pixels, and red dots represent tracks with $\varepsilon_t > 5$ pixels.

When the window size is too small to capture the required displacement as in Fig. 7.4 (a), the tracks drift away resulting in high tracking error (represented by red tracks) or are pushed out of the frame (represented by blue dots). This is because, with a smaller region of support, the KLT estimates are easily skewed by noisy pixels within the search window.

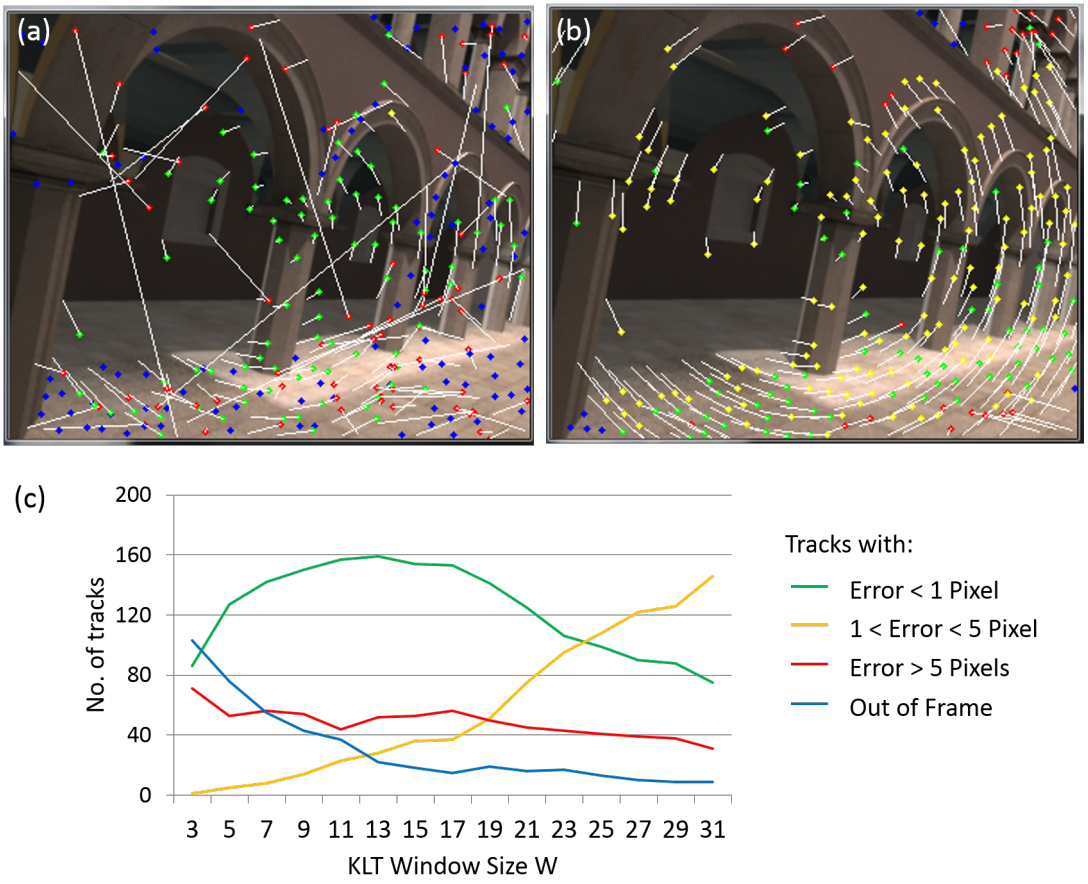

Figure 7.4: KLT tracks with in-plane rotation of 10 degrees (a) $W = (3, 3)$, (b) $W = (31, 31)$. For a range of window sizes (x-axis) (c) Total no. of tracks categorized based on tracking error ε_t

In contrast, when the window size is too large as in Fig. 7.4 (b), the estimates from the noisy pixels get averaged out and the displacement is comfortably captured, resulting in the KLT estimates being close to the final location. However, as there is distortion due to rotation, the far away neighbours tend to skew the KLT estimate and tracking error increases resulting in many yellow tracks.

Figure 7.4 (c) shows the trend in tracking errors when window size is varied: the no. of red tracks with large tracking errors ($\varepsilon_t > 5$ pixels) and blue (out of frame) tracks dominate the small window sizes, whereas for large window sizes, the no. of yellow tracks with tracking error $1 \leq \varepsilon_t \leq 5$ pixels dominates the tracks. The proposed method aims to arrive at an appropriate window size for each feature track resulting in higher number of green tracks and reducing the noisy tracks – both red and yellow.

Detecting Tracking Failure

For the search window in KLT to be adaptive to the displacement and distortion experienced by a feature across frames, the KLT feature correspondence with the current window size being sampled, needs to be evaluated to determine the degree of tracking success. First, a well-known metric is considered: the forward-backward error [85] for tracking success which has been included in the Matlab implementation of KLT. The limitations of this metric for the goal of finding an optimal window size are highlighted and the necessary conditions for incorporating it to the proposed method are shown. Then it is shown that the iterations taken by KLT to converge for a given window size can be used as an indicator of tracking performance in the smaller ranges of window size and this motivates a window size sampling scheme that starts with a small window and grows if needed.

Forward-Backward Error

The forward-backward error [85] is a well-cited metric to automatically detect tracking failures. It is based on the *time-reversibility* (also known as *forward-backward consistency*) assumption: that correct tracking should be independent of the direction of time-flow. In other words, if the tracking is correct for a feature correspondence (P_1, P_2) where P_1 is the location of the feature in frame t and P_2 is its location in frame $t+1$, then if the tracker is applied in the reverse direction in time, to P_2 in frame $t+1$, it should arrive at a location very close to P_1 in frame t. Therefore tracking is applied in both the forward and backward directions in time, and the error between the two trajectories, called the forward-backward error is computed. This error is used to automatically detect slowly drifting tracking trajectories in [85] and is now a parameter flag in the Matlab implementation of KLT. In [85], if the forward-backward error is higher than a threshold of 1 pixel, then the track is declared a failure.

First, the forward-backward error is applied to a large window size ($W = 31$) to detect features with distortions that became yellow tracks (with tracking error

$W = 31$	G	Y	R	B
No. of Tracks	64	148	35	11
No. of tracks with FB Error< 1 Pixel	56	117	8	-
% Tracks correctly classified	88%	21%	77%	-

Table 7.1: Forward-Backward Error for Detection of Tracking Failure for Various Types of Tracks with $W = 31$ (G: $\varepsilon_t < 1$, Y: $1 \leq \varepsilon_t \leq 5$, R: $\varepsilon_t > 5$, B: Out-of-frame)

$1 \leq \varepsilon_t \leq 5$). Table 7.1 shows the results for the feature set with a camera motion of an in-plane rotation of 10 degrees. Using the forward-backward error, 88% of the green tracks are correctly declared as successful. But only 21% of the yellow tracks are declared as failures. In other words, the forward-backward error is well within the 1-pixel threshold for 79% of the yellow tracks, and therefore it is unable to distinguish the yellow tracks from the (more accurate) green tracks. This is attributed to the fact that although these yellow tracks have a higher tracking error ε_t, it does not cause the reverse track to *drift away*, resulting in a small forward-backward error.

Next the forward-backward error is applied to small window sizes that resulted in the failed red tracks (tracking error $\varepsilon_t > 5$ pixels). An example is shown for the same camera motion as above of in-plane rotation of 10 degrees, with a window size of $W = 5$ in Table 7.2. For the red tracks, in 91% cases the forward-backward error is higher than the threshold of 1 pixel. However, it was found that, out of these 91% tracks, 32% fail in the reverse direction by going out of frame or being declared a KLT failure, and therefore in these cases the forward-backward error is not even computed. This also happens for green tracks that are wrongly declared as failures - in this case, 24% are declared as tracking failures, however 9% of them are in fact due to failure of the reverse KLT tracking. In all, for $W = 5$, there are 28 cases of reverse track failing compared to 5 cases for $W = 31$. This shows that although forward-backward error can be used to detect an erroneous track, it assumes that the tracker is setup to handle the displacements of the majority of features - specifically, that the window size is "large enough". However, it is unable to detect the yellow tracks with a large window size. For the red tracks with the small window size, applying the forward backward error check is unnecessary as

$W = 5$	G	Y	R	B
No. of Tracks	124	6	53	75
No. of tracks with FB Error< 1 Pixel	94	2	5	-
% Tracks correctly classified	76%	67%	91%	-

Table 7.2: Forward-Backward Error for Detection of Tracking Failure for Various Types of Tracks with $W = 5$ (G: $\varepsilon_t < 1$, Y: $1 \leq \varepsilon_t \leq 5$, R: $\varepsilon_t > 5$, B: Out-of-frame)

there are a large number of failures in the reverse tracks. In the proposed method, the forward-backward error is only applied as a final check after a suitable window size is determined using simpler checks.

Convergence within Maximum KLT Iterations

It was found that the red tracks that have *drifted* far from their actual location (tracking error $\varepsilon_t > 5$ pixels) also converge with distinctly larger number of KLT iterations as shown in Fig. 7.5, where the average number of iterations[1] is significantly higher for the lost red tracks. However, the yellow tracks converge fast similar to the green tracks at the end of KLT. Therefore, the iterations can be used as an indicator to detect tracking errors caused by a window size that is "too small" to capture the displacement of the feature (represented by the red tracks). This motivates a window size sampling scheme that starts with a small window size and grows when a tracking failure is detected, by monitoring the KLT iterations.

Handling Erroneous Early Convergences

Not all *drifting* (red) tracks hit the maximum iterations with a small window size. Sometimes, tracks converge fast for a certain window size when they hit a local minimum - on a similar looking patch in the neighbourhood. Increasing the window size, exposes more of the neighbourhood and the track can be pushed out of the local minimum. In order to prevent such early convergences at local

[1]Note that the number of iterations that KLT takes to converge at each level of the pyramid is averaged, for each feature.

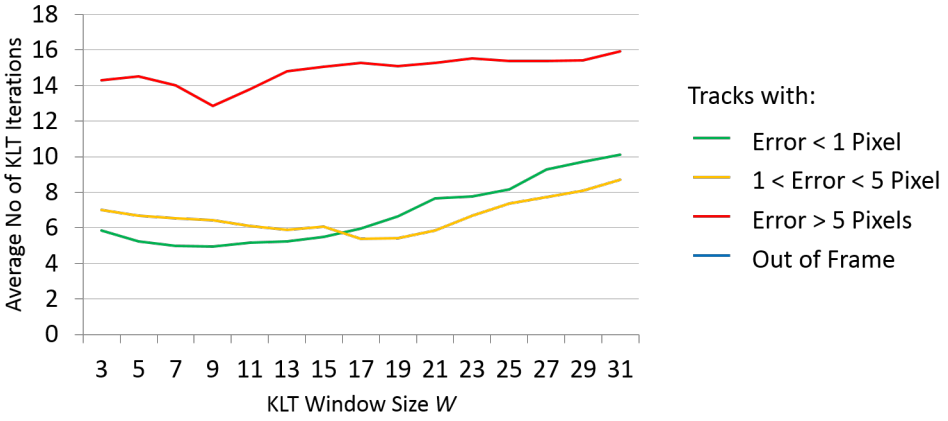

Figure 7.5: Average no. of iterations taken by KLT to converge for the various classes of tracks based on their tracking errors

minima, a stability criterion is employed that requires the previous and the current window size sampled to be *fast* convergences. A minimum iterations threshold (= 8 iterations) was empirically determined, and is used to declare the convergence as *fast*.

Integrating with Pyramidal KLT

In Section 7.2.2, it was shown that the displacement that needs to be captured varies across the pyramid levels. Therefore, the window size sampling strategy is applied to, each feature, at each level of the pyramid, in the pyramidal KLT implementation, so that an appropriate window size is used depending on the displacement that needs to be captured at that level of the pyramid. Specifically, this allows the window size to grow as needed at the highest level when the displacement to be captured is unknown. Once the displacement has been captured, at the subsequent lower levels, smaller window sizes result in fast convergences avoiding noisy estimates from far away neighbours in the presence of distortion.

Figure 7.6 shows the flow of the proposed adaptive window strategy for KLT (applied to each feature and a given level L of the pyramid) by combining all the steps described above. The initial estimate g_L for the current feature is passed from the processing at the higher level $L + 1$. As described in Section 7.2.2, at

the highest level L_m the initial guess is 0. KLT is applied with a window size W, starting with the smallest size W_{min}. If KLT converges within the maximum number of iteration (*MaxIterations*), then a check for *fast* convergence is performed by looking at the earlier window size sampled. Therefore, at least two window sizes are always sampled. If either of these checks fails, then the window is declared as sub-optimal and the next higher window size is sampled. If both these checks are passed and the current level is not the highest level L_m of the pyramid, then the track is selected and passed on to the next lower level $(L-1)$ of the pyramid processing. However, if this is the highest level L_m, the forward-backward error [85] is applied as an additional check before selecting the track. It should be noted that for the lower levels, as there is an initial guess from the higher level, forward-backward error cannot be applied to qualify the tracks. Finally, when the maximum window size W_{max} is reached, if the lowest pyramid level $(L = 0)$ is reached, the track is selected only if it converged within the maximum number of iterations, otherwise it is rejected. For higher levels, the track is selected and passed on to the next lower level $(L-1)$ of the pyramid processing.

Performance Evaluations

In this section, the proposed adaptive window size strategy for KLT will be evaluated with existing KLT algorithms. The proposed method is referred to as *Adaptive KLT*. First, the evaluation setup will be described. Next, the performance results will be reported.

Evaluation Setup

The evaluations need to compare the proposed *Adaptive KLT* method with the baseline KLT algorithms on its robustness to distortions and its overall efficiency while achieving this robustness.

Image data: First, in order to evaluate the robustness, simulated frames have been generated, by applying global rotation and scale change, as described in

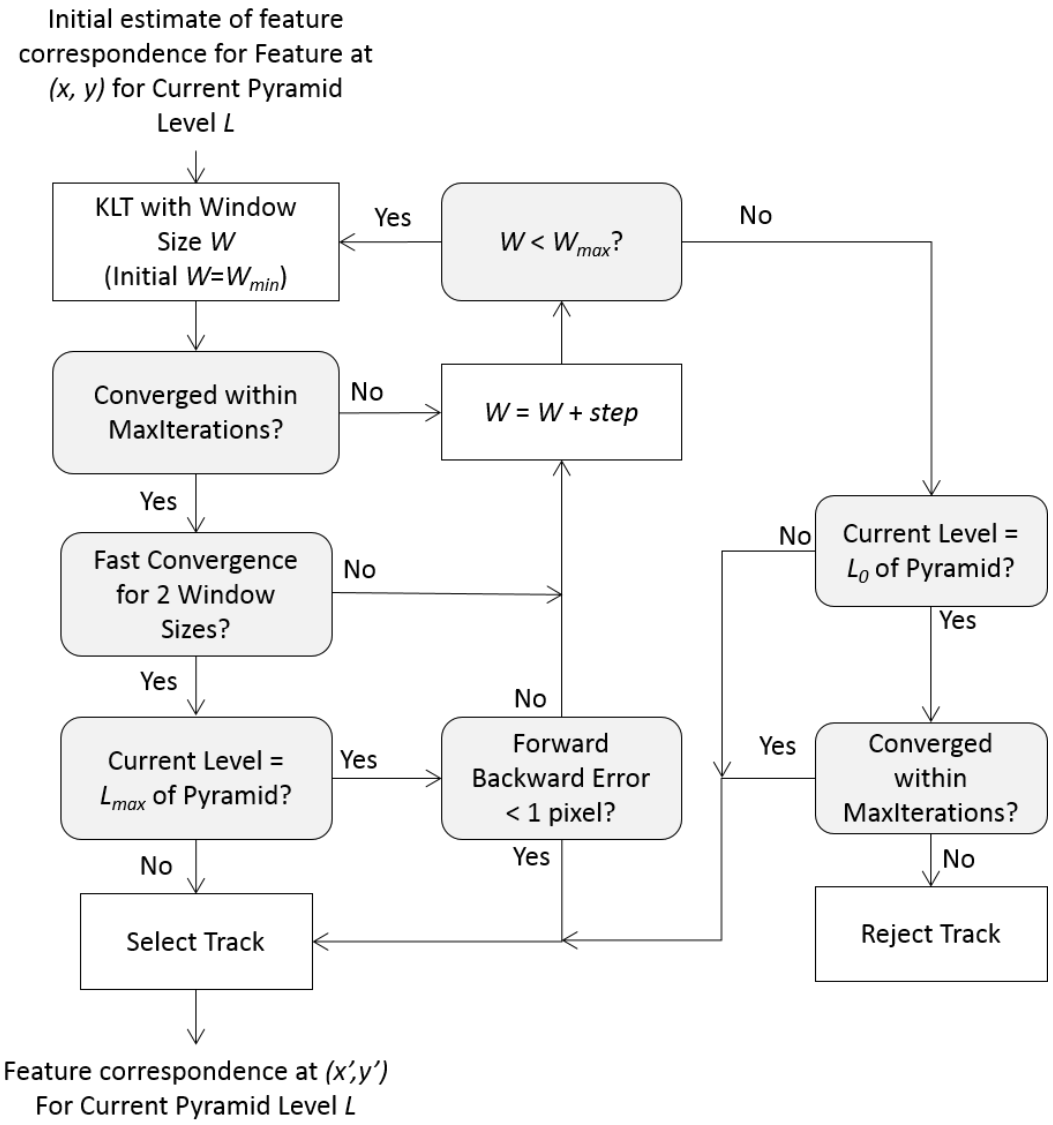

Figure 7.6: Adaptive windowing for KLT feature tracker (at each pyramid level)

Section 6.5.1, without the moving objects. Images from Fig. 7.7 have been used. A range of in-plane rotations of angles 0-20 degrees and scale factor of 0-0.8x is applied on these images.

Figure 7.7: Image data used for simulated data in the evaluation of adaptive windowing for KLT

In addition, a ground truth annotated tracking dataset in [117] is used, as shown in Fig. 7.8. The dataset consists of video sequences where the camera and the transparent plate with the rectangular texture are moved to create – rotation, zoom, perspective distortion and panning. This causes both displacement and distortion for the features on the texture area. The ground truth has been provided for the motion of the plane formed by the four *red* dots at the corners of the plate, in the form of the homography matrix H. Applying this homography, the ground truth tracked locations, for the features in the texture area, can be determined. Feature detection is performed only inside the plane texture area and features very close to the boundary are ignored in order to eliminate boundary effects as shown in Fig. 7.8 (d). Features are selected in every frame and tracked in the next frame. As the frame-to-frame motion is very small for this dataset, every other frame is skipped and the results are shown for this wider baseline video sequence.

Baseline algorithms: In order to evaluate the proposed adaptive window size strategy for KLT, it is compared with the following baseline algorithms:

1. Conventional translation-model (as in Eq. 7.1) KLT tracking algorithm with a fixed window size W_{fixed}, referred to as *Conv KLT*. For the baseline KLT algorithms these parameter values were used: $W_{fixed} = 31$, Maximum no. of iterations $k = 20$, No. of pyramid levels $L_{max} = 2$, Minimum eigen threshold $\lambda_{th} = 0.0001$.

2. Affine-model (as in Eq. 7.2) KLT [79] with a fixed window size W_{fixed}, referred to as *Affine KLT*. All other parameters are setup the same as for *Conv KLT* above.

For the *Adaptive KLT*, Minimum window size $W_{min} = 5$, Maximum window size $W_{max} = 31$, Step size $step = 2$, Minimum no. of iterations $minIterations = 8$. All other parameters are the same as the baseline algorithms. The KLT implementations in OpenCV v2.4.9 for the conventional (*calcOpticalFlowPyrLK*) and affine KLT (*cvCalcAffineFlowPyrLK*) are used for the baselines. The *Adaptive KLT*

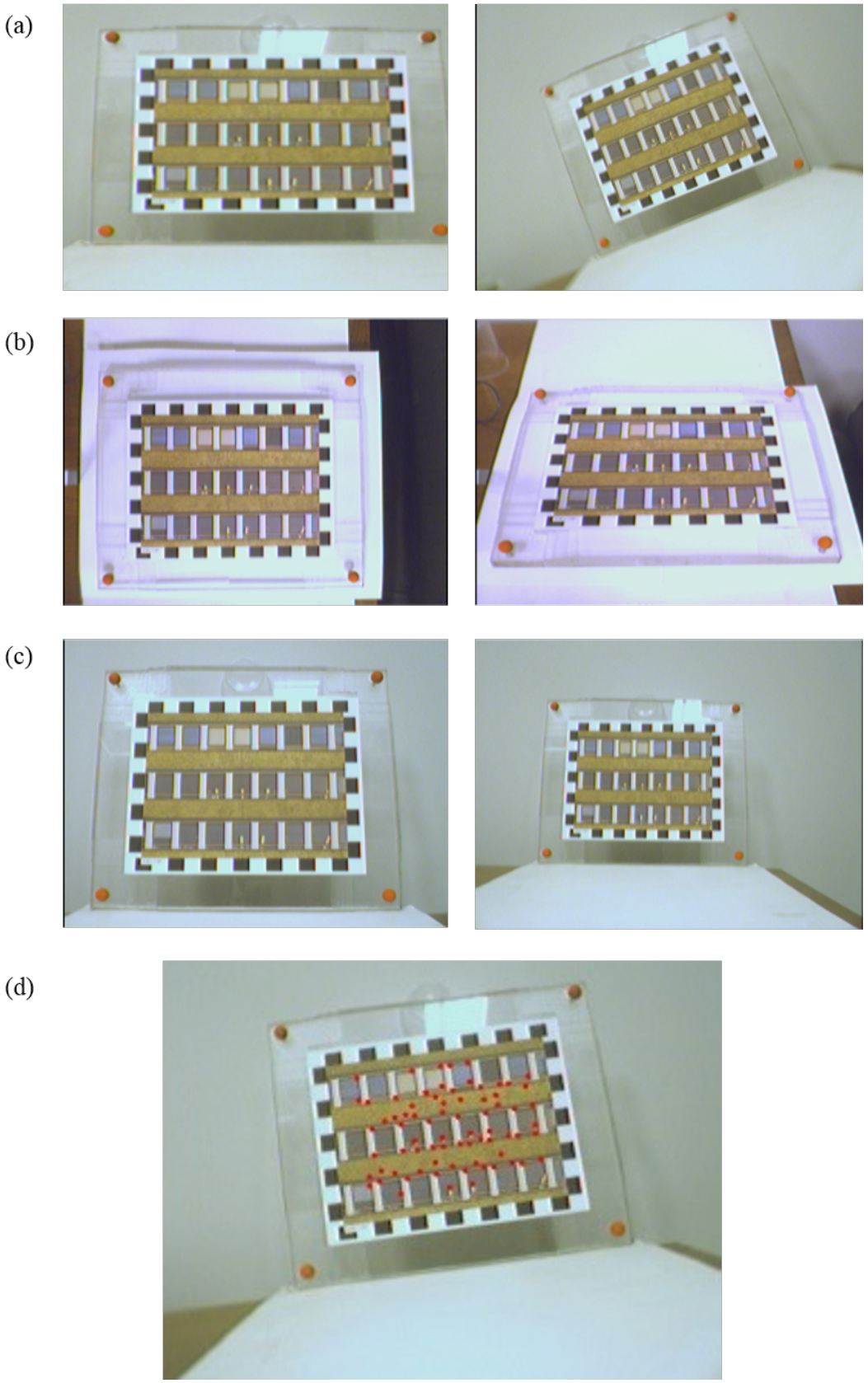

Figure 7.8: Sample frames from the tracking dataset [117] used for evaluations (a) Rotation (b) Perspective Distortion (c) Zoom (d) Shi-Tomasi features (red) selected within the texture area

was implemented by modifying the *calcOpticalFlowPyrLK* function as outlined in Fig. 7.6.

Evaluation Criteria: Feature detection is performed with the Shi-Tomasi feature detector [50] in the current frame resulting in a feature set. Each variant of the KLT tracker (*Conv KLT*, *Affine KLT*, *Adaptive KLT*) is then applied on this feature set. This generates the corresponding location (x_{KLT}, y_{KLT}) for each feature at (x, y) in the current frame. The ground truth correspondences (x_{GT}, y_{GT}) in the next frame, are available for both the simulated frames and the tracking dataset. The tracking error ε_t is computed as in Eq. 7.6. If the tracking error $\varepsilon_t < 1$ pixel, the track is labeled *useful* (corresponding to the green tracks in the earlier sections), else it is labeled as *noisy* (red and yellow tracks). The robustness of the tracker to various motions is determined by the extent to which it can maximize the number of *useful* tracks and minimize the number of *noisy* tracks.

Performance Results

Robustness Evaluations:

Figure 7.9 shows the average number of *useful* and *noisy* tracks for all the simulated frames in the chosen images. For the global rotation, the performance of all the KLT variants is comparable for angles in the range of 1-5 degrees. In the range of 6-15 degrees, *Conv KLT* sees a drastic drop in the no. of *useful* features, and a corresponding increase in the no. of *noisy* features, indicating an overall rise in the tracking error. *Affine KLT* is able to salvage many of the noisy tracks and hence shows an improvement in the no. of *useful* tracks compared to *Conv KLT*. However the *Adaptive KLT* is the most successful in eliminating the *noisy* tracks while still matching the *Affine KLT* in the number of *useful* tracks reported, thus generating the cleanest feature set among all the KLT variants considered for this range of angles. Beyond 15 degrees, all the 3 variants of KLT are unable to deal with the distortion incurred.

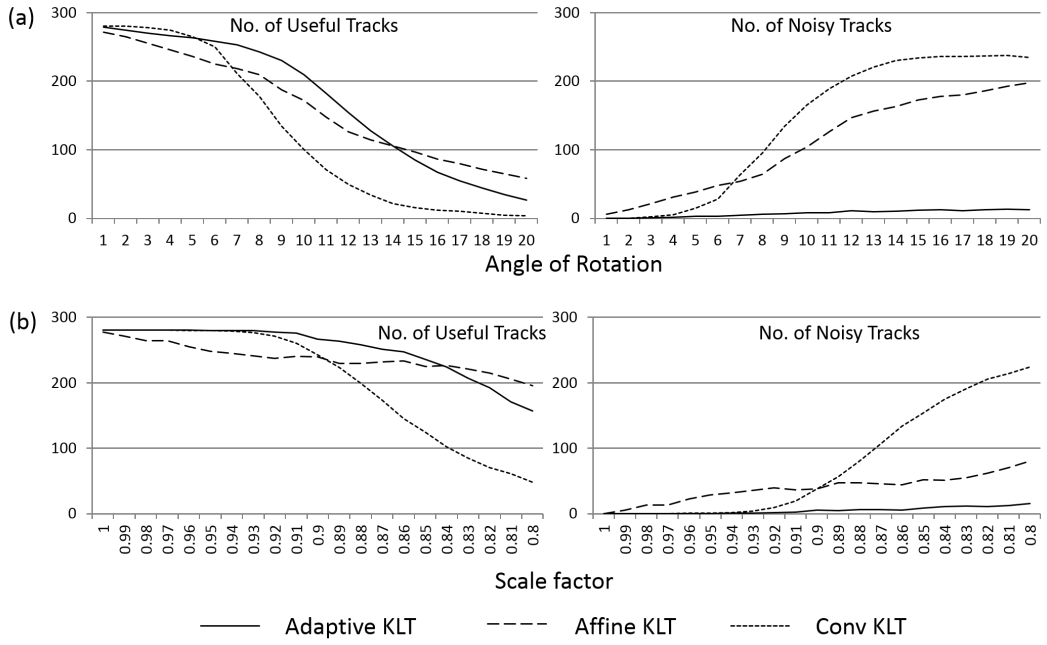

Figure 7.9: Accuracy evaluation of adaptive windowing for KLT: Number of useful and noisy tracks for varying (a) rotation angles and (b) scale factors

When the scale factor is increased, *Conv KLT* suffers in the range of 0.89-0.8x, showing a marked decrease in the no. of *useful* tracks and a corresponding increase in the no. of *noisy* tracks. However, both *Affine KLT* and the proposed *Adaptive KLT* perform better for the larger scale changes.

For the annotated tracking dataset [117], the evaluation results for the texture "bu" (building) are presented in Fig. 7.10. Rotation, as shown in Fig. 7.10 (a), (b), incurs the most inter-frame distortions among all the 3 motions considered. *Conv KLT* suffers in frames 7-9 and 13-15 and *Affine KLT* improves the number of *useful* tracks, however it is unable to keep the *noisy* tracks to a minimum. *Adaptive KLT* outperforms the *Affine KLT* both in the number of *useful* and *noisy* tracks. For perspective distortion, as shown in Fig. 7.10 (c), (d), the total number of tracks goes down towards the end of the video sequence, as very less area of the texture is seen. *Conv KLT* suffers in terms of robustness, but both *Affine* and *Adaptive KLT* show similar robustness. The zoom video sequence in Fig. 7.10 (e), (f) has a zoom-out motion, and therefore the area covered by the texture goes down as the video progresses. This is reflected in the total no. of

tracks reported by all KLT variants falling. In frames 6-8 both the *Conv* and *Affine KLT* suffer in terms of number of *noisy* tracks but *Adaptive KLT* keeps the *noisy* tracks to a minimum. In the panning video sequence in Fig. 7.10 (g), (h), the displacement is large and the distortion in minimal. *Adaptive KLT* has comparable performance with the *Conv KLT*, but *Affine KLT* is unable to handle the large displacement.

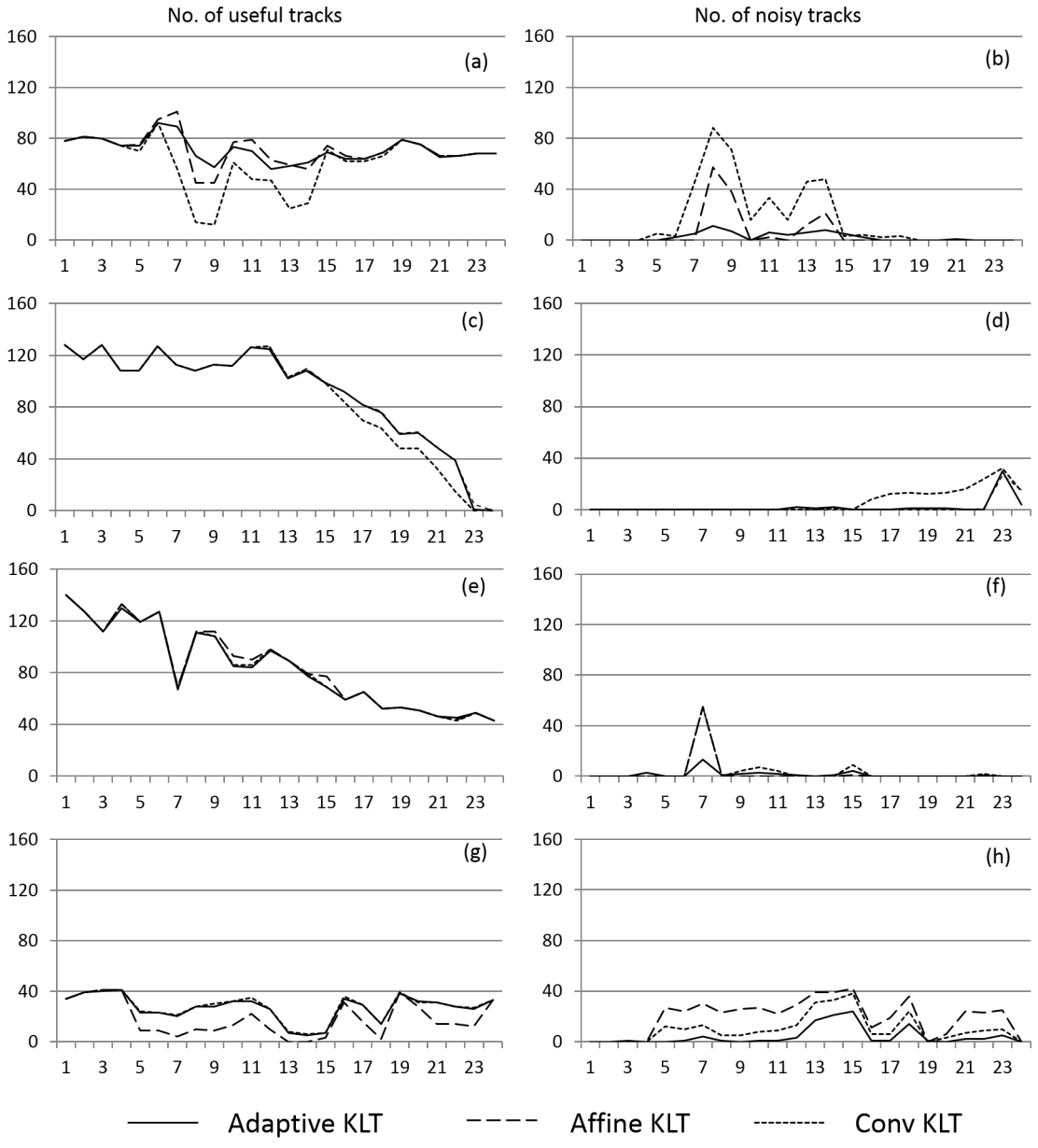

Figure 7.10: Robustness results on tracking dataset [117] (a)-(b) Rotation (c)-(d) Perspective Distortion (e)-(f) Zoom (g)-(h) Panning

Overall, the robustness results show that even though *Affine KLT* is designed to allow distortions in the template patch, it is still outperformed by *Adaptive KLT*. This can be attributed to the large fixed search window size that is used with *Affine KLT* which prevents it from reaching the accuracy, that a more controlled and therefore, optimal window size reached by the *Adaptive KLT* can offer, leading to higher number of *useful* tracks. In addition, the proposed *Adaptive KLT* is able to reliably discard the *noisy* tracks, after all the window sampling trials, by applying the convergence within maximum iterations criterion, at the lowest level of the pyramid. In contrast, the baselines report all tracks as long as they are within the frame, and this leads to a large number of *noisy* tracks when there is drastic distortion.

Efficiency Evaluations:

Next, the efficiency of the proposed *Adaptive KLT* method is compared with the baselines when applied to the annotated tracking dataset [117] as in Fig. 7.8. The computation times were measured on a 3.5 GHz Intel (R) Xeon (R) desktop computer and are shown in Fig. 7.11. As expected, the *Affine KLT* being the most computationally complex of all the methods considered here, results in the highest computations times – incurring an average computation time 7x times more than the proposed *Adaptive KLT*. As *Adaptive KLT* iteratively samples various window sizes, in the presence of distortion, the computation time is marginally higher than the *Conv KLT* which uses a single fixed window. The *Affine KLT* is only able to operate at an average of 5 frames per second (FPS), while the *Adaptive KLT* and *Conv KLT* operates at an average of 39.6 FPS and 41.5 FPS respectively. This shows that the proposed *Adaptive KLT* is not only robust against distortions but it can also be used in real-time applications.

Adaptive KLT for Sparse-GME

In Chapter 6, the sparse-GME method was proposed that adapts the number of feature correspondences needed for performing global motion estimation with the

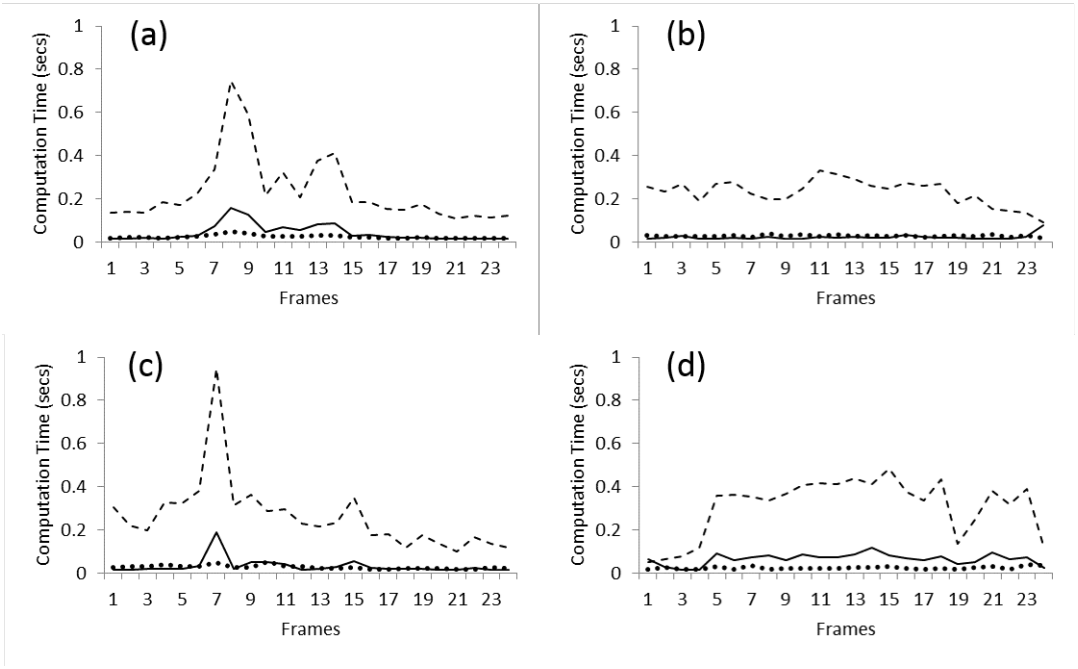

Figure 7.11: Computation time on tracking dataset [117] (a) Rotation (b) Perspective Distortion (c) Zoom (d) Panning motions

complexity of the scene. For the evaluations in Section 6.5, the conventional fixed window KLT, *Conv KLT*, was employed. These results are shown in the left column of the Fig. 7.12. *Ground truth* refers to the background PSNR (as computed in Section 6.5) corresponding to the ground truth global motion. *Dense-GME* refers to using a dense feature set. *Sparse-GME* refers to the proposed adaptive feature selection proposed in Chapter 6. *4x4x1* is a naive sparse features method that does not adaptively repopulate features when GME fails. The right column in Fig. 7.12 shows the same set of results when the *Adaptive KLT* is employed in place of the *Conv KLT*. It is clear that when the number of moving objects in the scene increases, a larger density of features can achieve higher accuracy for the GME and employing the more robust *Adaptive KLT* does not significantly improve the results. However, when the global rotations are increased as shown in Fig. 7.12 (c), the density of the feature set does not affect the GME accuracy. With larger rotation angles, even the *Dense-GME* suffers. This is because the tracking accuracy deteriorates with larger rotation angles due to distortion. *Adaptive KLT* improves the accuracy of the feature correspondences and this results in a substantial improvement in the accuracy of the GME for both *Dense-GME* as well as the

Sparse-GME, as shown in Fig. 7.12 (d). Similar improvement in GME accuracy is seen for scale change in Fig. 7.12 (e), (f). *Adaptive KLT* therefore enables the use of very sparse feature sets for global motion estimation achieving high accuracy in the presence of distortions.

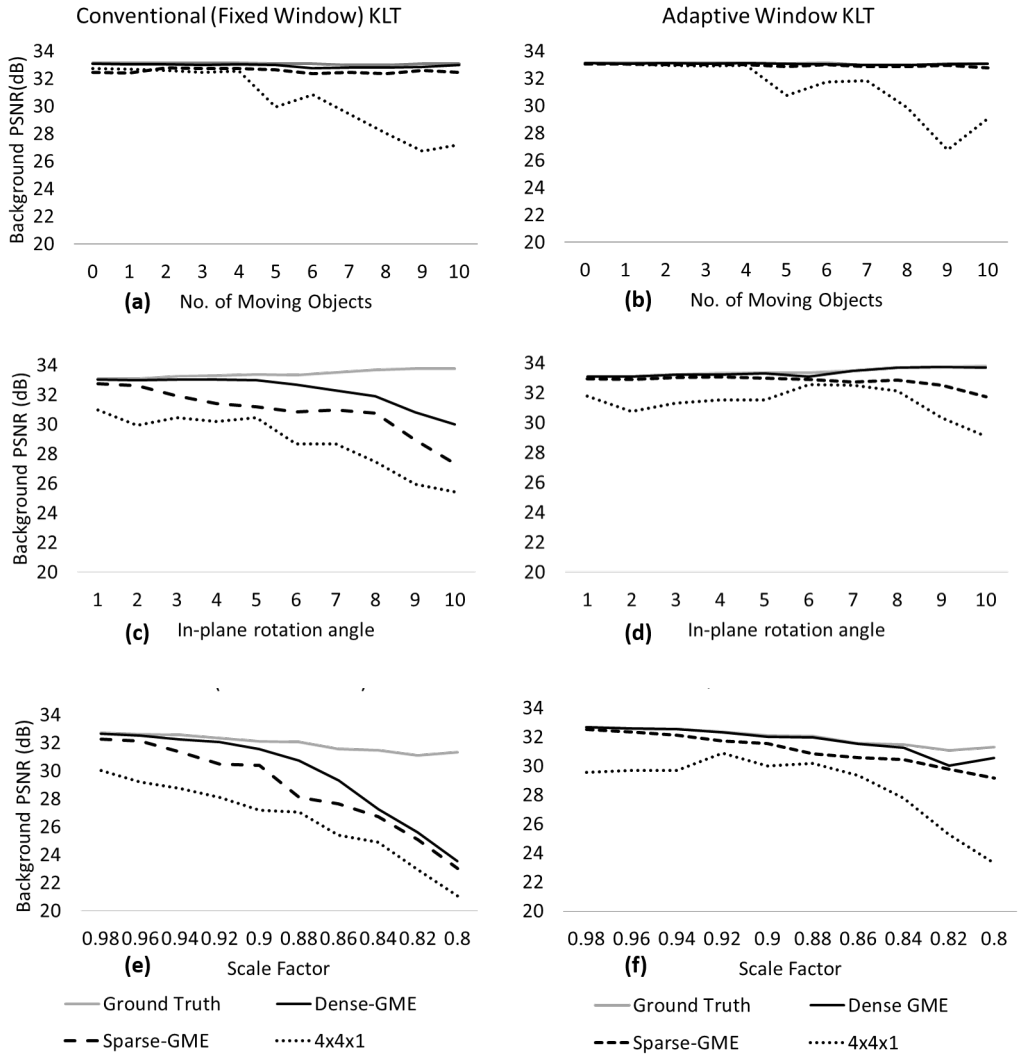

Figure 7.12: GME with adaptive windowing for KLT (right column) compared to using conventional KLT (left column) (a, b) Moving objects 1-10 (c, d) In-plane rotation angle 1-10 degrees (e, f) Scale factor of 1-0.8x

Summary

In this chapter, a novel adaptive window size strategy [5] is proposed for the classical Kanade-Lucas-Tomasi (KLT) feature tracker in order to make it robust against

distortions due to rotations and scaling. It was shown that the search window size determines the accuracy of the KLT feature tracker, and in the presence of distortions around the feature due to rotation and scaling, this window size needs to adapt to the local displacement of the feature. The proposed adaptive strategy adopts a controlled selection of window size by sampling various window sizes starting with the smallest window. By employing the iterations needed to converge for KLT with a given window size, as an indicator of the success of KLT, the sampling of window size is performed with no additional computational overhead for evaluating the KLT tracks. This enables faster sampling of window sizes resulting in a near-optimal window size when the iterations needed to converge, stabilize. The proposed window size sampling scheme also lends itself well to the pyramidal implementation of KLT, such that the optimal window size is determined at each level of the image pyramid, leading to higher accuracy.

The evaluations on a tracking dataset show that the proposed strategy significantly outperforms the conventional fixed-window KLT in terms of robustness against rotation and scaling, and achieves comparable robustness of the more complex affine KLT with 7x faster runtime. It was also shown that when applied for global motion estimation with simulated global motion, of drastic in-plane rotations and scale changes, adaptive windowing for KLT leads to 70% reduction in the GME error for the proposed sparse-GME, compared to using a fixed window size.

The proposed adaptive window size strategy substantially improves the robustness of the KLT feature tracker without incurring computational overhead. This improved KLT can therefore be applied to scenarios where the distortion around the feature is likely to be high. For the aerial surveillance videos, the proposed method results in low-complexity yet robust feature tracking leading to high accuracy in the global motion estimation without additional computational cost.

8

Framework for Adaptive Low-Complexity GME

In this thesis, compute-efficient and adaptive techniques have been developed for the key functional blocks in a feature-based global motion estimation (GME) pipeline, namely corner detection, feature tracking and robust estimation. This chapter describes how these functional blocks can be combined to realize an adaptive and low-complexity GME pipeline for on-board processing of aerial videos.

In a conventional setup of a feature-based GME, as shown in Fig. 2.6, the GME is performed as a one-shot pipeline in which each block processes the frame(s) and hands over the results to the next stage in the pipeline: the corner detector passes the locations for corners to the feature tracker, which then generates the feature correspondences and passes them to the robust estimator, that finally

computes the global motion model parameters. Therefore, the parameters for each of these functional blocks, are setup to handle worst case scenarios. In this thesis, low-complexity and adaptive methods for the functional blocks of GME have been developed by replacing fixed parameters with adaptive parameter selection methods as highlighted below:

- *Quality threshold in corner detection*: The proposed methods for corner detection with Shi-Tomasi and Harris in Chapters 3, 4 and 5 allow the computations needed for corner detection to adapt to the number and density of corners needed by the end-application. This is achieved, by replacing the corner measure with a simpler pruning method, to rapidly select corner candidates, and then automating the threshold selection for extracting these candidates. In contrast to a fixed threshold, that is setup to detect the required number of corners for a wide range of image content, the threshold for quality of corners now adapts to the required number of corners and the quality of the image content.

- *Search window size in feature tracking*: The robustness and complexity of the KLT feature tracker is enhanced in Chapter 7 by replacing a fixed search window parameter with an adaptive window. The fixed window is setup to capture the largest displacement. However, for small inter-frame displacements, this incurs unnecessary computations. In addition, in the presence of distortions, the tracking accuracy deteriorates substantially. The adaptive windowing method is applied to each KLT operation - i.e. for each feature at every pyramid level, thereby adapting to the specific local displacement that needs to be captured.

- *Number of features for robust estimation*: In Chapter 6, the robust estimator RanSaC, is employed on very sparse but well-distributed feature correspondences. An on-line evaluation method is used to associate severity levels for estimation failures and this guides a progressive re-population of features, only in regions that have lost representation. The conventional robust estimation assumes that a dense set of feature correspondences is available.

In contrast, the number of features needed for estimation in the proposed method, adapts to the complexity of the camera and object motion in the frame pair and additional features are added on an on-demand basis depending on the quality of the current estimation.

It is proposed to combine the individual functional blocks such that the GME can be performed with minimal computation effort when conditions are conducive, specifically:

- *Image content*: The frame contains enough number of well-distributed, good quality corners.
- *Feature correspondences*: Chosen corners can be tracked successfully, i.e. KLT can handle the camera motion, the features are not occluded or move out of the frame, and the majority of the correspondences belong to the background and can contribute to the GME.

The overall flow of the proposed adaptive low-complexity GME framework is shown in Fig. 8.1. The sparse-GME method as proposed in Chapter 6 acts as the main *controller*. A well-distributed feature set is extracted by detecting corners using the proposed pruning-based corner detection (PP-ER + Shi-Tomasi/Harris), as described in Chapters 3-5, in a block-based manner. The proposed adaptive window based KLT feature tracker is employed to generate the feature correspondences for this sparse set. This is fed into the sparse-GME which then applies the *2-RanSaC* method as described in Chapter 6 and evaluates the GME with the sparse set. If the GME is successful in the presence of conducive conditions, the process ends. Only if the GME is not successful, the corner detector and feature tracker are invoked by the *controller* for additional feature correspondences. The sparse-GME progressively checks the severity of the failure and then creates demand for more corners to re-populate blocks that have lost representation, in a systematic manner as shown in Fig. 6.4. In the following sections, these interactions between the modules are described in more detail.

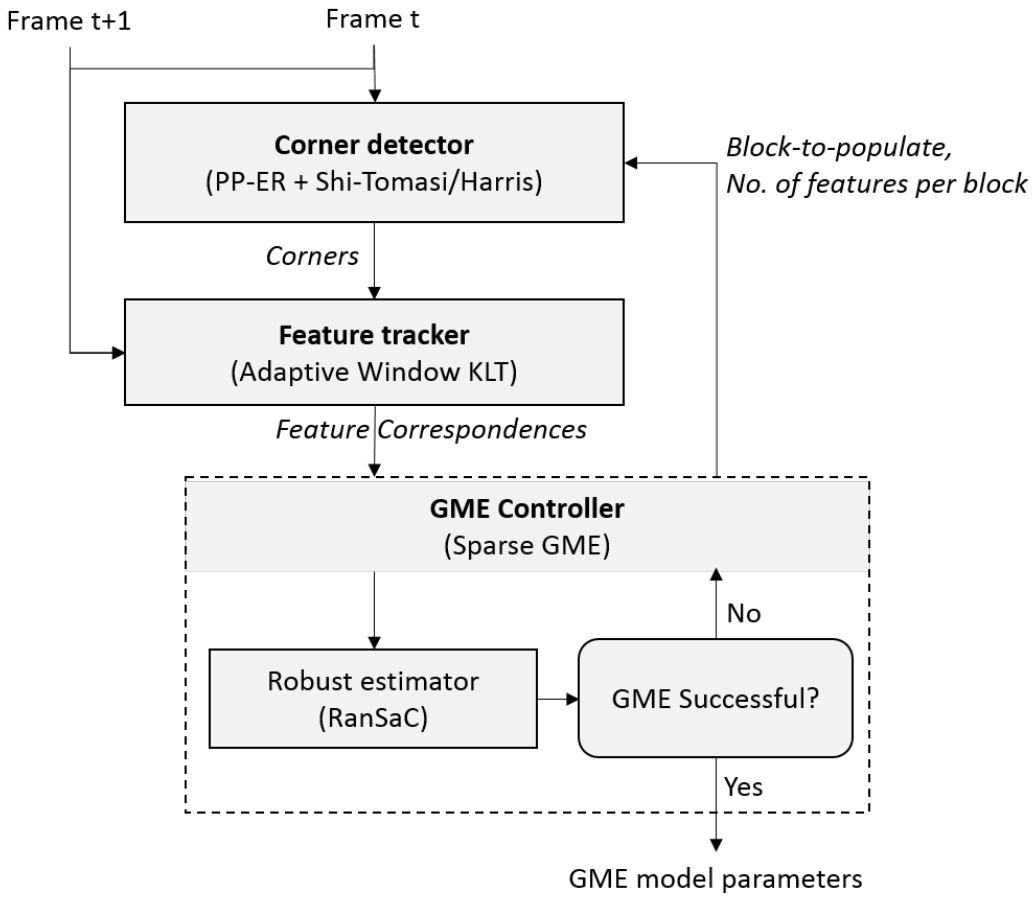

Figure 8.1: Unified framework for low-complexity and adaptive GME

Corner Detection

The automated thresholding with pruning method proposed in Chapters 3, 4 and 5 for the Shi-Tomasi and Harris corner detectors, lends itself well to block-based processing, where each block is processed as an independent image and reports its corners, as shown in Fig. 8.2. The required number of corners per block ($nPerBlock$) and the blocks in which the features need to be extracted (B_i) are sent in as input from the GME controller. The corner detector applies the proposed automated thresholding on each block B_i, and an optimal threshold T_i is found releasing sufficient candidates to extract the required number of corners, $nPerBlock$. The inputs from the GME controller vary depending on the severity of the alignment failure as described later in Section 8.3.

It should be noted that the flow of sparse-GME in Fig. 6.4, assumes that all blocks in the frame contain corner regions - e.g. all 16 blocks in the 4x4 grid have corners. In the first pass of automated thresholding in corner detection, a global minimum threshold on quality of corners can be applied on the best corner extracted in a block, to discard blocks without corner regions. This defines the usable blocks in the frame. The GME controller can use this information to adjust the density of features per block, such that sufficient number of corners are extracted for successful evaluation. The GME controller performs a spatial distribution check, and increases density per block if the total usable area has fallen for the current frame.

Thus the corner detection block and the GME controller adapt to the image content in extracting sufficient number of corners for successful estimation.

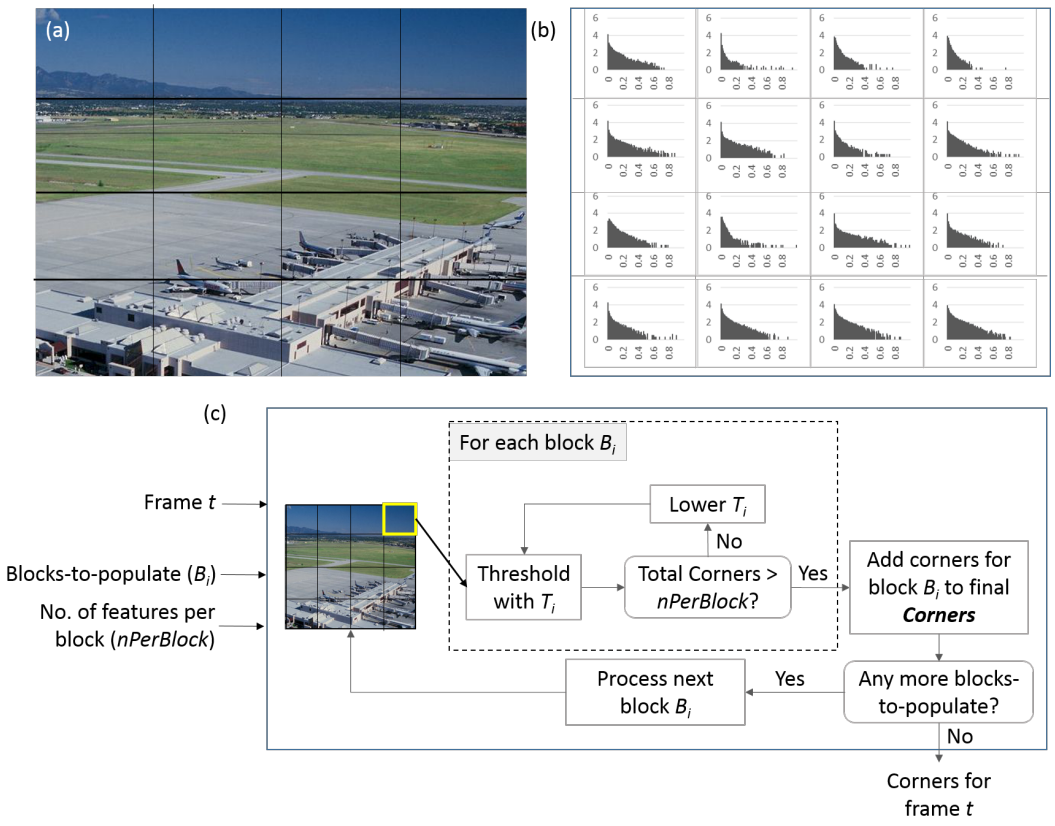

Figure 8.2: Adaptive low-complexity corner detection for GME (a) block processing for corner detection (b) optimal threshold to detect required number of corners for each block with a different corner measure histogram (c) flow of automated thresholding at block level

Feature Tracking

For each selected corner, adaptive windowing for KLT is applied resulting in accurate feature correspondence in the presence of wide range of distortions. As is described in Chapter 7, the adaptive windowing is applied to every level of the pyramid in the pyramidal implementation of KLT. The improved robustness of KLT, leading to higher tracking accuracy of the feature correspondences allows GME to be performed with a sparse set of features for a wide range of distortions due to rotations and scaling.

The conventional GME pipeline invests in a large number of features to compensate for inaccurate tracks due to distortion. However, the inability of the tracker in handling large distortions limits the range of rotation angles/zoom factors that can be handled for GME. In the proposed method, as shown in Fig. 8.3, the window size for tracking adapts to the local motion of the features, thereby improving the accuracy of the tracker. Fig. 8.3 (b) shows an example for a sparse feature set undergoing global rotation: the optimal window size W_i that the adaptive windowing arrives at, for each pyramid level is shown. The adaptive windowing is performed for each pyramid level L_k for each corner C_i. The proposed method alleviates the problem of inaccurate tracks skewing the estimation and higher robustness can be achieved even with very sparse set of features. The GME controller can also provide good initial estimates for tracking and minimum window size for the adaptive windowing strategy by maintaining a history of the global motions for the previous frames resulting in lower computations for KLT. Together, the GME controller and the adaptive windowing for KLT, invest resources in an on-demand manner, depending on the local motion experienced by each feature. The feature correspondences are then passed on to the GME controller.

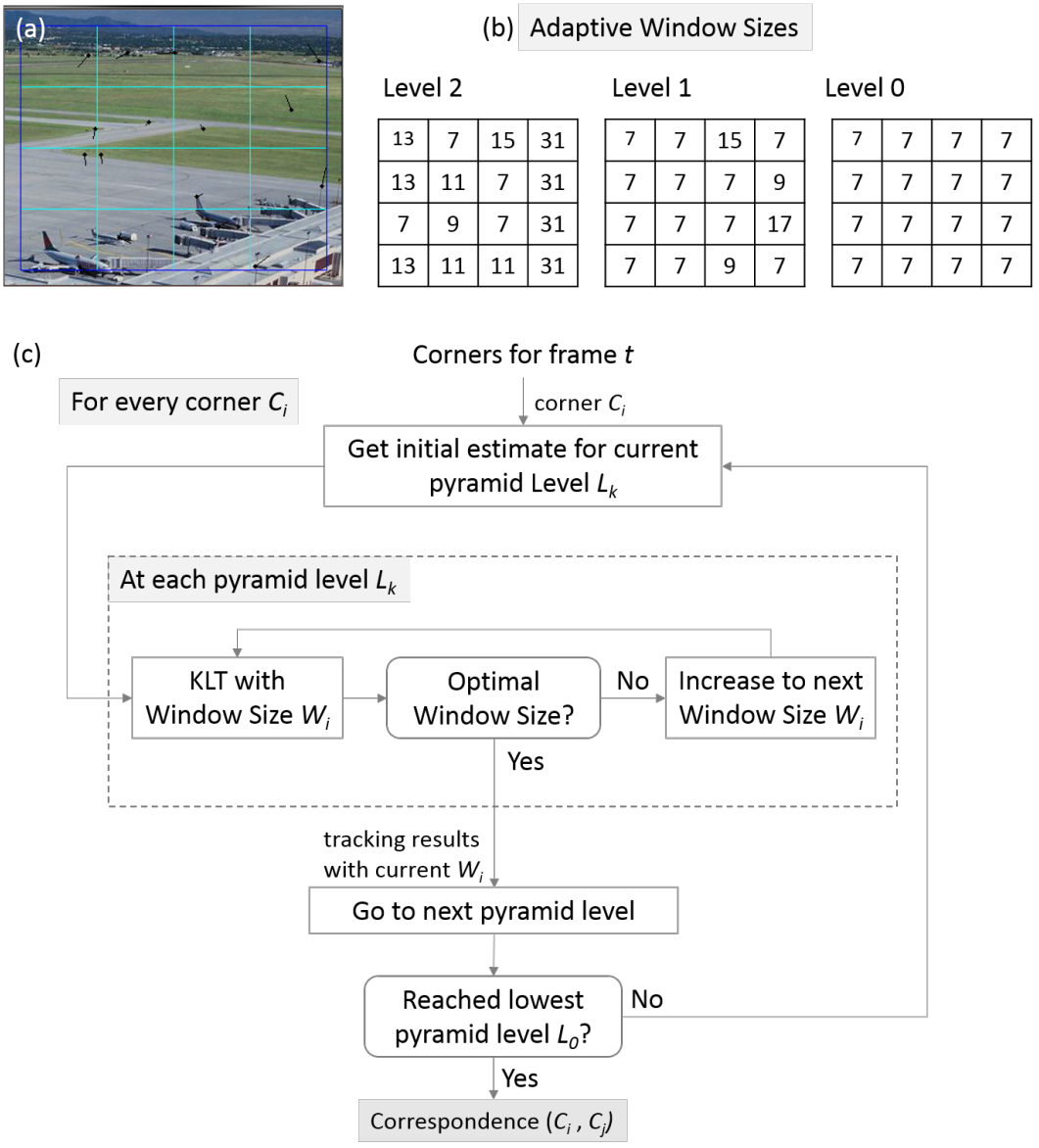

Figure 8.3: Adaptive feature tracking for robust feature correspondences (a) Varying displacements for global motion of in-plane rotation (b) Window size at which the method converges for each feature and at each pyramid level of KLT (c) Flow of adaptive windowing method

GME Controller

The GME controller (sparse-GME) starts with a very sparse, well-distributed feature set for the estimation and proceeds to progressively repopulate with additional features, only if the evaluation of the current estimation fails. As shown in Fig. 8.4 the GME controller uses progressive phases of repopulation (called P1-P4 for phases 1-4), with P1 being the initial sparse feature set. The corresponding feature tracks from the feature tracker are used by the robust estimator for the current GM estimation. The controller then evaluates this estimation as described in Fig. 6.4. Subsequent phases of repopulation are entered if the current estimation fails.

Corner detector: Phases P1 and P2 use a very sparse feature set with sufficient coverage, whereas phases P3 and P4 require denser sets. The specific number of corners and the blocks in which they are required become the input for corner detection block. Only blocks that need corners need to be processed. For the experiments, the sparse set of $n_1 = 1$ per block and dense sets of $n_2 = 5$ and $n_3 = 10$ features per block was used.

Robust estimator: The robust estimator (RanSaC algorithm) is invoked in two modes as follows:

- *RanSaC*: This is the conventional RanSaC which is invoked with a dense set in phase P4 when none of the earlier sparse feature sets result in successful GME. The GME controller does not perform any evaluation for the estimation, and the GME model parameters from the RanSaC are reported directly.
- *2-RanSaC*: This mode is applied for the sparse feature phases of P1 and P3. As shown in Fig. 6.3, 2-RanSaC refers to applying two sets of thresholds that determines the *high confidence inliers* enabling the evaluation of the estimation.

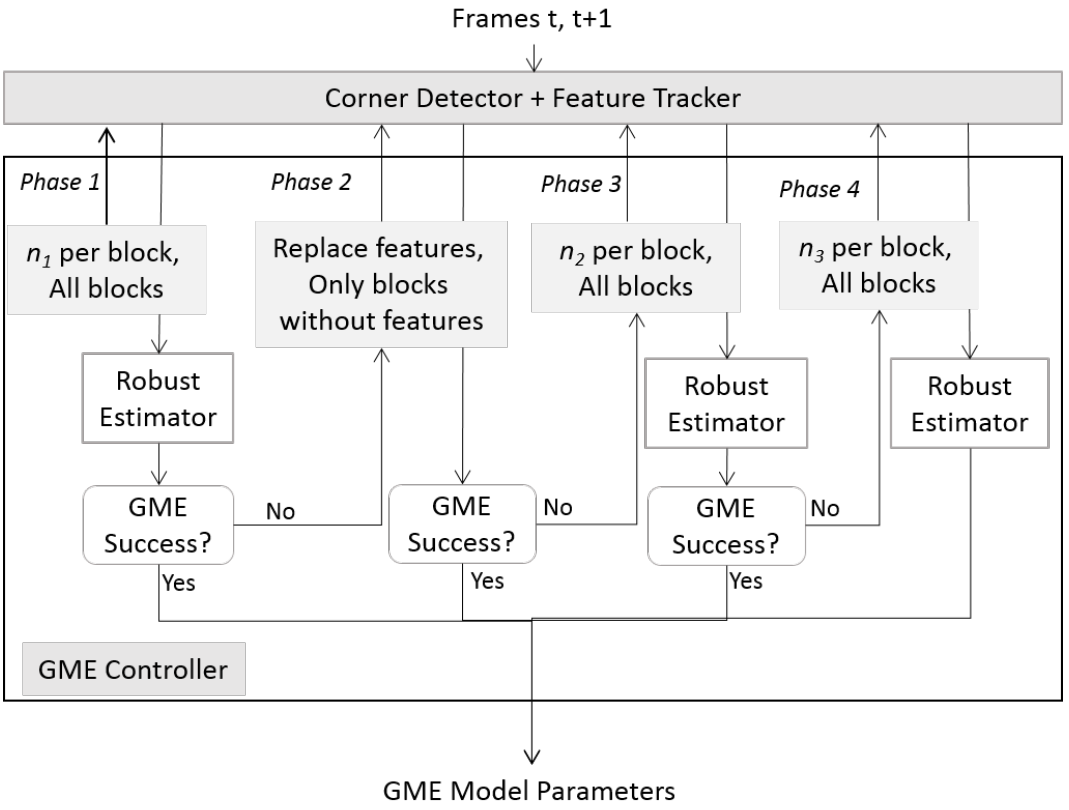

Figure 8.4: Flow of the GME controller showing inputs to the corner detector for each phase of repopulation in sparse-GME; each subsequent phase increases density as $n_1 < n_2 < n_3$

Note that for phase P2, only "supporters" for the current estimation in the blocks that have lost features during tracking or estimation are collected. In this case, the GME model parameters estimated in phase P1 are used and RanSaC is not invoked.

Thus the GME controller monitors the performance of the sparse feature sets for the current scene and adapts the features to the complexity of the scene.

Summary

In this chapter, a unified framework for adaptive and low-complexity global motion estimation is presented, by combining all the individual functional blocks for corner detection, feature tracking and robust estimation. A GME controller performs estimation with a very sparse but well-distributed feature set and evaluates the

estimation by monitoring the number, spread and degree of agreement of the inliers to the estimation. In the event of a failure, additional corners are extracted in specific regions where features have been lost. The corner detector proceeds in a block-based manner, with an optimal threshold for quality of corners being determined in each block of the frame. The adaptive windowing in the feature tracker improves the robustness of each track, enabling a sparse set of features to be used in a wider range of global motions. Combined, the proposed GME pipeline adapts to the image content during corner detection, to the local motion during feature tracking and to the global motion and presence of moving objects during robust estimation. This leads to a low-complexity GME pipeline that predominantly operates on low computation mode and ramps up resources only when the conditions for successful global motion estimation are not met. This makes the proposed low-complexity and adaptive GME realizable on low-resource platforms on surveillance UAVs.

9

Conclusions and Future Work

Conclusions

The contributions in this thesis have led to the development of low-complexity techniques for key functional blocks (corner detection, feature tracking and robust estimation) in global motion estimation (GME) for on-board processing of aerial videos. Also, the proposed techniques can adapt to varying image content and camera motions.

The proposed pruning method (referred to as PP-ER in this thesis) for Shi-Tomasi and Harris corner detectors, leverages on a computationally simpler pruning measure and edge removal step derived from the original corner measures, to rapidly extract high-quality corner candidates. Thresholding the pixels based on their

pruning measure in partial-pruning (PP) discards the non-corner pixels, but releases candidates that also contain edge pixels. The edge-removal (ER) step discards these edge pixels in a compute efficient manner by binning together pixels in the neighbourhood of a candidate, with similar gradient directions, through direct comparisons of the gradient magnitudes. As these corner candidates represent a very small portion of the image, the proposed method results in substantial reduction in the complexity of corner detection by restricting the corner measure computations involving floating point arithmetic to only these candidates. It was shown that incorporating the intensity variations in the neighbourhood of a pixel, as is done in the proposed method, resulted in much higher accuracy compared to existing approaches that selected candidates that were intensity or gradient maxima. Another factor contributing to the accuracy is that the original corner measures of Shi-Tomasi and Harris are still applied to the candidates to rank them and select the final corners. Evaluations on a Nios-II processor without floating point unit, show a speedup in execution time of 48-82% in Shi-Tomasi and 45-81% in Harris corner detection when detecting 300 corners. The computation savings are dependent on the threshold used for pruning and corner measures. If this is set very low, then a large number of pixels are released as candidates and pruning can then become a computational overhead as both the pruning and the corner measure will be applied to the candidate pixels.

The proposed automated thresholding method addresses the problem of setting optimal thresholds for corner detection. Unlike existing methods that employ exhaustive sampling to arrive at an optimal threshold, the proposed method relies on iterative thresholding with non-linear steps in threshold reduction, leading to a small number of trials to release sufficient number of corner candidates. The mask-based non-maximal suppression exploits the batch processing of corner candidates to turn off neighbours of already selected corners, leading to further reduction in corner candidates. The evaluations show a reduction in the corner candidates of 98.2% in Shi-Tomasi and 95.7% in Harris corner detectors. The proposed non-linear steps for lowering the threshold on the corner measure is only a crude fit to the non-linearly increasing distribution of pixels. This distribution depends on

both the image content and the corner detection method. For images with very rich content, the initial threshold used for sampling may itself be too low and release candidates much larger in number than required. In general, it was found that for the same number of required corners, Harris incurs more number of scans than Shi-Tomasi.

The proposed automated thresholding is combined with the PP-ER pruning technique resulting in further reduction in the complexity of corner detection, as the corner measure computation is restricted to the small pool of corner candidates that are released in each iteration. The novel bin mechanism roughly orders and collects the corner candidates, and ensures that all the corner candidates for a given range of corner measure values are collected before corners are extracted using non-maximal suppression. This results in corners which are very similar to the conventional Shi-Tomasi/Harris detectors; the evaluations show an average of 98% match. When the required number of corners and/or the minimum distance is high, the number of candidates that need to be processed also grows. It was found that the ER step and the mask-based NMS are critical in reducing the number of spurious corner candidates from being released in the lower quality ranges. In addition, a heuristic measure of the yield of corners from previous iteration and the remaining number of corners to be extracted, is used for a controlled release of candidates in the lower quality ranges. Evaluations of the proposed method show an average speedup in execution time of 67% for Shi-Tomasi and 51% in Harris corner detectors.

The proposed low-complexity GME method (sparse-GME) employs minimum number of well-distributed sparse features. The method relies on a quick evaluation of GME to facilitate additional features to be injected in a selective and systematic manner. The evaluation strategy monitors the inlier agreement during the estimation to determine if it leads to a successful or failed estimation. As aerial videos predominantly have conducive conditions for successful GME, i.e. simple camera motion and small number of moving targets, GME is achieved with the first-pass sparse feature set itself for the majority of the frames. The 2-RanSaC mode allows the localization of feature correspondences causing estimation failure,

allowing the proposed method to phase out the repopulation by prioritizing coverage improvement to replace these features, over a uniform increase in density of features. When the number of moving objects in the scene increases, leading to loss of feature correspondences, the selective injection to improve coverage results in successful GME, by targeting blocks with moving objects. This postpones a uniform increase in density to the infrequently occurring case of drastic camera motions and/or a high number of moving targets. It was found that for the evaluation method, the error in the tracked and estimated location was better than the error between the actual and estimated intensity patches, as the latter was unable to distinguish between alignment errors and actual physical differences for the features between the two frames. Evaluations on aerial video datasets show that for 95% of the frames, GME with the first pass sparse estimation is performed, while achieving a similar accuracy as the dense set of features for 97% of the cases. For extreme camera motions with rotations or scale changes, it was found that the proposed method was unable to match the accuracy of a dense feature set. This is attributed to the choice of the final dense set which was still not sufficient, and this set needs to be selected by characterizing the worst case scenario for camera motion based on the expected trajectories of the aerial vehicle.

The proposed adaptive windowing for KLT feature tracker monitors the iterations taken by KLT to converge, as an indicator of tracking accuracy to arrive at an optimal window size. By growing the window size to be just sufficient for capturing the displacement, it achieves robustness to distortions by keeping the neighbourhood small, at the same time avoiding unnecessary computations compared to the conventional fixed large window scheme. It was found that although the forward-backward error, commonly used for evaluating feature tracking, was able to detect failures when the window size was too small, it failed when the window size was too large. The proposed method uses the KLT iterations to detect drifting tracks when tracking fails with small window sizes, and avoids the additional KLT in the reverse direction that is required by the forward-backward error. The method tackles the issue of accidental convergence at local minima, which cannot be detected with the KLT iterations, by applying additional checks,

specifically stable convergence for two consecutive window sizes and the forward-backward error. Further, it was shown that the proposed adaptive windowing was suitable to be applied across pyramid levels in a hierarchical KLT implementation, to improve accuracy in the presence of distortions at all levels of the pyramid processing. Evaluations show that the proposed method is able to achieve substantial improvement in accuracy when the distortions are increased, without a significant computational overload, at 7x faster runtime compared to conventional methods. If the average displacement between frames is large due to the speed of the aerial vehicle and/or low frame rate, the proposed method will always need to grow the window size to be able to capture this displacement.

Finally, a unified adaptive framework for low-complexity GME is proposed that combines all the proposed methods for corner detection, tracking and robust estimation. The sparse-GME method creates additional demand for corners and their correspondences dynamically based on the evaluation of the current estimation. This replaces the conventional one-shot GME pipeline in which each block works hard and hands over the results to the next block in the pipeline, with an adaptive and low-complexity GME in which additional computations are incurred in each block only when conditions for successful GME are not met.

Future Work

The following directions for future work are identified based on the contributions in this thesis:

- **Enhance automated thresholding**: An automated thresholding method for corner detection was proposed in this thesis that uses non-linear threshold steps for the threshold sampling scheme. The threshold steps are now fixed. However, the choice of the initial threshold for sampling as well as subsequent threshold steps need to adapt to the number of corner candidates being released in each trial so that they closely match the image content. Also, the threshold steps for Shi-Tomasi need to be different from Harris corner

detector. When pruning is not used, the number of sampling trials to release corner candidates can be reduced by doing a rough sorting and collection of candidates for a range of thresholds instead of a single threshold.

- **Confidence measure for RanSaC algorithm**: In this thesis, the RanSaC algorithm was employed with very sparse features and a method was proposed to evaluate the estimation using a pair of re-projection thresholds. This work can be incorporated within the RanSaC algorithm itself such that given any size of the features correspondences data, RanSaC provides an estimation with an associated confidence measure, that allows the application to then make informed choices in terms of increasing density. This would allow the RanSaC algorithm to be used with very sparse features, in a compute efficient manner.

- **Confidence measure for KLT**: In this thesis, it was shown that the iterations needed to converge can be used as a reliable indicator of KLT success to adapt the KLT window size. Another such indicator is the error between the intensity patches, which can be explored to detect KLT failure. Also, the displacement reported and the window size used by KLT can be analyzed for detecting failures - such as the presence of distortion, when the displacement will be very small compared to the window. All these can be combined to get a confidence measure for the KLT for a given window size, leading to a better evaluation scheme.

- **Systematic parameter selection for adaptive windowing for KLT**: In the proposed adaptive windowing method for KLT, the choice of the parameter values for the minimum and maximum window size, as well as the step sizes can have an impact on the computational complexity. For aerial videos with uniform motion, it will be beneficial to keep a history of the average displacements for previous frames to predict the displacement in the current frame. This can be used to derive the minimum window size in the adaptive window method. The theoretical maximum window size will

depend on the image size and number of pyramid levels. In addition, the frame rate and the vehicle speed can guide the choice of these parameters.

- **Integrated implementation of proposed GME framework**: The computation efficiency of the contributions in this thesis, have been demonstrated as separate modules. An integrated real-time implementation for the adaptive and low-complexity GME as outlined in Chapter 8 needs to be completed. The intermediate data for the individual algorithms - such as the pruning measures and corner measure bins during corner detection and image pyramid representation during feature tracking - need to be maintained until the GME is completed. Data representation for efficient memory usage needs to be explored. It was demonstrated that the proposed pruning technique for corner detection leads to benefits on platforms without floating point units. Further studies are to be undertaken to evaluate performance benefits in terms of power consumption in comparison with other existing work such as [68, 71] on representative embedded platforms. In addition, the integrated GME framework needs to be employed on a UAV platform to evaluate its performance during flight.

- **Mosaic-based compression (MBC) for aerial videos**: MBC or ROI coding can achieve very high compression rates for aerial videos. As described in Chapter 2, GME is the first step but is also a bottleneck in terms of computations. The real-time implementation of the proposed low-complexity GME framework needs to be applied to an ROI coding pipeline and evaluated for aerial video compression on embedded platforms, compared to the conventional GME methods. An important consideration is to characterize the accuracy of GME required for efficient compression. For instance, prior to compression, the moving targets are extracted as ROIs and the accuracy of this step will in turn depend on the accuracy of the GME. This can then guide the appropriate selection of the parameters in the sparse-GME.

- **Robust KLT for motion discontinuities**: The adaptive windowing method

for KLT proposed in this thesis is applicable to any scenario where the uniform translation motion model is violated for a large search window size. An example is object tracking in sports videos where the boundary of the objects experience severe motion discontinuities as the background is moving with the camera and the object is also moving independently. Further evaluations are needed to assess if the adaptive windowing can be applied to improve the accuracy of tracking for these objects.

Appendices

A

Description of the Kanade-Lucas-Tomasi (KLT) feature tracker

In this appendix, the Kanade-Lucas-Tomasi (KLT) feature tracking algorithm is described. The main objective of KLT is to register a feature patch from one frame to its new location in the successive frame thereby providing the feature correspondence. As is mentioned in [54], KLT is faster because it examines far fewer potential matches to register the feature patch, compared to other techniques. This is achieved by considering the image as a 2D signal and employing the spatial intensity gradient of the image to find a good match using a type of Newton-Raphson iteration. The intuitive reasoning behind the KLT algorithm is presented for the 1D case in Section A.1. This is followed by the formulation for

2D case, which is applied to feature tracking in images [54] in Section A.2 where the core computation in KLT is derived.

1-dimensional Case

The basic problem of feature correspondence can be described in 1D as shown in Fig. A.1 (a). Consider a curve $G(x)$ which is shifted horizontally by a disparity of h resulting in $F(x)$ such that $G(x) = F(x + h)$. In the 1D sense, KLT aims to find the disparity h so that the two curves can be aligned.

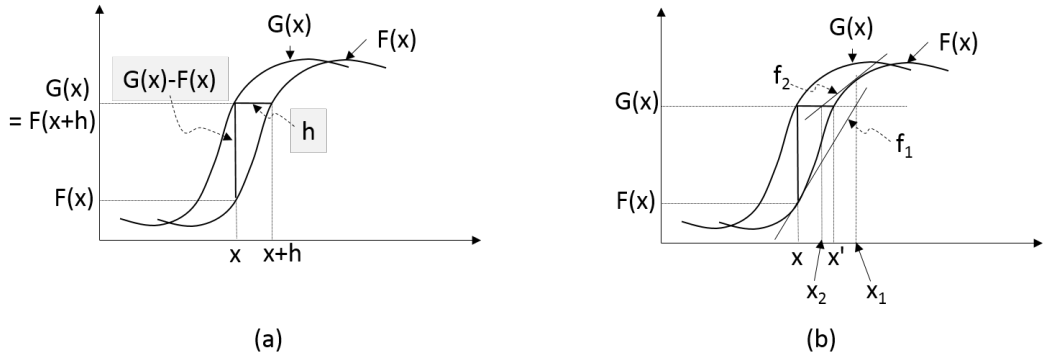

Figure A.1: KLT for 1D case (a) Registering two signals: finds the disparity h such that $(x + h)$ is the correspondence for x (b) Newton-Raphson type iterations in KLT to find the correspondence (x, x') where $x' = x + h$

If the disparity h is small, linear approximation can be used in the neighbourhood of x for $F(x)$ as illustrated in Fig. A.1 (a) giving:

$$F'(x) \approx (F(x + h) - F(x))/h = (G(x) - F(x))/h \tag{A.1}$$

Therefore, disparity h can be solved by:

$$h \approx (G(x) - F(x))/F'(x) \tag{A.2}$$

The approximation to h above depends on x, and therefore it is reasonable to combine the estimates of h in the neighbourhood of x by averaging them as:

$$h = \frac{\sum_x (G(x) - F(x))/F'(x)}{\sum_x 1} \tag{A.3}$$

Now that the disparity h has been estimated, $F(x)$ can be moved by this estimate to $x+h$, and this procedure can be repeated in a Newton-Raphson style iterations as follows:

$$\begin{aligned} h_0 &= 0, \\ h_{k+1} &= h_k + \frac{\sum_x (G(x) - F(x+h_k))/F'(x+h_k)}{\sum_x 1} \end{aligned} \tag{A.4}$$

In Fig. A.1 (b), these iterations are illustrated. Assume the estimation of disparity h starts at x with $h_0 = 0$. h_1 is found by linearly approximating with the line f_1 giving the new location $x_1 = x + h_1$. The estimation is now performed at x_1 by linearly approximating with line f_2 that gives the new disparity h_2 providing the next location at $x_2 = x_1 + h_2$. This process continues and converges to $x' = x + h$ provided h is small.

It is clear that the search for the disparity h is not brute force and exhaustive but is guided by the spatial intensity gradient $F'(x)$ and the intensity difference $G(x) - F(x)$ in an iterative manner leading to reduction in the potential matches.

2-dimensional Case: Feature Tracking

The formulation for finding the disparity h in the 1D case can be applied to 2D case (for finding the correspondence of a feature patch) where $\mathbf{x}$ and $\mathbf{h}$ are *2*-dimensional row vectors. The linear approximation in A.1 can be written as:

$$F(\mathbf{x} + \mathbf{h}) \approx F(\mathbf{x}) + \mathbf{h}\frac{\partial}{\partial \mathbf{x}} F(\mathbf{x}) \tag{A.5}$$

An alternative derivation is used for h where the disparity h must minimize the $\mathbf{L_2}$ norm measure of the difference between the two patches E:

$$E = \sum_x (F(\mathbf{x}+\mathbf{h}) - G(\mathbf{x}))^2 \tag{A.6}$$

To minimize E, its derivative is set to 0 as:

$$\begin{aligned}
\mathbf{0} &= \frac{\partial}{\partial \mathbf{h}} E \\
&\approx \frac{\partial}{\partial \mathbf{h}} \sum_{\mathbf{x}} [F(\mathbf{x}) + \mathbf{h}\frac{\partial}{\partial \mathbf{x}} F(\mathbf{x}) - G(\mathbf{x})]^2 \\
&= \sum_{\mathbf{x}} 2\frac{\partial F}{\partial \mathbf{x}} [F(\mathbf{x}) + \mathbf{h}\frac{\partial}{\partial \mathbf{x}} F(\mathbf{x}) - G(\mathbf{x})]
\end{aligned}$$

Solving for h gives:

$$\mathbf{h} \approx \left[\sum_{\mathbf{x}} \left(\frac{\partial F}{\partial \mathbf{x}}\right)^T [G(\mathbf{x}) - F(\mathbf{x})]\right] \left[\sum_{\mathbf{x}} \left(\frac{\partial F}{\partial \mathbf{x}}\right)^T \left(\frac{\partial F}{\partial \mathbf{x}}\right)\right]^{-1} \tag{A.7}$$

If $\mathbf{x} = (x, y)$ and $\partial F/\partial \mathbf{x} = [\partial F/\partial x, \partial F/\partial y]^T = [F_x, F_y]^T$, then a spatial gradient matrix M at (x, y) can be expressed as:

$$\begin{aligned}
M &= \sum_{\mathbf{x}} \left(\frac{\partial F}{\partial \mathbf{x}}\right)^T \left(\frac{\partial F}{\partial \mathbf{x}}\right) \\
&= \sum_{(x,y)\in W} \begin{bmatrix} {F_x}^2 & F_x F_y \\ F_x F_y & {F_y}^2 \end{bmatrix}
\end{aligned} \tag{A.8}$$

Also an the image difference δF at (x, y) can be defined as:

$$\delta F = G(\mathbf{x}) - F(\mathbf{x}) \tag{A.9}$$

An image mismatch vector $\bar{b}$ is defined using the image difference δF, as:

$$\begin{aligned}\bar{b} &= \sum_{\mathbf{x}} \left(\frac{\partial F}{\partial \mathbf{x}}\right)^T [G(\mathbf{x}) - F(\mathbf{x})] \\ &= \sum_{\mathbf{x}} \left(\frac{\partial F}{\partial \mathbf{x}}\right)^T [\delta F] \\ &= \sum_{(x,y)\in W} \begin{bmatrix} \delta F.F_x \\ \delta F.F_y \end{bmatrix}\end{aligned} \tag{A.10}$$

Substituting the spatial gradient matrix M in A.8 and image mismatch vector $\bar{b}$ in A.10 into A.7, the core computation for estimating the disparity $\mathbf{h}$ can be derived as:

$$\mathbf{h}^T \approx M^{-1}\bar{b} \tag{A.11}$$

The above estimation of disparity $\mathbf{h}$ is performed in an iterative manner as in A.4. As $F(\mathbf{x})$ and $G(\mathbf{x})$ are assumed to be spatially shifted versions of each other, the computations for the spatial gradient matrix M can be performed once for each location (x, y) using $G(\mathbf{x})$. In each iteration k, the image mismatch vector $\bar{b_k}$ is recomputed as it depends on the image difference δF_k:

$$\bar{b_k} = \sum_{(x,y)\in W} \begin{bmatrix} \delta F_k.F_x \\ \delta F_k.F_y \end{bmatrix}$$

$$\delta F_k = G(\mathbf{x}) - F(\mathbf{x} + \mathbf{h_{k-1}})$$

Therefore, the core computation in each iteration k is given by:

$$\mathbf{h_k} = \mathbf{h_{k-1}} + M^{-1}\bar{b_k} \tag{A.12}$$

The pseudo code of pyramidal implementation for the KLT feature tracking algorithm is provided in [77].

B

Experimental Setup with Nios-II

The Nios-II embedded soft core processor was used to demonstrate the reduction in the computational complexity, achieved with the proposed pruning (Chapter 3) and automated thresholding (Chapter 5), for the Shi-Tomasi/Harris corner detection algorithms. As the Nios-II is a configurable soft IP core, it allows the addition and removal of on-chip features. Of particular interest was the floating point unit (FPU) and data cache. The execution time was measured by enabling and disabling the floating point hardware and the data cache. The details of the experimental setup with Nios-II processor have been provided below:

1. *FPGA development board*: Altera's Cyclone III FPGA development kit [131] with the Cyclone III EP3C120F780 FPGA as in Fig. B.1 was used for the evaluations. It supports 256-Mbyte dual-channel DDR2 SDRAM, 8-Mbyte Sync-SRAM and 64-Mbyte flash.

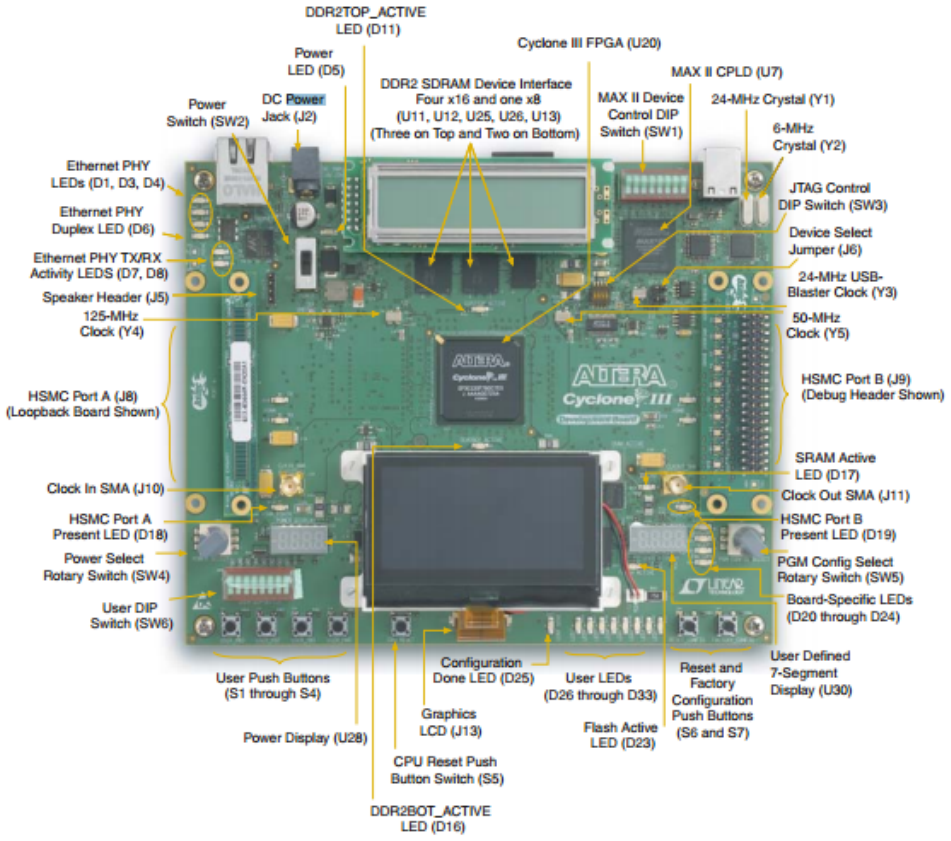

Figure B.1: Top view of Cyclone III FPGA development board

2. *Nios-II on-chip configuration*: The Quartus II v9.1 was used to create the hardware design files to configure the Nios-II soft core processor [132] on the FPGA, using the SOPC builder tool. Nios-II/f (fast) core was used for the evaluations as shown in Fig. B.2. The configuration setup is described below:

   - System frequency: 75 MHz
   - On-chip RAM: 12 Kb
   - On-chip instruction cache: 32 Kb
   - On-chip data cache: None (disabled) or 32 Kb (enabled)
   - Floating point hardware: enabled/disabled

3. *Software tools*: The Nios-II embedded development suite (EDS) v9.1 was used to compile, download and execute the C code on the FPGA. The C

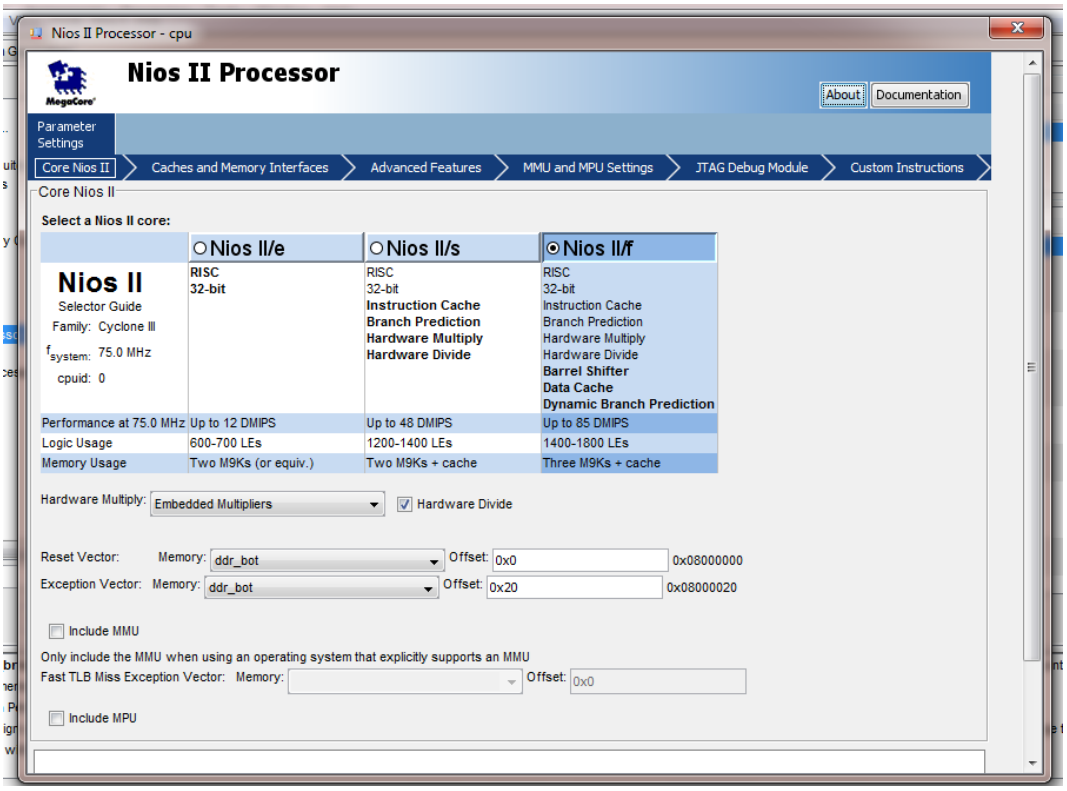

Figure B.2: Nios-II/f (fast) core configuration

code was executed bare metal on the Nios-II, without any operating system. The steps for executing the C code on the Nios-II [133] are listed below:

(a) A Nios-II application and board support package (BSP) project are created from template by linking them to the hardware design (.sopc) file.

(b) The C source codes are added to the application project.

(c) The read-only zip file system (*rozipfs*) is setup as a software package in the BSP project by linking it to the on-chip flash memory.

(d) The Quartus flash programmer tool is used to download the image files used by the C code into the flash memory.

(e) The projects are compiled and built. The Quartus tool is used to connect to the Cyclone III board, to download and execute the executable file.

References

[1] Nirmala Ramakrishnan, Meiqing Wu, Siew-Kei Lam, and Thambipillai Srikanthan. Enhanced low-complexity pruning for corner detection. *Journal of Real-Time Image Processing*, 2(1):197–213, 2016.

[2] Meiqing Wu, Nirmala Ramakrishnan, Siew-Kei Lam, and Thambipillai Srikanthan. Low-complexity pruning for accelerating corner detection. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1684–1687. IEEE, 2012.

[3] Nirmala Ramakrishnan, Meiqing Wu, Siew-Kei Lam, and Thambipillai Srikanthan. Automated thresholding for low-complexity corner detection. In *IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 97–103. IEEE, 2014.

[4] Nirmala Ramakrishnan, Meiqing Wu, Siew-Kei Lam, and Thambipillai Srikanthan. Mask-based non-maximal suppression with iterative pruning for low-complexity corner detection. In *IEEE 14th International Symposium on Integrated Circuits (ISIC)*, pages 368–371. IEEE, 2014.

[5] Nirmala Ramakrishnan, Thambipillai Srikanthan, Siew Kei Lam, and Gauri Ravindra Tulsulkar. Adaptive window strategy for high-speed and robust KLT feature tracker. In *Pacific-Rim Symposium on Image and Video Technology (PSIVT)*, pages 355–367. Springer, 2015.

[6] Rita Cucchiara and Giovanni Gualdi. Mobile video surveillance systems: An architectural overview. In *Mobile Multimedia Processing*, pages 89–109. Springer, 2010.

[7] Hovering over UAV risks. https://www.zurichcanada.com/en-ca/knowledge-hub/articles/2015/10/hovering-over-opportunities. Accessed: 2016-03-26.

[8] UAV worldwide market forecast. http://www.prnewswire.com/news-releases/teal-group-predicts-worldwide-uav-production-will-total-93-billion-in-its-2015-uav-market-profile-and-forecast-300128745.html. Accessed: 2016-03-26.

[9] Pixhawk open hardware project for UAVs autopilot. https://pixhawk.org/. Accessed: 2016-11-01.

[10] Precision hawk UAV platform. http://www.precisionhawk.com/. Accessed: 2016-04-12.

[11] DJI Phantom 3 drone. http://www.dji.com/product/phantom-3-pro. Accessed: 2016-03-25.

[12] Fotokite. http://fotokite.com/wp-content/uploads/2016/01/160113_Brochure_Fotokite_Pro.pdf. Accessed: 2016-03-25.

[13] UAVs for early diagnosis of HIV in infants. http://www.unicef.org/media/media_90462.html. Accessed: 2016-03-23.

[14] Amazon Prime Air. http://www.amazon.com/b?node=8037720011. Accessed: 2016-03-31.

[15] Ken Whitehead, Chris H Hugenholtz, Stephen Myshak, Owen Brown, Adam LeClair, Aaron Tamminga, Thomas E Barchyn, Brian Moorman, and Brett Eaton. Remote sensing of the environment with small unmanned aircraft systems (UASs), part 2: scientific and commercial applications. *Journal of Unmanned Vehicle Systems*, 2(3):86–102, 2014.

[16] Ali Haydar Göktoğan and Salah Sukkarieh. Autonomous remote sensing of invasive species from robotic aircraft. In *Handbook of Unmanned Aerial Vehicles*, pages 2813–2834. Springer, 2015.

[17] Stuart M Adams and Carol J Friedland. A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management. In *9th International Workshop on Remote Sensing for Disaster Response*, 2011.

[18] Sivakumar Rathinam, Zu Whan Kim, and Raja Sengupta. Vision-based monitoring of locally linear structures using an unmanned aerial vehicle. *Journal of Infrastructure Systems*, 14(1):52–63, 2008.

[19] Konstantinos Kanistras, Goncalo Martins, Matthew J Rutherford, and Kimon P Valavanis. Survey of unmanned aerial vehicles (UAVs) for traffic monitoring. In *Handbook of Unmanned Aerial Vehicles*, pages 2643–2666. Springer, 2015.

[20] Chad C Haddal and Jeremiah Gertler. Homeland security: Unmanned aerial vehicles and border surveillance. Technical Report Congressional Research Service RS21698, 2010.

[21] Conservation drones. `http://conservationdrones.org/`. Accessed: 2016-03-23.

[22] Ken Whitehead and Chris H Hugenholtz. Remote sensing of the environment with small unmanned aircraft systems (UASs), part 1: A review of progress and challenges. *Journal of Unmanned Vehicle Systems*, 2(3):69–85, 2014.

[23] Giuseppe Loianno, Gareth Cross, Chao Qu, Yash Mulgaonkar, Joel A Hesch, and Vijay Kumar. Flying smartphones: Automated flight enabled by consumer electronics. *Robotics & Automation Magazine, IEEE*, 22(2):24–32, 2015.

[24] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor. *Proc. of Robotics: Science and Systems (RSS)*, 2013.

[25] Shoaib Ehsan and Klaus D McDonald-Maier. On-board vision processing for small UAVs: Time to rethink strategy. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 75–81. IEEE, 2009.

[26] Dries Hulens, Toon Goedemé, and Jon Verbeke. How to choose the best embedded processing platform for on-board UAV image processing? In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pages 1–10. IEEE, 2015.

[27] Honghai Liu, Shengyong Chen, and Naoyuki Kubota. Intelligent video systems and analytics: A survey. *IEEE Transactions on Industrial Informatics*, 9(3):1222–1233, 2013.

[28] Weiming Hu, Tieniu Tan, Liang Wang, and Steve Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(3):334–352, 2004.

[29] Hannah M Dee and Sergio A Velastin. How close are we to solving the problem of automated visual surveillance? *Machine Vision and Applications*, 19(5-6):329–343, 2008.

[30] Datamapper aerial data analytics. `https://www.datamapper.com/`. Accessed: 2016-04-12.

[31] Aryo Wiman Nur Ibrahim, Pang Wee Ching, GL Gerald Seet, WS Michael Lau, and Witold Czajewski. Moving objects detection and tracking framework for UAV-based surveillance. In *Fourth Pacific-Rim Symposium on Image and Video Technology (PSIVT)*, pages 456–461. IEEE, 2010.

[32] Michael Teutsch and Wolfgang Krüger. Detection, segmentation, and tracking of moving objects in UAV videos. In *Ninth International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, pages 313–318. IEEE, 2012.

[33] Isaac Cohen and Gerard Medioni. Detecting and tracking moving objects for video surveillance. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2. IEEE, 1999.

[34] Marko Heikkila and Matti Pietikainen. A texture-based method for modeling the background and detecting moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):657–662, 2006.

[35] AG Amitha Perera, Chukka Srinivas, Anthony Hoogs, Glen Brooksby, and Wensheng Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 666–673. IEEE, 2006.

[36] Jia Wei Tang, Nasir Shaikh-Husin, Usman Ullah Sheikh, and MN Marsono. FPGA-based real-time moving target detection system for unmanned aerial vehicle application. *International Journal of Reconfigurable Computing*, 2016, 2016.

[37] T Klassen. The UAV video problem: using streaming video with unmanned aerial vehicles. *Military and Aerospace Electronics*, 20(7), 2009.

[38] Malavika Bhaskaranand and Jerry D Gibson. Low-complexity video encoding for UAV reconnaissance and surveillance. In *Military Communications Conference (MILCOM)*, pages 1633–1638. IEEE, 2011.

[39] Holger Meuel, Marco Munderloh, Matthias Reso, and Jörn Ostermann. Mesh-based piecewise planar motion compensation and optical flow clustering for ROI coding. *APSIPA Transactions on Signal and Information Processing*, 4, 2015.

[40] Michal Irani, Steve Hsu, and P Anandan. Video compression using mosaic representations. *Signal Processing: Image Communication*, 7(4):529–552, 1995.

[41] Eren Soyak, Sotirios A Tsaftaris, and Aggelos K Katsaggelos. Low-complexity tracking-aware H.264 video compression for transportation surveillance. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1378–1389, 2011.

[42] Frédéric Dufaux and Fabrice Moscheni. Background mosaicking for low bit rate video coding. In *Proceedings of International Conference on Image Processing (ICIP)*, volume 1, pages 673–676. IEEE, 1996.

[43] Michal Irani, P al Anandan, Jim Bergen, Rakesh Kumar, and Steve Hsu. Efficient representations of video sequences and their applications. *Signal Processing: Image Communication*, 8(4):327–351, 1996.

[44] Malavika Bhaskaranand and Jerry D Gibson. Global motion assisted low complexity video encoding for UAV applications. *IEEE Journal of Selected Topics in Signal Processing*, 9(1):139–150, 2015.

[45] Cesario Vincenzo Angelino, Luca Cicala, Marco De Mizio, Paolo Leoncini, Enrico Baccaglini, Marco Gavelli, Nadir Raimondo, and Roberto Scopigno. Sensor aided H.264 video encoder for UAV applications. In *Picture Coding Symposium (PCS)*, pages 173–176. IEEE, 2013.

[46] Pascual Campoy, Juan F Correa, Ivan Mondragón, Carol Martínez, Miguel Olivares, Luis Mejías, and Jorge Artieda. Computer vision onboard UAVs for civilian tasks. In *Unmanned Aircraft Systems*, pages 105–135. Springer, 2008.

[47] Mark D Pritt and Kevin J LaTourette. Aircraft navigation by means of image registration. In *Applied Imagery Pattern Recognition Workshop (AIPR): Sensing for Control and Augmentation*, pages 1–6. IEEE, 2013.

[48] Feng Lin, Xiangxu Dong, Ben M Chen, Kai-Yew Lum, and Tong H Lee. A robust real-time embedded vision system on an unmanned rotorcraft for ground target following. *IEEE Transactions on Industrial Electronics*, 59(2):1038–1049, 2012.

[49] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends® in Computer Graphics and Vision*, 3(3):177–280, 2008.

[50] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600. IEEE, 1994.

[51] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, pages 147–151, 1988.

[52] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV)*, pages 430–443. Springer, 2006.

[53] Stephen M Smith and J Michael Brady. SUSAN: A new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997.

[54] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[55] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[56] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *European Conference on Computer vision (ECCV)*, pages 404–417. Springer, 2006.

[57] Dirk Farin and Peter H. N. de With. Evaluation of a feature-based global-motion estimation system. *Visual Communications and Image Processing*, 5960, 2005.

[58] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

[59] Han Wang and Michael Brady. Real-time corner detection algorithm for motion estimation. *Image and Vision Computing*, 13(9):695–703, 1995.

[60] Christopher Claus, Robert Huitl, Joachim Rausch, and Walter Stechele. Optimizing the SUSAN corner detection algorithm for a high speed FPGA implementation. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 138–145. IEEE, 2009.

[61] Henrik Aanæs, Anders Lindbjerg Dahl, and Kim Steenstrup Pedersen. Interesting interest points. *International Journal of Computer Vision*, 97(1):18–35, 2012.

[62] Chih-Chi Cheng, Chia-Hua Lin, Chung-Te Li, and Liang-Gee Chen. iVisual: An intelligent visual sensor SoC with 2790 fps CMOS image sensor and 205 GOPS/W vision processor. *IEEE Journal of Solid-State Circuits*, 44(1):127–135, 2009.

[63] Benedikt Dietrich. Design and implementation of an FPGA-based stereo vision system for the EyeBot M6. Master's thesis, University of Western Australia, 2009.

[64] Tarik Saidani, Lionel Lacassagne, Samir Bouaziz, and Taj Muhammad Khan. Parallelization strategies for the points of interests algorithm on the cell processor. In *Parallel and distributed processing and applications*, pages 104–112. Springer, 2007.

[65] Fouzhan Hosseini, Amir Fijany, and Jean-Guy Fontaine. Highly parallel implementation of Harris Corner detector on CSX SIMD architecture. In *Euro-Par Parallel Processing Workshops*, pages 137–144. Springer, 2011.

[66] Stephane Piskorski, Lionel Lacassagne, Samir Bouaziz, and Daniel Etiemble. Customizing CPU instructions for embedded vision systems. In *International Workshop on Computer Architecture for Machine Perception and Sensing (CAMP)*, pages 59–64. IEEE, 2006.

[67] Beau J Tippetts, Dah-Jye Lee, and James K Archibald. An on-board vision sensor system for small unmanned vehicle applications. *Machine Vision and Applications*, 23(3):403–415, 2012.

[68] X Benedetti and Pietro Perona. Real-time 2-D feature detection on a reconfigurable computer. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–593. IEEE, 1998.

[69] Lucas Teixeira, Waldemar Celes Filho, and Marcelo Gattass. Accelerated corner-detector algorithms. In *19th British Machine Vision Conference (BMVC)*, pages 625–634, 2008.

[70] Sudipta N Sinha, Jan-Michael Frahm, Marc Pollefeys, and Yakup Genc. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications*, 22(1):207–217, 2011.

[71] Pradip Mainali, Qiong Yang, Gauthier Lafruit, Luc Van Gool, and Rudy Lauwereins. Robust low complexity corner detector. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(4):435–445, 2011.

[72] Sultan Alkaabi and Farzin Deravi. Candidate pruning for fast corner detection. *Electronics Letters*, 40(1):18–19, 2004.

[73] Mosalam Ebrahimi and Walterio W Mayol-Cuevas. Adaptive sampling for feature detection, tracking, and recognition on mobile platforms. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1467–1475, 2011.

[74] Yitzhak Yitzhaky and Eli Peli. A method for objective edge detection evaluation and detector parameter selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):1027–1033, 2003.

[75] Stephen P DelMarco, Victor Tom, and Helen F Webb. A theory of automatic parameter selection for feature extraction with application to feature-based multisensor image registration. *IEEE Transactions on Image Processing*, 16(11):2733–2742, 2007.

[76] OpenCV Dynamic Adapted Feature Detector Implementation. `http://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_feature_detectors.html#dynamicadaptedfeaturedetector`. Accessed: 2015-12-07.

[77] Jean-Yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. Technical report, Microsoft Research Labs, Intel Corporation, 2000.

[78] Supannee Tanathong and Impyeong Lee. Translation-based KLT tracker under severe camera rotation using gps/ins data. *IEEE Geoscience and Remote Sensing Letters*, 11(1):64–68, 2014.

[79] Jean-Yves Bouguet. Pyramidal implementation of the affine Lucas Kanade feature tracker: description of the algorithm. *Intel Corporation*, 5:1–10, 2001.

[80] Jun-Sik Kim, Myung Hwangbo, and Takeo Kanade. Realtime affine-photometric KLT feature tracker on GPU in CUDA framework. In *12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 886–893. IEEE, 2009.

[81] Myung Hwangbo, Jun Sik Kim, and Takeo Kanade. Inertial-aided KLT feature tracking for a moving camera. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1909–1916. IEEE, 2009.

[82] Masatoshi Okutomi and Takeo Kanade. A locally adaptive window for signal matching. *International Journal of Computer Vision*, 7(2):143–162, 1992.

[83] Pradip Mainali, Qiong Yang, Gauthier Lafruit, Rudy Lauwereins, and Luc Van Gool. Robust low complexity feature tracking. In *International Conference on Image Processing (ICIP)*, pages 829–832. IEEE, 2010.

[84] Juan C SanMiguel, Andrea Cavallaro, and José M Martínez. Adaptive online performance evaluation of video trackers. *IEEE Transactions on Image Processing*, 21(5):2812–2823, 2012.

[85] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *20th International Conference on Pattern Recognition (ICPR)*, pages 2756–2759. IEEE, 2010.

[86] Sameer Sheorey, Shalini Keshavamurthy, Huili Yu, Hieu Nguyen, and Clark N Taylor. Uncertainty estimation for KLT tracking. In *Asian Conference on Computer Vision-Workshops*, pages 475–487. Springer, 2014.

[87] Iain Matthews, Takahiro Ishikawa, and Simon Baker. The template update problem. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):810–815, 2004.

[88] Timo Zinßer, Christoph Gräßl, and Heinrich Niemann. Efficient feature tracking for long video sequences. In *Pattern Recognition*, pages 326–333. Springer, 2004.

[89] Tiziano Tommasini, Andrea Fusiello, Emanuele Trucco, and Vito Roberto. Making good features track better. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 178–183. IEEE, 1998.

[90] Michal Irani and P Anandan. About direct methods. In *Vision Algorithms: Theory and Practice*, pages 267–277. Springer, 1999.

[91] Aljoscha Smolić, Thomas Sikora, and Jens-Rainer Ohm. Long-term global motion estimation and its application for sprite coding, content description, and segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1227–1242, 1999.

[92] Yosi Keller and Amir Averbuch. Fast gradient methods based on global motion estimation for video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(4):300–309, 2003.

[93] Frederic Dufaux and Janusz Konrad. Efficient, robust, and fast global motion estimation for video coding. *IEEE Transactions on Image Processing*, 9(3):497–501, 2000.

[94] Adrien Bartoli, Navneet Dalal, and Radu Horaud. Motion panoramas. *Computer Animation and Virtual Worlds*, 15(5):501–517, 2004.

[95] Philip HS Torr and Andrew Zisserman. Feature based methods for structure and motion estimation. In *Vision Algorithms: Theory and Practice*, pages 278–294. Springer, 1999.

[96] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[97] Hussein Alzoubi and W David Pan. Very fast global motion estimation using partial data. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages I–1189–I–1192. IEEE, 2007.

[98] Md Nazmul Haque, Moyuresh Biswas, Mark R Pickering, and Michael R Frater. A low-complexity image registration algorithm for global motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(3):426–433, 2012.

[99] Ryan C Jones, Daniel DeMenthon, and David S Doermann. Building mosaics from video using MPEG motion vectors. In *International Conference on Multimedia (Part 2)*, pages 29–32. ACM, 1999.

[100] Michael Tok, Alexander Glantz, Marina Georgia Arvanitidou, Andreas Krutz, and Thomas Sikora. Compressed domain global motion estimation using the Helmholtz tradeoff estimator. In *17th International Conference on Image Processing (ICIP)*, pages 777–780. IEEE, 2010.

[101] Ramon L Felip, Lluís Barceló, Xavier Binefa, and John R Kender. Robust dominant motion estimation using MPEG information in sport sequences. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(1):12–22, 2008.

[102] Martin Haller, Andreas Krutz, and Thomas Sikora. Evaluation of pixel-and motion vector-based global motion estimation for camera motion characterization. In *10th Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, pages 49–52. IEEE, 2009.

[103] David Capel and Andrew Zisserman. Automated mosaicing with super-resolution zoom. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 885–891. IEEE, 1998.

[104] Jen-Chi Huang and Wen-Shyong Hsieh. Automatic feature-based global motion estimation in video sequences. *IEEE Transactions on Consumer Electronics*, 50(3):911–915, 2004.

[105] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, School of Computer Science, Carnegie Mellon University, 1991.

[106] Pietro Azzari, Luigi Di Stefano, and Stefano Mattoccia. An evaluation methodology for image mosaicing algorithms. In *Advanced Concepts for Intelligent Vision Systems*, pages 89–100. Springer, 2008.

[107] Michael Tok, Alexander Glantz, Andreas Krutz, and Thomas Sikora. Feature-based global motion estimation using the Helmholtz principle. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1561–1564. IEEE, 2011.

[108] James R Bergen, Patrick Anandan, Keith J Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *European Conference on Computer Vision (ECCV)*, pages 237–252. Springer, 1992.

[109] Mahesh Ramachandran and Rama Chellappa. Stabilization and mosaicing of airborne videos. In *International Conference on Image Processing (ICIP)*, pages 345–348. IEEE, 2006.

[110] Xiaoming Chen, Zhendong Zhao, Ahmad Rahmati, Ye Wang, and Lin Zhong. Sensor-assisted video encoding for mobile devices in real-world environments. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(3):335–349, 2011.

[111] Shoaib Ehsan, Nadia Kanwal, Adrian F Clark, and Klaus D McDonald-Maier. Measuring the coverage of interest point detectors. In *Image Analysis and Recognition*, pages 253–261. Springer, 2011.

[112] Ondřej Chum and Jiří Matas. Randomized RANSAC with Td, d test. In *British Machine Vision Conference (BMVC)*, volume 2, pages 448–457, 2002.

[113] Ondřej Chum and Jiří Matas. Matching with PROSAC-progressive sample consensus. In *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 220–226. IEEE, 2005.

[114] David Nistér. Preemptive RANSAC for live structure and motion estimation. *Machine Vision and Applications*, 16(5):321–329, 2005.

[115] Daniel J Mirota, Masaru Ishii, and Gregory D Hager. Vision-based navigation in image-guided interventions. *Annual review of biomedical engineering*, 13:297–319, 2011.

[116] Adam Schmidt, Marek Kraft, and Andrzej Kasiński. An evaluation of image feature detectors and descriptors for robot navigation. In *Computer Vision and Graphics*, pages 251–259. Springer, 2010.

[117] Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *International Journal of Computer Vision*, 94(3):335–360, 2011.

[118] Arturo Gil, Oscar Martinez Mozos, Monica Ballesta, and Oscar Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual SLAM. *Machine Vision and Applications*, 21(6):905–920, 2010.

[119] Affine Covariant Features. `http://www.robots.ox.ac.uk/~vgg/research/affine/`. Accessed: 2015-12-04.

[120] Nios II Processor: The World's Most Versatile Embedded Processor. `http://www.altera.com/devices/processor/nios2/ni2-index.html`. Accessed: 2015-12-04.

[121] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3153–3160. IEEE, 2011.

[122] Shizhe Shen, Xiaolong Zhang, and Wei Heng. Auto-adaptive Harris Corner detection algorithm based on block processing. In *International Symposium on Signals Systems and Electronics (ISSSE)*, volume 1, pages 1–4. IEEE, 2010.

[123] Flore Faille. A fast method to improve the stability of interest point detection under illumination changes. In *International Conference on Image Processing (ICIP)*, volume 4, pages 2673–2676. IEEE, 2004.

[124] Diego Rodríguez and Nabil Aouf. Robust Harris-SURF features for robotic vision based navigation. In *13th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1160–1165. IEEE, 2010.

[125] Gioacchino Vino and Angel D Sappa. Revisiting Harris Corner detector algorithm: A gradual thresholding approach. In *Image Analysis and Recognition*, pages 354–363. Springer, 2013.

[126] Raw images from Curiosity rover, Mars science laboratory. `http://mars.jpl.nasa.gov/msl/multimedia/raw/`. Accessed: 2015-12-07.

[127] Shoaib Ehsan, Adrian F Clark, and Klaus D McDonald-Maier. Rapid online analysis of local feature detectors and their complementarity. *Sensors*, 13:10876–10907, 2013.

[128] Erkan Bostanci. Enhanced image feature coverage: Key-point selection using genetic algorithms. *arXiv preprint arXiv:1512.03155*, 2015.

[129] RanSaC in 2011 (30 years after). `http://www.imgfsr.com/CVPR2011/Tutorial6/RANSAC_CVPR2011.pdf`. Accessed: 2016-05-03.

[130] Wide Area Aerial Surveillance Dataset (CLIF 2006). `https://www.sdms.afrl.af.mil/index.php?collection=clif2006`. Accessed: 2016-05-03.

[131] Cyclone III FPGA development board: Reference manual. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/rm_cycloneiii_dev_kit_host_board.pdf`. Accessed: 2016-11-19.

[132] Nios-II classic processor reference guide. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf`. Accessed: 2016-11-19.

[133] Nios-II software development tutorial. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/tt/tt_my_first_nios_sw.pdf`. Accessed: 2016-11-19.