OSMANIA UNIVERSITY

VIDEO STABILIZATION USING POINT FEATURE MATCHING AND MATLAB

PROJECT REPORT

Submitted

In the partial fulfillment of the award of degree

Bachelor of Engineering
IN
ELECTRONICS AND COMMUNICATION ENGINEERING
BY

S.YASHASWI RANGA	**2451-16-735-126**
P.SUMAN	**2451-16-735-138**
R.AMARTYA VARMA	**2451-16-735-140**

Under the Guidance of
Mrs. K. KAVYA
(Assistant Professor, Department of ECE)

MATURI VENKATA SUBBA RAO (MVSR) ENGINEERING COLLEGE
(Sponsored by Matrusri Education Society, Estd 1981)
(Approved by AICTE & Affiliated to OU)
(Accredited by NBA & NAAC)

June,2020

DECLARATION

We hereby declare that the contents presented in the project report titled **"VIDEO STABILIZATION USING POINT FEATURE MATCHING AND MATLAB"** submitted in partial fulfillment for the award of Degree of Bachelor of Engineering in *ELECTRONICS AND COMMUNICATION ENGINEERING (ECE)* to the Department of Electronics and Communication Engineering, *MATURI VENKATA SUBBA RAO (MVSR) ENGINEERING COLLEGE* affiliated to *OSMANIA UNIVERSITY*, *Hyderabad* is a record of the original work carried out by me under the supervision of **Mrs. K. KAVYA**. Further this is to state that the results embodied in this project report have not been submitted to any University or institution for the award of any Degree or Diploma.

S.Yashaswi Ranga	P.Suman	R.Amartya Varma
(2451-16-735-126)	(2451-16-735-138)	(2451-16-735-140)

CERTIFICATE

This is to certify that the thesis titled **"VIDEO STABILIZATION USING POINT FEATURE MATCHING AND MATLAB"** that is being submitted by **S.Yashaswi Ranga (2451-16-735-126), P.Suman (2451-16-735-138), R.Amartya Varma (2451-16-735-140)** in partial fulfillment for the award of Bachelor of Engineering (BE) in Electronics and Communication Engineering to the Department of Electronics and Communication Engineering, *MATURI VENKATA SUBBA RAO (MVSR) ENGINEERING COLLEGE* affiliated to *OSMANIA UNIVERSITY*, *Hyderabad,* is a record of the bonafide project work carried out by him/her under my guidance and supervision.

The results embodied in this thesis have not been submitted to any other University or Institution for the award of any Degree or Diploma.

Internal Guide

Mrs. K. Kavya

Assistant Professor

Head of the Department

Dr. S.P. Venu Madhava Rao

External Examiner

ACKNOWLEDGEMENTS

We express my deep sense of gratitude to my guide Mrs. K. Kavya, Assistant Professor for her constant guidance, untiring help and sparing their valuable inputs throughout my project work. The discussions I had with them enriched my understanding of the project and helped in achieving the goal

We also express my deep sense of gratitude to our Head of the Department, Dr. S.P. Venu Madhava Rao, Professor and our Principal Dr. G. Kanaka Durga for their constant support, encouragement and providing necessary facilities to carry out the project.

We also express our sense of gratitude to all the faculty and staff of the Department for their direct and indirect help in making the project a success.

We express my deep sense of gratitude to my parents for their constant support, encouragement and blessings.

S.Yashaswi Ranga (2451-16-735-126)

P.Suman (2451-16-735-138)

R.Amartya Ramineni (2451-16-735-140)

CONTENTS

ABSTRACT

Video capturing by non-professionals will lead to unanticipated effects. Such as image distortion, image blurring etc. Hence, many researchers study such drawbacks to enhance the quality of videos. In this paper an algorithm is proposed to stabilize jittery videos. A stable output video will be attained without the effect of jitter which is caused due to shaking of handheld camera during video recording. Firstly, salient points from each frame from the input video is identified and processed followed by optimizing and stabilize the video. Optimization includes the quality of the video stabilization. This method has shown good result in terms of stabilization and it discarded distortion from the output videos recorded in different circumstances.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BBC	British Broadcasting Company
NTSC	National Television Sub-Committee
SECAM	Sequential Colour And Memory
PAL	Phase Alteration Line
PIP	Picture In Picture
AVI	Audio Video Interleave
MPEG	Moving Pictures Experts Groups
ISO	International Organization For Standardization
DCT	Discrete Cosine Transform
FAST	Features from Accelerated Segment Test
FREAK	Fast Retina Key Point
RANSAC	Random Sample Consensus
GUI	Graphical User Interface

CHAPTER-1
INTRODUCTION

Nowadays nearly 2 billion people own smartphones worldwide, and an increasing number of videos are captured by mobile devices. However, videos captured by hand handled devices are always shaky and undirected due to the lack of stabilization equipment on the handhold devices. Even though there are commercial hardware components that could stabilize the device when we record, they are relatively redundant and not handy for daily use. Moreover, most hardware stabilization systems only remove high frequencies jitters but are unable to remove low frequency motions arise from panning shots or walking movements. Such slow motion is problematic in shots that intend to track prominent foreground object or person. To overcome the above difficulties, we implement a post-processing video stabilization pipeline aiming to remove undesirable high and low frequency motions from casually captured videos. Similar to most post-processing video stabilization algorithms, our implementation involves three main steps:

(1) estimate original shaky camera path from feature tracking in the video.

(2) calculate a smoothed path, which is cast as a constraint optimization problem.

(3) Synthesizing the stabilized video using the calculated smooth camera path. To reduce high frequency noise, we use the L1 path optimization method described in [1] to produce purely constant, linear or parabolic segments of smoothed motion, which follows cinematographic rules. To reduce low frequency swanning in videos containing a person as the central object, we apply further restraint to the motion of the facial features. In order to make the solution approachable, our method uses automatic feature detection and do not require user interaction.

Our video stabilization method is a purely software approach, and can be applied to videos from any camera devices and sources. Another popular class of mobile video stabilization method use the phone's build-in gyroscope to measure the camera path. Our method has the advantage of being applicable to any video from any sources, for example online video, without any prior knowledge of the capturing device or other physical parameters of the scene. Our approach also enables facial retargeting, which can be extent to other kinds of salient features.

Recently, the market of handheld camera has grown rapidly. However, video capturing by nonprofessional user normally will lead to unanticipated effects, such as image distortion, image blurring etc. Hence, many researchers focused on such drawbacks to enhance the quality of casual videos. [1] Digital video stabilization is the process of removing unwanted movement from a video stream. Generally, the processes of stabilization go through three phases namely

1) Motion estimation
2) Motion smoothing
3) Motion compensation.

The purpose of first phase is to estimate the motion between frames. After that the parameters of estimated motion which are obtained from the first phase will be sent to motion smoothening, where it removes the high-frequency distortion and calculates the global transformation, which is very important to stabilize the current frame [2]. Next, warping will be done by image composition for the motion compensation. These three-step frameworks are the essential steps in most of the video stabilization algorithms.

1.1. VIDEO IMAGE PROCESSING:

A digital video is a moving picture, or movie. Digital video processing is the study of algorithms for processing moving videos that are represented in digital format. Here, we distinguish digital video from digital still videos, which are videos that do not change with time basically, digital photographs.

Digital videos are multidimensional signals, meaning that they are functions of more than a single variable. Indeed, digital video is ordinarily a function of three dimensions – two in space and one in time, as depicted in Fig 1.1. Because of this, digital video processing is data intensive: significant bandwidth, computational, and storage resources are required to handle video streams in digital format.

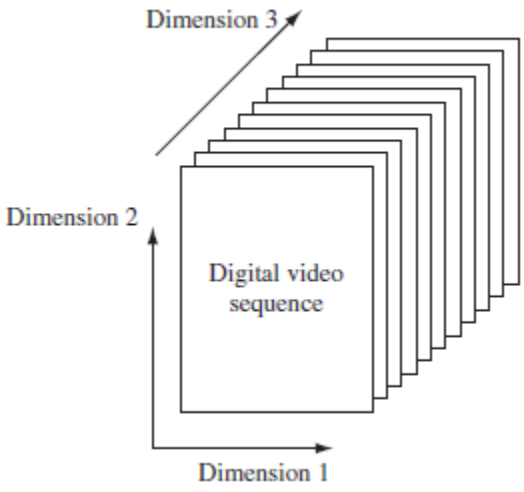


Fig 1.1. Digital Video Sequence

Digital processing of video requires that the video stream be in a digital format, meaning that it must be sampled and quantized.

1.2. HISTORY:

The history of video processing starts when image scanning, the fundamental principle enabling television, was first described in 1840's by Bakewell. It lasted almost till the end of the 19th century before Nipkov, in 1883, converted the principle into the first practical system using mechanical scanning. Around 1926, Baird in London and Jenkins in Washington independently gave the first demonstrations of actual television, using Nipkov's invention. By 1935, the electronic scanning system, invented by Braun in 1897, has become mature for the receiving end of the television system and devices implementing transmission standards have been developed. The British Broadcasting Company (BBC) started the first regular black and white TV broadcast in 1936, but it was only afterworld War II that television broadcasting became popular.

In 1950, a colour television transmission system was developed in the United States, which lead to the NTSC-standard (National Television Sub-Committee) and US colour TV broadcasting in 1953. The European alternatives, SECAM (Sequential Colour and Memory) developed in France and PAL (Phase Alternation Line) developed in Germany, overcome the sensitivity for phase errors in the transmission path of the NTSC-system. Broadcast based on SECAM and PAL has been in use since 1967. The early black and white TV channels occupied approximately 6 to 8 MHz transmission bandwidth. To introduce the colour signal in a compatible fashion, a sub-carrier was added within the least visible part of the video spectrum, modulated by the bandwidth-reduced chrominance signal. By 1970 regular colour TV broadcasts had started in most European countries. After over half a century of using analog television systems, a revolution triggered by digital technology is changing the area of video processing profoundly.

The digital technology generates many new applications, particularly because it significantly simplifies storage and delay of signals, but also creates new problems. Also, the introduction of personal computers, game consoles and digital recording provide new platforms for digital video and generate new challenges for video processing research. More recently, revolutionary developments in display technology, where the traditional scanning electron beam hitting an electro-Luminant screen, is replaced by a flat array (matrix) of cells that individually generate or modulate light, provide large area, high resolution, bright displays with a perfect geometry unequalled by the earlier display technology. The performance of these displays sets new goals for video processing. In the following, we shall first review some technological trends and application domains and reveal the most interesting challenges in modern video systems. Given the scope of the thesis, we shall focus on challenges in the area of image enhancement.

Trends in video:

Digital signal processing did not instantaneously change the video chain, but rather faded into the television receiver with islands of digital signal processing for separate functions to improve the picture quality, such as noise reduction, scan rate conversion, etc. or extending the functionality such as Picture-in- Picture (PIP) and Internet browsing. To date, the process has not ended yet, as this would imply that digital processing should cover every aspect including capture, transmission, storage, etc. Clearly, the impact of digital television is more significant than simply moving from an analog to a digital transmission system. It permits a number of program elements, including video, audio and data that match the services selected by the consumer. The developments in modern display technology, semiconductor manufacturing and digital communication networks, stimulate the evolution of the digital television system and make it feasible.

The evolution from analog to digital:

With the existing analog television broadcasting system, including NTSC, PAL and SECAM, the video audio and some limited data information are conveyed by modulating a Radio Frequency (RF) carrier. Digital transmission has advantages over the analog system in many aspects, like stability, ease of integration, flexibility and high quality, while digital video signals can be relayed over a long distance at extremely low bit error rate. However, there are drawbacks as well.

1.3. Challenges in digital video processing:

Considering the trends discussed in the previous section, various challenges in the area of digital video processing are apparent. The main challenges that we shall discuss in this section include digital video compression, video format conversion, interlaced-to-progressive conversion, and resolution up-conversion and coding-block artefact reduction.

1.4. Different Approaches to Video Stabilization:

Video Stabilization approaches include mechanical, optical and digital stabilization methods. These are discussed briefly below:

- **Mechanical Video Stabilization:** Mechanical image stabilization systems use the motion detected by special sensors like gyros and accelerometers to move the image sensor to compensate for the motion of the camera.
- **Optical Video Stabilization:** In this method, instead of moving the entire camera, stabilization is achieved by moving parts of the

lens. This method employs a moveable lens assembly that variably adjusts the path length of light as it travels through the camera's lens system.

- **Digital Video Stabilization:** This method does not require special sensors for estimating camera motion. There are three main steps — 1) motion estimation 2) motion smoothing, and 3) image composition. The transformation parameters between two consecutive frames are derived in the first stage. The second stage filters out unwanted motion and in the last stage the stabilized video is reconstructed.

1.5. Applications:

- The need for video stabilization spans many domains.

- It is extremely important in consumer and professional **videography**. Therefore, many different mechanical, optical, and algorithmic solutions exist. Even in still image **photography**, stabilization can help take handheld pictures with long exposure times.

- In **medical diagnostic** applications like endoscopy and colonoscopy, videos need to be stabilized to determine the exact location and width of the problem.

- Similarly, in **military applications**, videos captured by aerial vehicles on a reconnaissance flight need to be stabilized for localization, navigation, target tracking, etc. The same applies to **robotic applications**.

CHAPTER 2
DIGITAL VIDEO TECHNOLOGY

Video is the technology of electronically capturing, recording, processing, storing, transmitting, and reconstructing a sequence of still videos representing scenes in motion. Digital video, and its associated technology, presents some significant advantages over analogical solutions mainly due to its potential and capability to become just another type of data that can be manipulated, stored and transmitted along with other types of digital data. Consequently, digital video is being adopted in a proliferating array of new applications, like mobile phones, DVD and digital TV, aimed to full the new and more demanding consumer needs. However, despite all the advantages that digital video technology offers, it has also some drawbacks, such as the amount of information that must be transmitted for good quality video communications.

From the most simplistic point of view, a digital video signal can be a sequence of bi-dimensional still videos which are taken at fixed time intervals. Each of these videos is composed by a set of points (or pixels) which represent the visual luminescence of the captured scene at a specific image point. Therefore, a monochromatic digital image can be seen as a 2-D matrix filled with pixel values, which are usually, quantized using an 8-bit representation. On the other hand, since polychromatic videos are made of three different videos, each of them regarding to one of the three primary colors, red (R), green (G) and blue (B), such videos require 24-bit per pixel to represent the scene luminescence. Hence, considering a video sequence at 30 fps with a spatial resolution of 720 X 480 pixels, which are common values for digital television, this implies a signal bit rate of 248:83 Mbits/s, which is too high for typical communication channels. Consequently, digital

video signals need to be compressed so that they can be used in practical applications.

2.1. Digital Video Processing:

- Digital: The data in desecrate format.
- Video: The sequence of still videos
- Processing: Computing operations (e.g., compression, filtering, retrieval)

Digital VIDEO processing is an area characterized by the need for extensive experimental work to establish the viability of proposed solutions to a given problem. An important characteristic underlying the design of VIDEO processing systems is the significant level of testing & experimentation that normally is required before arriving at an acceptable solution. This characteristic implies that the ability to formulate approaches &quickly prototype candidate solutions generally plays a major role in reducing the cost & time required to arrive at a viable system implementation. There are no clear-cut boundaries in the continuum from VIDEO processing at one end to complete vision at the other. However, consider three types of computerized processes low- , mid-, & high-level processes. Low-level process involves primitive operations such as VIDEO processing to reduce noise, contrast enhancement & VIDEO sharpening. A low- level process is characterized by the fact that both its inputs & outputs are VIDEOs.

Mid-level process on VIDEOs involves tasks such as segmentation, description of that object to reduce them to a form suitable for computer processing & classification of individual objects. A mid-level process is characterized by the fact that its inputs generally

are VIDEOs, but its outputs are attributes extracted from those VIDEOs.

Finally, higher- level processing involves “Making sense” of an ensemble of recognized objects, as in VIDEO analysis & at the far end of the continuum performing the cognitive functions normally associated with human vision.

2.2. Video Basics:

A video (file types like AVI and MPEG) is a collection of several videos. Each image is called frames and the number of videos shown per second is called frames per second or simply FPS. The more frames per second your video has, the better, since more realistic the image will be. Videos are usually saved using at least TV quality settings, i.e. 30 frames per second.

Video is transmitted as a series of pictures, or frames, at a rate typically between 25-60 frames per second. For digital video, the display size is expressed as the number of horizontal and vertical pixels that comprise the screen. The higher the resolution and frame rate, the greater the bitrate required for transmission of the video.

2.3. Video and image Formats:

The format is used to specifies how the image /movie stored in a memory.

2.3.1. File format:

The way a computer or digital A/V device stores information any kind of information is through a series of 0's and 1's. Each 0 or 1 is known as a bit; a string of 8 bits makes up a unit of memory known as a byte. A file format is simply a specified way of arranging a given type of information into bytes.

2.3.2. What's a Video Format?

Video formats are confusing because most video files have at least two different types: the container and the codec used inside that container. The container describes the structure of the file: where the various pieces are stored, how they are interleaved, and which codec's are used by which pieces. It may specify an audio codec as well as video.

A codec ("coder/decoder") is a way of encoding audio or video into a stream of bytes.

2.4. Video Formats:

AVI:

Audio Video Interleaves, this is basic, and system supported format, it has maximum information to form a video sequence.

Video on PC's is cool - you can download video files off the Web, and you capture clips on your home PC to share with friends. There's a standard audio / video file format under Windows called AVI (Audio Video Interleave).

The sad story is that an "AVI" file is just a wrapper, a package that contains some audio / visual stuff, but with no guarantees about what's inside. Microsoft created the AVI format for packaging A/V data, but it's just a specification for sticking A/V data in a file, along with some control information about what's inside.

It's sort of like having rules for putting boxes together ("insert flap A in slot B"), and rules for labelling the box, but anybody can put any sort of stuff in a box, in any language. You can make sense of the contents of a box only if you have the right translation kit, otherwise it's all Greek to you (and to Windows).

Each developer of a new A/V format is responsible for writing the translation kits that permit your Windows system to understand that flavour of AVI. These kits are called "codecs", for compressor-decompressor, because the video and audio formats usually perform some form of compression to reduce the amount of data in the file. Windows comes with some basic codecs built-in (and with additional ones in more recent versions). If you buy video capture hardware like a USB camera or a PCI board, it will include the codec’s needed to understand the formats produced by the hardware. If you buy a video editing program, it will often include additional codecs to support a wider variety of video formats. However, this means you now have a license to create files that other people can't play. Unless they have the same codec, the file is useless to them.

The result is that sharing video files on PC's can be quite messy. Sometimes the hardware or software product provides a way to share codec’s with others or has posted them on a Web site. Sometimes you can find a third party that provides a compatible codec. And sometimes you can't, and it seems like nobody has considered the possibility that

you might want to share your video with others. There is no central clearinghouse for video formats, no standard way to figure out how to get an AVI file to play on your system.

The same basic idea of a general file format with add-in codec's for different video formats is used for the Apple QuickTime video format. As usual, the situation is not as anarchistic on the Macintosh, because Apple tries to build in a wider range of standard formats. Things get more complicated on the PC because video files can be in both AVI (.AVI) and QuickTime (. QT) format, which means installing additional codecs for new AVI formats, plus the separate Apple QuickTime package with its own codec's.

MPEG (pronounced EHM-peg), the Moving Picture Experts Group, develops standards for digital video and digital audio compression. It operates under the auspices of the International Organization for Standardization (ISO). The MPEG standards are an evolving series, each designed for a different purpose.

To use MPEG video files, you need a personal computer with sufficient processor speed, internal memory, and hard disk space to handle and play the typically large MPEG file (which has a file name suffix of .mpg). You also need an MPEG viewer or client software that plays MPEG files. (Note that .mp3 file suffixes indicate MP3 (MPEG-1 audio layer-3) files, not MPEG-3 standard files.) You can download shareware or commercial MPEG players from a number of sites on the Web.

Digital video is part of digital versatile disc (DVD), a new optical disc technology that is expected to rapidly replace the CD-ROM over the

next few years. The DVD holds 4.7 gigabytes of information on one of its two sides, or enough for a 133-minute movie. With two layers on each of its two sides, it will hold up to 17 gigabytes of video, audio, or other information. (Compare this to the current CD-ROM disc of the same physical size, holding 600 megabytes. The DVD can hold more than 28 times as much information.)

The DVD player will also play regular CD-ROM discs. DVDs can be recorded in any of three formats, variously optimized for: video (for example, continuous movies) audio (for example, long-playing music), or a mixture (for example, interactive multimedia presentations). The DVD drive has a transfer rate somewhat faster than an eight-speed CD-ROM player.DVD uses the MPEG-2 file and compression standard. MPEG-2 videos have four times the resolution of MPEG-1 videos and can be delivered at 60 interlaced fields per second where two fields constitute one image frame.

(MPEG-1 can deliver 30 non-interlaced frames per second.) Audio quality on DVD is comparable to that of current audio compact discs. The MPEG standards are an evolving set of standards for video and audio compression and for multimedia delivery developed by the Moving Picture Experts Group (MPEG). MPEG-1 was designed for coding progressive video at a transmission rate of about 1.5 million bits per second. It was designed specifically for Video-CD and CD-i media. MPEG-1 audio layer-3 (MP3) has also evolved from early MPEG work.MPEG-2 was designed for coding interlaced videos at transmission rates above 4 million bits per second. MPEG-2 is used for digital TV broadcast and DVD. An MPEG-2 player can handle MPEG-1 data as well.

MPEG-1 and -2 define techniques for compressing digital video by factors varying from 25:1 to 50:1. The compression is achieved using five different compression techniques:

1. The use of a frequency-based transform called Discrete Cosine Transform (DCT).
2. Quantization, a technique for losing selective information (sometimes known as lossy compression) that can be acceptably lost from visual information.
3. Huffman coding, a technique of lossless compression that uses code tables based on statistics about the encoded data.
4. Motion compensated predictive coding, in which the differences in what has changed between an image and its preceding image are calculated and only the differences are encoded.
5. Bi-directional prediction, in which some videos are predicted from the pictures immediately preceding and following the image.

The first three techniques are also used in JPEG file compression.

A proposed MPEG-3 standard, intended for High Definition TV (HDTV), was merged with the MPEG-2 standard when it became apparent that the MPEG-2 standard met the HDTV requirements.MPEG-4 is a much more ambitious standard and addresses speech and video synthesis, fractal geometry, computer visualization, and an artificial intelligence (AI) approach to reconstructing videos. MPEG-4 addresses a standard way for authors to create and define the media objects in a multimedia presentation, how these can be synchronized and related to each other in transmission, and how users are to be able to interact with the media

objects. MPEG-21 provides a larger, architectural framework for the creation and delivery of multimedia. It defines seven key elements:

- Digital item declaration
- Digital item identification and declaration
- Content handling and usage
- Intellectual property management and protection
- Terminals and networks
- Content representation
- Event reporting

The details of various parts of the MPEG-21 framework are in various draft stages.

CHAPTER 3
LITERATURE REVIEW

Video stabilization methods can be categorized into three major directions: 2D method, 3D method and motion estimation method. 2D methods estimate frame-to-frame 2D transformations, and smooth the transformations to create a more stable camera path. Early work by Matsushita et al. [5] applied low-path filters to smooth the camera trajectories. Gleicher and Liu [4] proposed to create a smooth camera path by inserting linearly interpolated frames. Liu et al. [6] later incorporated subspace constraints in smoothing camera trajectories, but it required longer feature tracks. 3D methods rely on feature tracking to stabilize shaky videos. Beuhler et al. [8] utilized projective 3D reconstruction to stabilize videos from uncalibrated cameras. Liu et al. [9] were the first to introduce content-preserving warping in video stabilization.

However, 3D reconstruction is difficult and not robust. Liu et al. [6] reduced the problem to smoothing long feature trajectories and achieved comparable results to 3D reconstruction-based methods. Goldstein and Fattal[10] proposed an epipolar transfer method to avoid direct 3D reconstruction. Obtaining long feature tracks is often fragile in consumer videos due to occlusion, rapid camera motion and motion blur. Lee et al. [11] incorporated feature pruning to select more robust feature trajectories to resolve the occlusion issue. Motion estimation methods calculate transitions between consecutive frames with view-overlap. To reduce the alignment error due to parallax, Shum and Szeliski[12] imposed local alignment, and Gao et al. [7] introduced a dualhomography model. Liu et al [13] proposed a mesh-based, spatially variant homography model to represent the motion between video frames, but the smoothing strategy did not follow cinematographic rules.

Our implementation, based on [1], apply L1-norm optimization to generate a camera path that consists of only constant, linear and parabolic segments, which follow cinematographic principles in producing professional videos. Initial step in the video stabilization is selection of feature points. These feature points are corners. Corner detectors were developed in the late 1970's. Labeeb Mohsin Abdullah et.al. [1] had proposed an algorithm to stabilize the image sequence by using Harris corner detection technique. Harris Corner Detection is one of the fastest algorithms to find corner values [3]. Aleksandr Shnayderman et.al. had proposed SVD (Signature Value Decomposition) based grey scale image quality measurement. Edward Rosten et.al proposed faster and better machine learning approach to corner detection. This algorithm is used for identifying Interest points in an image [4]. Yue Wang et.al. had proposed Real-Time Video Stabilization for Unmanned Aerial Vehicles. Elmar Mair et.al.proposed technique of Adaptive and Generic Corner Detection

Based on the Accelerated Segment Test. C. Harris and M.J. Ephens proposed combined corner and edge detection to cater for image regions containing texture and isolated features, which is based on the local auto-correlation function. Mohammed A. Alharbi, had proposed Fast Video Stabilization Algorithm. [5] In this affine motion model is utilized to determine the parameters of translation and rotation between images. The determined affine transformation is then exploited to compensate for the abrupt temporal discontinuities of input image sequences.

3.1. Implementation:

Methodology:

Features from accelerated segment test (FAST) is a corner detection method which could be used to extract feature points and

later used to track and map objects in many computer visions tasks. FAST is an algorithm proposed originally by Rosten and Drummond for identifying Interest points in an image. It is fast than many other well-known feature extraction methods, such as Difference of Gaussians (DoG) used by SIFT, SUSAN and Harris. An interest point in an image is a pixel which has a well-defined position and can be robustly detected [7]. Interest points have high local information content and they should be ideally repeatable between different images.

3.2. Flowchart:

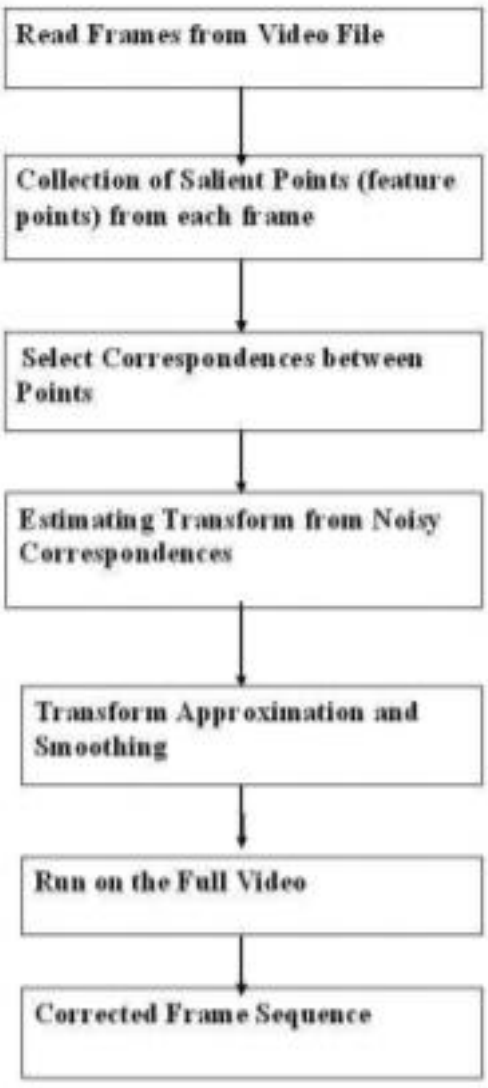


Fig 3.2. Flow chart of the video stabilization algorithm

3.3. Algorithm & Implementation:

The Flow chart of video stabilization algorithm is shown in Fig 3.2. The total process of video capturing and stabilization has been performed in several steps. These steps are elaborated with simulation results.

3.3.1. Reading Frames:

Read Frames from Video File In this step, first two frames of a video sequence are read. We read them as intensity images since colour is not necessary for the stabilization algorithm and also using grayscale images improve the speed. Fig 3.3.1. shows both frames side by side. The data related intensity of images has been separated from colour

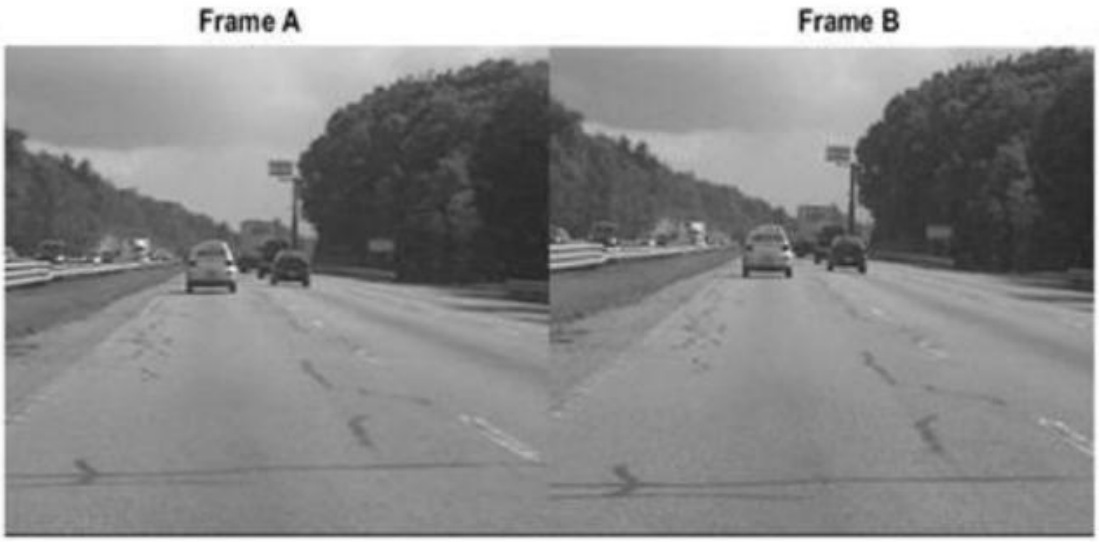


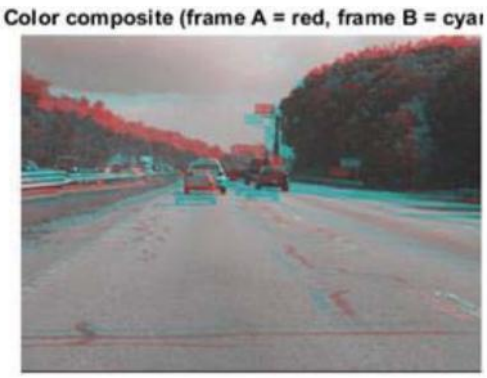
Fig 3.3.1. Reading the two frames

3.3.2. Collection of Salient Points (feature points) from each frame:

After reading two frames, Red-Cyan color composite is produced to illustrate the pixel wise difference between them as shown in Fig 3.3.2.(a) Our goal is to determine a transformation that will correct for

the distortion between the two frames, for this Geometric Transform function can be used, which return an affine transform [8]. The input for this function shall comprise a set of point correspondences between the two frames. To generate these correspondences, collection of points of interest from both frames is performed [9]. For this purpose, FAST is corner detection algorithm is used. Fig 3.3.2.(b) shows Image showing the interest point under test and the 16 pixels on the circle.

It gives high measured quality video and less computational time. The detected points from both frames are shown in the Fig 3.3.2. (c) & Fig 3.3.2(d). Observe how many of them cover the same image features, such as points along the tree line, the corners of the large road sign, and the corners of the cars.



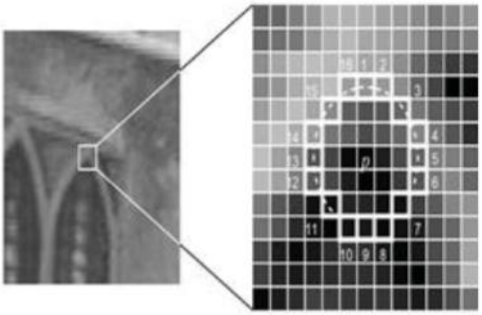


Fig 3.3.2.(a). **Fig3.3.2.(b).**

- **Fig 3.3.2.(a). Red-Cyan color composite**
- **Fig 3.3.2.(b). Image showing the interest point under test and the 16 pixels on circle**

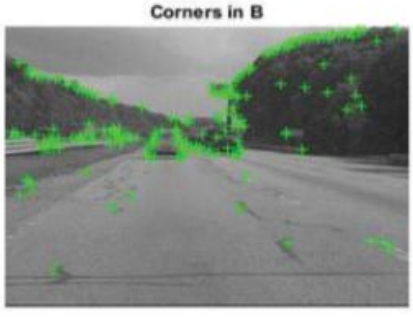


Fig 3.3.2(c). Corners in frame A **Fig 3.3.2.(d). Corners in frame B**

3.3.3. Select Correspondences between Points:

Select Correspondences between Points In this step, correspondences between the points derived in the 3.3.2 should be established. For each point, we extract a Fast Retina Key point (FREAK) descriptor centred around it [10]. The matching cost we use between points is the Hamming distance since FREAK descriptors are binary. Points in frame A and frame B are matched putatively. Note that there is no uniqueness constraint, so points from frame B can correspond to multiple points in frame A. Fig 3.3.4.(a) shows the same colour composite given above, in this point from frame A are shown with red colour, and the points from frame B in green colour. Yellow lines are drawn between points to show the correspondences. Many of these correspondences are correct, but still there is significant number of outliers.

3.3.4. Estimating Transform from Noisy Correspondences:

Estimating Transform from Noisy Correspondences Many of the point correspondences obtained in the previous step are identified with limited accuracy. To rectify this problem, Random Sample Consensus (RANSAC) algorithm is used which is implemented in the Geometric Transform function. It is observed that the inliers correspondences in the image background are not aligned with foreground. The reason behind this is the background features are far enough those act as if they were on an infinitely distant plane. We can assume that background plane is static and will not change dramatically between the first and second frame, instead, this transform is capturing the motion of the camera. Thus, correcting process will stabilize the video.

Furthermore, as long as the motion of the camera between frame A and frame B is minimize or the time of sampling the video is high enough, this condition is maintained. The RANSAC algorithm is repeated multiple times and at each run the cost of the result is

calculated by projecting frame B onto frame A via Sum of Absolute Differences between the two image frames. Fig 3.3.4.(b) shows a colour composite showing frame A overlaid with the re-projected frame B, along with the re-projected point correspondences. It is clear from this figure that the results are favourable, with the inliers correspondences nearly exactly coincident. The cores of the images are both well aligned, such that the red-cyan colour composite becomes almost purely black- and white in that region.

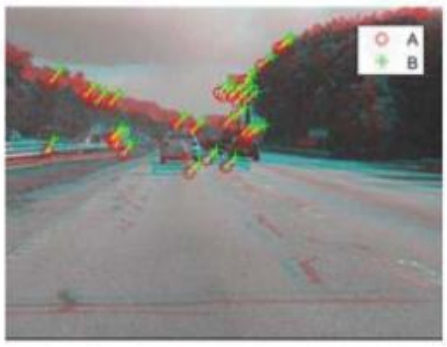


Fig 3.3.4.(a). Correspondence between Points

Fig 3.3.4.(b). Color composites

3.3.5. Transform Approximation and Smoothing:

Transform Approximation and Smoothing We could use all the six parameters of the affine transform, but, for numerical simplicity and stability, we choose to re-fit the matrix as a simple scale-rotation-translation transform. This has only four free parameters as compared to the full affine transform which are one scale factor, one angle, and two translations. Finally, reconstructed S-R-t transform output is shown in Fig 3.3.6.(a).

3.3.6. Run on the Full Video:

Run on the Full Video The above procedure of estimating the transform between two images has been performed in the MATLAB® function cvexEstStabilizationTform. The function cvexTformToSRT also converts a general affine transform into a scale-rotation-translation

transform. The smoothed video frame is shown in Fig 3.3.6.(b) by using Video Player in MATLAB.

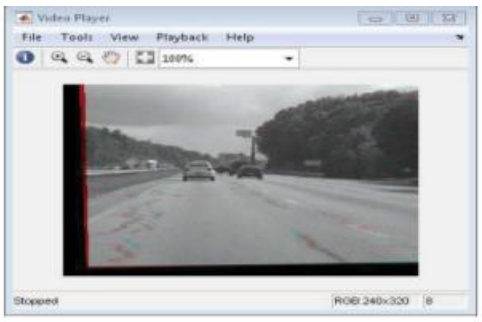


Fig 3.3.6.(a). **Fig 3.3.6.(b).**

3.3.6.(a). Color composites of affine and s-R-t Transform output

3.3.6.(b). Camera movement to stabilize video

3.3.7. Corrected Frame Sequence:

Corrected Frame Sequence: During computation, we computed the mean of the raw video frames and mean of the corrected frames. These mean values are shown side-by-side. The left image shows the mean of the raw input frames, proving that there was a great deal of distortion in the original video. The mean of the corrected frames on the right, however, shows the image with almost no distortion.

CHAPTER 4
SOFTWARE INTRODUCTION

What Is MATLAB?

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modelling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB uses software developed by the LAPACK and ARPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the

tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The MATLAB System:

The MATLAB system consists of five main parts:

Development Environment:

This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, and browsers for viewing help, the workspace, files, and the search path.

The MATLAB Mathematical Function Library:

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

The MATLAB Language:

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

Handle Graphics®:

This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

The MATLAB Application Program Interface (API) :

This is a library that allows you to write C and FORTRAN programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

4.1. Development Environment:

Introduction:

This chapter provides a brief introduction to starting and quitting MATLAB, and the tools and functions that help you to work with MATLAB variables and files. For more information about the

topics covered here, see the corresponding topics under Development Environment in the MATLAB documentation, which is available online as well as in print.

Starting and Quitting MATLAB:

Starting MATLAB:

On a Microsoft Windows platform, to start MATLAB, double-click the MATLAB shortcut icon on your Windows desktop.

On a UNIX platform, to start MATLAB, type MATLAB at the operating system prompt. After starting MATLAB, the MATLAB desktop opens - see MATLAB Desktop.

You can change the directory in which MATLAB starts, define start-up options including running a script upon start-up, and reduce start-up time in some situations.

Quitting MATLAB:

To end your MATLAB session, select Exit MATLAB from the File menu in the desktop, or type quit in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a finish.m script.

4.2. MATLAB Desktop:

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The first time MATLAB starts, the desktop appears as shown in the following illustration, although your Launch Pad may contain different entries.

You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. You can also move tools outside of the desktop or return them back inside the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts.

You can specify certain characteristics for the desktop tools by selecting Preferences from the File menu. For example, you can specify the font characteristics for Command Window text. For more information, click the Help button in the Preferences dialog box.

4.2.1. Desktop Tools:

This section introduces MATLAB's desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are:

- Current Directory Browser
- Workspace Browser
- Array Editor
- Editor/Debugger
- Command Window
- Command History
- Launch Pad
- Help Browser

Command Window

Use the Command Window to enter variables and run functions and M-files.

Command History

Lines you enter in the Command Window are logged in the Command History window. In the Command History, you can view previously used functions, and copy and execute selected lines. To save the input and output from a MATLAB session to a file, use the diary function.

Running External Programs

You can run external programs from the MATLAB Command Window. The exclamation point character! is a shell escape and indicates that the rest of the input line is a command to the operating system. This is useful for invoking utilities or running other programs without quitting MATLAB. On Linux, for example,!emacs magik.m invokes an editor called emacs for a file named magik.m. When you quit the external program, the operating system returns control to MATLAB.

Launch Pad

MATLAB's Launch Pad provides easy access to tools, demos, and documentation.

Help Browser

Use the Help browser to search and view documentation for all your Math Works products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents.

To open the Help browser, click the help button in the toolbar, or type helpbrowser in the Command Window. The Help browser consists of two panes, the Help Navigator, which you use to find information, and the display pane, where you view the information

Help Navigator

Use to Help Navigator to find information. It includes:

Product filter - Set the filter to show documentation only for the products you specify.

Contents tab - View the titles and tables of contents of documentation for your products.

Index tab - Find specific index entries (selected keywords) in the MathWorks documentation for your products.

Search tab - Look for a specific phrase in the documentation. To get help for a specific function, set the Search type to Function Name.

Favourites tab - View a list of documents you previously designated as favourites.

Display Pane

After finding documentation using the Help Navigator, view it in the display pane. While viewing the documentation, you can:

Browse to other pages - Use the arrows at the tops and bottoms of the pages or use the back and forward buttons in the toolbar.

Bookmark pages - Click the Add to favourites button in the toolbar.

Print pages - Click the print button in the toolbar.

Find a term in the page - Type a term in the Find in page field in the toolbar and click Go.

Other features available in the display pane are: copying information, evaluating a selection, and viewing Web pages.

Current Directory Browser:

MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path.

Search Path:

To determine how to execute functions you call, MATLAB uses a search path to find M-files and other MATLAB-related files, which are organized in directories on your file system. Any file you want to run in MATLAB must reside in the current directory or in a directory that is on the search path. By default, the files supplied with MATLAB and MathWorks toolboxes are included in the search path.

Workspace Browser

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces.

To view the workspace and information about each variable, use the Workspace browser, or use the functions who and whos.

To delete variables from the workspace, select the variable and select Delete from the Edit menu. Alternatively, use the clear function.

The workspace is not maintained after you end the MATLAB session. To save the workspace to a file that can be read during a later MATLAB session, select Save Workspace as from the File menu, or use the save function. This saves the workspace to a binary file called a MAT-file, which has a.mat extension. There are options for saving to different formats. To read in a MAT-file, select Import Data from the File menu, or use the load function.

Array Editor

Double-click on a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.

Editor/Debugger

Use the Editor/Debugger to create and debug M-files, which are programs you write to runMATLAB functions. The Editor/Debugger provides a graphical user interface for basic text editing, as well as for M-file debugging.

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop File menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint.

If you just need to view the contents of an M-file, you can display it in the Command Window by using the type function.

4.3. Manipulating Matrices:

Entering Matrices

The best way for you to get started with MATLAB is to learn how to handle matrices. Start MATLAB and follow along with each example.

You can enter matrices into MATLAB in several different ways:

- Enter an explicit list of elements.

- Load matrices from external data files.
- Generate matrices using built-in functions.
- Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. You have only to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [].

To enter Dürer's matrix, simply type in the Command Window

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered.

```
A =

    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

This exactly matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A.

4.3.1 Expressions:

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages,

these expressions involve entire matrices. The building blocks of expressions are:

- Variables
- Numbers
- Operators
- Functions

4.3.2 Variables:

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

num_students = 25

Creates a 1-by-1 matrix named num_students and stores the value 25 in its single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are not the same variable. To view the matrix assigned to any variable, simply enter the variable name.

4.3.3. Numbers:

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are

3 -99 0.0001

9.6397238 1.60210e-20 6.02252e23

1i -3.14159j 3e5i

All numbers are stored internally using the long format specified by the IEEE floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly 10-308 to 10+308.

4.3.4 Operators:

Expressions use familiar arithmetic operators and precedence rules.

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division (described in "Matrices and Linear Algebra" in Using MATLAB)
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

Table 4.3.4. Various Operators

4.3.5. Functions:

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

help elfun : For a list of more advanced mathematical and matrix functions, type

help specfun

help elmat

Some of the functions, like sqrt and sin, are built-in. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like gamma and sinh, are implemented in M-files. You can see the code and even modify it if you want. Several special functions provide values of useful constants.

Pi	3.14159265...
i	Imaginary unit, $\sqrt{-1}$
I	Same as i
Eps	Floating-point relative precision, 2^{-52}
Realmin	Smallest floating-point number, 2^{-1022}
Realmax	Largest floating-point number, $(2-\varepsilon)2^{1023}$
Inf	Infinity
NaN	Not-a-number

Table 4.3.5. Functions

CHAPTER 5
GUI

A graphical user interface (GUI) is a user interface built with graphical objects, such as buttons, text fields, sliders, and menus. In general, these objects already have meanings to most computer users. For example, when you move a slider, a value changes; when you press an OK button, your settings are applied and the dialog box is dismissed. Of course, to leverage this built-in familiarity, you must be consistent in how you use the various GUI-building components.

Applications that provide GUIs are generally easier to learn and use since the person using the application does not need to know what commands are available or how they work. The action that results from a particular user action can be made clear by the design of the interface.

The sections that follow describe how to create GUIs with MATLAB. This includes laying out the components, programming them to do specific things in response to user actions, and saving and launching the GUI; in other words, the mechanics of creating GUIs. This documentation does not attempt to cover the "art" of good user interface design, which is an entire field unto itself. Topics covered in this section include:

Creating GUIs with GUIDE

MATLAB implements GUIs as figure windows containing various styles of uicontrol objects. You must program each object to perform the intended action when activated by the user of the GUI. In addition, you must be able to save and launch your GUI. All of these tasks are

simplified by GUIDE, MATLAB's graphical user interface development environment.

5.1. GUI Development Environment:

The process of implementing a GUI involves two basic tasks:

- Laying out the GUI components
- Programming the GUI components

GUIDE primarily is a set of layout tools. However, GUIDE also generates an M-file that contains code to handle the initialization and launching of the GUI. This M-file provides a framework for the implementation of the call-backs - the functions that execute when users activate components in the GUI.

The Implementation of a GUI:

While it is possible to write an M-file that contains all the commands to lay out a GUI, it is easier to use GUIDE to lay out the components interactively and to generate two files that save and launch the GUI:

A FIG-file - contains a complete description of the GUI figure and all of its

children (uncontrol's and axes), as well as the values of all object properties.

An M-file - contains the functions that launch and control the GUI and the

call-backs, which are defined as subfunctions. This M-file is referred to as the application M-file in this documentation.

Note that the application M-file does not contain the code that lays out the uncontrol's; this information is saved in the FIG-file.

The following diagram illustrates the parts of a GUI implementation.

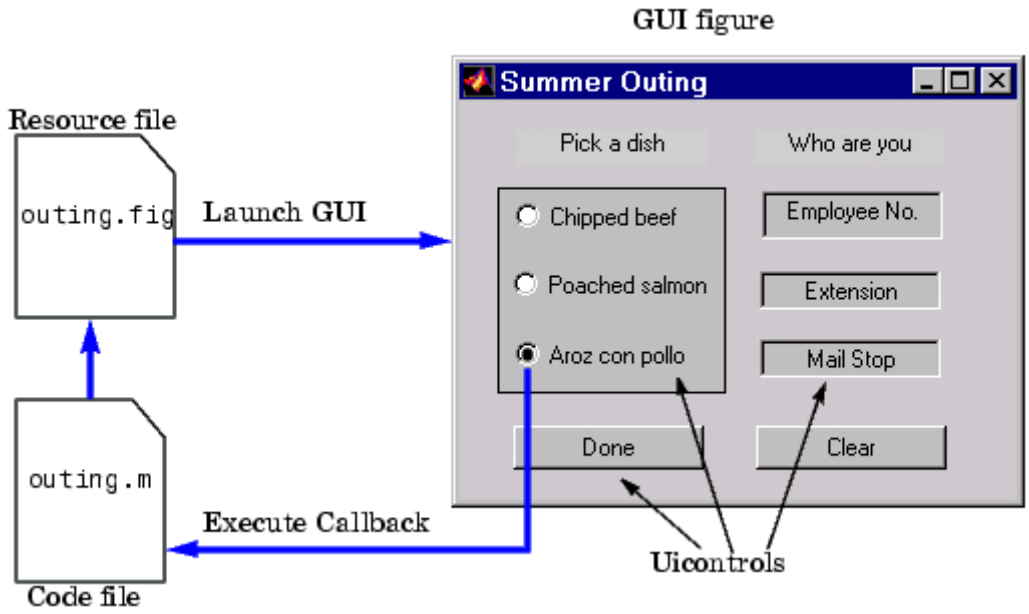


Fig 5.1 Parts of GUI Implementation

5.2. Features of the GUIDE-Generated Application

M-File:

GUIDE simplifies the creation of GUI applications by automatically generating an M-file framework directly from your layout. You can then use this framework to code your application M-file. This approach provides a number of advantages:

The M-file contains code to implement a number of useful features (see Configuring Application Options for information on these features). The M-file adopts an effective approach to managing object handles and executing callback routines (see Creating and Storing the

Object Handle Structure for more information). The M-files provides a way to manage global data (see Managing GUI Data for more information).

The automatically inserted subfunction prototypes for callbacks ensure compatibility with future releases. For more information, see Generating Callback Function Prototypes for information on syntax and arguments.

You can elect to have GUIDE generate only the FIG-file and write the application M-file yourself. Keep in mind that there are no uicontrol creation commands in the application M-file; the layout information is contained in the FIG-file generated by the Layout Editor.

5.2.1. Beginning the Implementation Process:

To begin implementing your GUI, proceed to the following sections:

Getting Started with GUIDE - the basics of using GUIDE.

Selecting GUIDE Application Options - set both FIG-file and M-file options.

Using the Layout Editor - begin laying out the GUI.

Understanding the Application M-File - discussion of programming techniques used in the application M-file.

Application Examples - a collection of examples that illustrate techniques which are useful for implementing GUIs.

5.2.2. Command-Line Accessibility:

When MATLAB creates a graph, the figure and axes are included in the list of children of their respective parents and their

handles are available through commands such as findobj, set, and get. If you issue another plotting command, the output is directed to the current figure and axes.

GUIs are also created in figure windows. Generally, you do not want GUI figures to be available as targets for graphics output, since issuing a plotting command could direct the output to the GUI figure, resulting in the graph appearing in the middle of the GUI.

In contrast, if you create a GUI that contains an axes and you want commands entered in the command window to display in this axes, you should enable command-line access.

5.3. User Interface Controls:

The Layout Editor component palette contains the user interface controls that you can use in your GUI. These components are MATLAB uicontrol objects and are programmable via their Callback properties. This section provides information on these components.

- Push Buttons
- Sliders
- Toggle Buttons
- Frames
- Radio Buttons
- Listboxes
- Checkboxes
- Popup Menus
- Edit Text
- Axes
- Static Text
- Figures

5.3.1. Push Buttons:

Push buttons generate an action when pressed (e.g., an OK button may close a dialog box and apply settings). When you click down on a push button, it appears depressed; when you release the mouse, the button's appearance returns to its nondepressed state; and its callback executes on the button up event.

Properties to Set

String - set this property to the character string you want displayed on the push button.

Tag - GUIDE uses the Tag property to name the callback subfunction in the application M-file. Set Tag to a descriptive name (e.g., close_button) before activating the GUI.

Programming the Callback:

When the user clicks on the push button, its callback executes. Push buttons do not return a value or maintain a state.

5.3.2. Toggle Buttons:

Toggle buttons generate an action and indicate a binary state (e.g., on or off). When you click on a toggle button, it appears depressed and remains depressed when you release the mouse button, at which point the callback executes. A subsequent mouse click returns the toggle button to the nondepressed state and again executes its callback.

Programming the Callback

The callback routine needs to query the toggle button to determine what state it is in. MATLAB sets the Value property equal to the Max property when the toggle button is depressed (Max is 1 by

default) and equal to the Min property when the toggle button is not depressed (Min is 0 by default).

From the GUIDE Application M-File

The following code illustrates how to program the callback in the GUIDE application M-file.

```
function varargout =
togglebutton1_Callback(h,eventdata,handles,varargin)

button_state = get(h,'Value');

if button_state == get(h,'Max')

   % toggle button is pressed

elseif button_state == get(h,'Min')

   % toggle button is not pressed

end
```

Adding an Image to a Push Button or Toggle Button

Assign the CData property an m-by-n-by-3 array of RGB values that define a truecolor image. For example, the array a defines 16-by-128 truecolor image using random values between 0 and 1 (generated by rand).

```
a(:,:,1) = rand(16,128);

a(:,:,2) = rand(16,128);

a(:,:,3) = rand(16,128);

set(h,'CData',a)
```

5.3.3. Radio Buttons:

Radio buttons are similar to checkboxes, but are intended to be mutually exclusive within a group of related radio buttons (i.e., only one button is in a selected state at any given time). To activate a radio button, click the mouse button on the object. The display indicates the state of the button.

Implementing Mutually Exclusive Behaviour

Radio buttons have two states - selected and not selected. You can query and set the state of a radio button through its Value property:

Value = Max, button is selected.

Value = Min, button is not selected.

To make radio buttons mutually exclusive within a group, the callback for each radio button must set the Value property to 0 on all other radio buttons in the group. MATLAB sets the Value property to 1 on the radio button clicked by the user.

The following subfunction, when added to the application M-file, can be called by each radio button callback. The argument is an array containing the handles of all other radio buttons in the group that must be deselected.

```
function mutual_exclude(off)

set(off,'Value',0)
```

Obtaining the Radio Button Handles.

The handles of the radio buttons are available from the handles structure, which contains the handles of all components in the GUI. This structure is an input argument to all radio button callbacks.

The following code shows the call to mutual_exclude being made from the first radio button's callback in a group of four radio buttons.

```
function varargout =
radiobutton1_Callback(h,eventdata,handles,varargin)

off =
[handles.radiobutton2,handles.radiobutton3,handles.radiobutton4];

mutual_exclude(off)

% Continue with callback

    .

    .

    .
```

After setting the radio buttons to the appropriate state, the callback can continue with its implementation-specific tasks.

5.3.4. Checkboxes:

Check boxes generate an action when clicked and indicate their state as checked or not checked. Check boxes are useful when providing the user with a number of independent choices that set a mode (e.g., display a toolbar or generate callback function prototypes).

The Value property indicates the state of the check box by taking on the value of the Max or Min property (1 and 0 respectively by default):

Value = Max, box is checked.

Value = Min, box is not checked.

You can determine the current state of a check box from within its callback by querying the state of its Value property, as illustrated in the following example:

```
function checkbox1_Callback(h,eventdata,handles,varargin)
if (get(h,'Value') == get(h,'Max'))
   % then checkbox is checked-take approriate action
else
   % checkbox is not checked-take approriate action
end
```

5.3.5. Edit Text:

Edit text controls are fields that enable users to enter or modify text strings. Use edit text when you want text as input. The String property contains the text entered by the user.

To obtain the string typed by the user, get the String property in the callback.

```
function edittext1_Callback(h,eventdata, handles,varargin)
user_string = get(h,'string');
% proceed with callback...
```

Obtaining Numeric Data from an Edit Test Component

MATLAB returns the value of the edit text String property as a character string. If you want users to enter numeric values, you must convert the characters to numbers. You can do this using the str2double command, which converts strings to doubles. If the user enters non-numeric characters, str2double returns NaN.

You can use the following code in the edit text callback. It gets the value of the String property and converts it to a double. It then checks if the converted value is NaN, indicating the user entered a non-numeric character (isnan) and displays an error dialog (errordlg).

```
function edittext1_Callback(h,eventdata,handles,varargin)
```

```
user_entry = str2double(get(h,'string'));
if isnan(user_entry)
   errordlg('You must enter a numeric value','Bad Input','modal')
end
% proceed with callback...
```

Triggering Callback Execution

On UNIX systems, clicking on the menubar of the figure window causes the edit text callback to execute. However, on Microsoft Windows systems, if an editable text box has focus, clicking on the menubar does not cause the editable text callback routine to execute. This behavior is consistent with the respective platform conventions. Clicking on other components in the GUI execute the callback.

5.3.6.Static Text:

Static text controls displays lines of text. Static text is typically used to label other controls, provide directions to the user, or indicate values associated with a slider. Users cannot change static text interactively and there is no way to invoke the callback routine associated with it.

5.3.7. Frames:

Frames are boxes that enclose regions of a figure window. Frames can make a user interface easier to understand by visually grouping related controls. Frames have no callback routines associated with them and only uicontrols can appear within frames (axes cannot).

Placing Components on Top of Frames

Frames are opaque. If you add a frame after adding components that you want to be positioned within the frame, you need to bring forward those components. Use the Bring to Front and Send to Back operations in the Layout menu for this purpose.

5.3.8. List Boxes:

List boxes display a list of items and enable users to select one or more items.

The String property contains the list of strings displayed in the list box. The first item in the list has an index of 1.

The Value property contains the index into the list of strings that correspond to the selected item. If the user selects multiple items, then Value is a vector of indices.

By default, the first item in the list is highlighted when the list box is first displayed. If you do not want any item highlighted, then set the Value property to empty, [].

The ListboxTop property defines which string in the list displays as the top most item when the list box is not large enough to display all list entries. ListboxTop is an index into the array of strings defined by the String property and must have a value between 1 and the number of strings. Noninteger values are fixed to the next lowest integer.

Single or Multiple Selection

The values of the Min and Max properties determine whether users can make single or multiple selections:

If Max - Min > 1, then list boxes allow multiple item selection.

If Max - Min <= 1, then list boxes do not allow multiple item selection.

Selection Type

Listboxes differentiate between single and double clicks on an item and set the figure SelectionType property to normal or open accordingly. See Triggering Callback Execution for information on how to program multiple selection.

Triggering Callback Execution

MATLAB evaluates the list box's callback after the mouse button is released or a keypress event (including arrow keys) that changes the Value property (i.e., any time the user clicks on an item, but not when clicking on the list box scrollbar). This means the callback is executed after the first click of a double-click on a single item or when the user is making multiple selections.

In these situations, you need to add another component, such as a Done button (push button) and program its callback routine to query the list box Value property (and possibly the figure SelectionType property) instead of creating a callback for the list box. If you are using the automatically generated application M-file option, you need to either:

Set the list box Callback property to the empty string ('') and remove the callback subfunction from the application M-file. Leave the callback subfunction stub in the application M-file so that no code executes when users click on list box items.

The first choice is best if you are sure you will not use the list box callback and you want to minimize the size and efficiency of the application M-file. However, if you think you may want to define a callback for the list box at some time, it is simpler to leave the callback stub in the M-file.

5.3.9. Popup Menus:

Popup menus open to display a list of choices when users press the arrow.

The String property contains the list of string displayed in the popup menu. The Value property contains the index into the list of strings that correspond to the selected item.

When not open, a popup menu displays the current choice, which is determined by the index contained in the Value property. The first item in the list has an index of 1.

Popup menus are useful when you want to provide users with several mutually exclusive choices, but do not want to take up the amount of space that a series of radio buttons requires.

Programming the Popup Menu

You can program the popup menu callback to work by checking only the index of the item selected (contained in the Value property) or you can obtain the actual string contained in the selected item.

This callback checks the index of the selected item and uses a switch statement to act based on the value. If the contents of the popup menu are fixed, then you can use this approach.

```
function varargout =
popupmenu1_Callback(h,eventdata,handles,varargin)

val = get(h,'Value');

switch val

case 1

% The user selected the first item
```

```
case 2
% The user selected the second item
% etc.
```

This callback obtains the actual string selected in the popup menu. It uses the value to index into the list of strings. This approach may be useful if your program dynamically loads the contents of the popup menu based on user action and you need to obtain the selected string. Note that it is necessary to convert the value returned by the String property from a cell array to a string.

```
function varargout =
popupmenu1_Callback(h,eventdata,handles,varargin)
val = get(h,'Value');
string_list = get(h,'String');
selected_string = string_list{val}; % convert from cell array to string
% etc.
```

Enabling or Disabling Controls

You can control whether a control responds to mouse button clicks by setting the Enable property. Controls have three states:

```
on          - The control is operational
off         - The control is disabled and its label (set by the string
property) is
grayed out.
```

inactive - The control is disabled, but its label is not grayed out.

When a control is disabled, clicking on it with the left mouse button does not execute its callback routine. However, the left-click causes two other callback routines to execute:

First the figure WindowButtonDownFcn callback executes. Then the control's ButtonDownFcn callback executes.

A right mouse button click on a disabled control posts a context menu, if one is defined for that control. See the Enable property description for more details.

5.3.10. Axes:

Axes enable your GUI to display graphics (e.g., graphs and images). Like all graphics objects, axes have properties that you can set to control many aspects of its behavior and appearance. See Axes Properties for general information on axes objects.

Axes Callbacks

Axes are not uicontrol objects, but can be programmed to execute a callback when users click a mouse button in the axes. Use the axes ButtonDownFcn property to define the callback.

Plotting to Axes in GUIs

GUIs that contain axes should ensure the Command-line accessibility option in the Application Options dialog is set to Callback (the default). This enables you to issue plotting commands from callbacks without explicitly specifying the target axes.

GUIs with Multiple Axes

If a GUI has multiple axes, you should explicitly specify which axes you want to target when you issue plotting commands. You can

do this using the axes command and the handles structure. For example, axes(handles.axes1)

This makes the axes whose Tag property is axes1 the current axes, and therefore the target for plotting commands. You can switch the current axes whenever you want to target a different axis. See GUI with Multiple Axes for an example that uses two axes.

5.3.11. Figure:

Figures are the windows that contain the GUI you design with the Layout Editor. See the description of figure properties for information on what figure characteristics you can control.

CHAPTER 6
SOURCE CODE

%Calculating MSE:

```
function [ MSE ] = CalculateMSE( CurrentFrame,PreviousFrame)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
InputImage=double(CurrentFrame);
ReconstructedImage=double(PreviousFrame);
n=size(InputImage);
M=n(1);
N=n(2);
MSE = sum(sum((InputImage-ReconstructedImage).^2))/(M*N);
end
```

%Compensate Rows:

```
function out = compensaterow(a);
[r c]=size(a);
out=ones(r,c);
% out1=ones(r,c);
% for i=3:r
%     for j=1:c
%         out(i-2,j)=a(i,j);
%     end
% end
out(1:r,1:c)=159;
for i=1:r
    for j=21:c-1
```

```
        out(i,j-20)=a(i,j);
    end
end
disp('exit');
```

%Compensate Columns:

```
function out = compensatecol(a);
[r c]=size(a);
out(1:r,1:c)=159;
% out1=ones(r,c);
 for i=21:r-1
    for j=1:c
        out(i-20,j)=a(i,j);
    end
end
% for i=1:r
%     for j=21:c-1
%         out(i,j-20)=a(i,j);
%     end
% end
disp('exit');
```

%Guidedemo.m(Main Code):

```
function varargout = guidemo(varargin)
% GUIDEMO MATLAB code for guidemo.fig
%      GUIDEMO, by itself, creates a new GUIDEMO or raises the existing
%      singleton*.
%      H = GUIDEMO returns the handle to a new GUIDEMO or the
handle to
```

```
%      the existing singleton*.
%      GUIDEMO('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in GUIDEMO.M with the given input
arguments.
%      GUIDEMO('Property','Value',...) creates a new GUIDEMO or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before guidemo_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to guidemo_OpeningFcn via varargin.
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help guidemo
% Last Modified by GUIDE v2.5 11-Jun-2017 11:18:44
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guidemo_OpeningFcn, ...
                   'gui_OutputFcn',  @guidemo_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before guidemo is made visible.
function guidemo_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guidemo (see VARARGIN)
% Choose default command line output for guidemo
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes guidemo wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = guidemo_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
% --- Executes on button press in Browse.
function Browse_Callback(hObject, eventdata, handles)
% hObject    handle to Browse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
[filename, pathname] = uigetfile('*.*', 'Pick a video');
    if isequal(filename,0) || isequal(pathname,0)
       disp('User pressed cancel')
    else
     readerobj = VideoReader(filename, 'tag', 'myreader1');
       % Read in all video frames.
       vidFrames = read(readerobj);
       % Get the number of frames.
       numFrames = get(readerobj, 'NumberOfFrames');
       % Create a MATLAB movie struct from the video frames.
       for k = 1 : numFrames
            mov(k).cdata = vidFrames(:,:,:,k);
            mov(k).colormap = [];
       end
       % Resize figure based on the video's width and height
       % Playback movie once at the video's frame rate
       movie( mov, 1, readerobj.FrameRate);
       handles.filename=filename;
    end
% Update handles structure
guidata(hObject, handles);
% --- Executes on button press in Frameseparation.
function Frameseparation_Callback(hObject, eventdata, handles)
% hObject    handle to Frameseparation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
   filename=handles.filename;
readerobj = VideoReader(filename, 'tag', 'myreader1');
% Read in all video frames.
vidFrames = read(readerobj);
```

```
% Get the number of frames.
numFrames = get(readerobj, 'NumberOfFrames');
datatype='.jpg';
% Create a MATLAB movie struct from the video frames.
for k = 1 : numFrames
str=num2str(k);
mov(k).cdata = vidFrames(:,:,:,k);
mov(k).colormap = [];
 Frame=mov(k).cdata;
 Frame=rgb2gray(Frame);
 imshow(Frame);
 pause(0.01)
 filename=strcat(str,datatype);
 imwrite(Frame,filename);
end
 warndlg('process completed');
% --- Executes on button press in videos.
function videos_Callback(hObject, eventdata, handles)
% hObject    handle to videos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
number_of_frames=177;
filetype='.jpg';
str3='o'
Next=2;
cc=1;
for k = 1:177
        statframe=imread('1.jpg');
    % Write each frame to the file.
   aa=k-1;
```

```
    % PreviousFrame=strcat(num2str(aa),filetype);
    CurrentFrame=strcat(num2str(k),filetype);
    %PreviousFrame=imread(PreviousFrame);
    CurrentFrame=imread(CurrentFrame);
     imgOut1 = compensaterow(CurrentFrame);
    imgOut2 = compensatecol(CurrentFrame);
    MSE1  = CalculateMSE( imgOut1,statframe);
    MSE2  = CalculateMSE( imgOut2,statframe);
    MSE3  = CalculateMSE( CurrentFrame,statframe);
     MSE1=round(MSE1);
    MSE2=round(MSE2);
    MSE3=round(MSE3);
    out=[-MSE1,-MSE2,-MSE3];
  [min,index]=max(out);
     if index==1
     outputfile=strcat(str3,num2str(aa),filetype);
    imgOut1=uint8(imgOut1);
    imwrite(imgOut1,outputfile);
    elseif index==2
    outputfile=strcat(str3,num2str(aa),filetype);
    imgOut2=uint8(imgOut2);
    imwrite(imgOut2,outputfile);
    else
    outputfile=strcat(str3,num2str(aa),filetype);
    %
    imwrite(CurrentFrame,outputfile);
    end
end
helpdlg('Video Stabilization completed');
% --- Executes on button press in playvideo.
```

```
function playvideo_Callback(hObject, eventdata, handles)
% hObject    handle to playvideo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
str3='o'
Next=2;
cc=1;
filetype='.jpg'
 for k = 1:177
    % PreviousFrame=strcat(num2str(aa),filetype);
Frame=strcat(str3,num2str(k),filetype);
 a=imread(Frame);
imshow(a);
pause(0.0001);
 end
```

CHAPTER 7

RESULTS

Step 1 : Default GUI Window

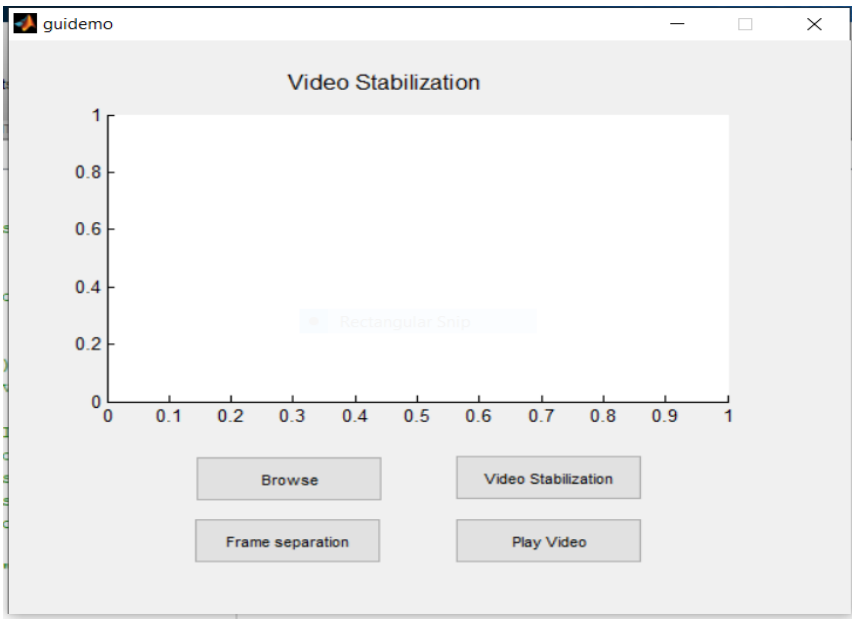


Fig 7.1

Step 2 : Select the Video to be stabilized

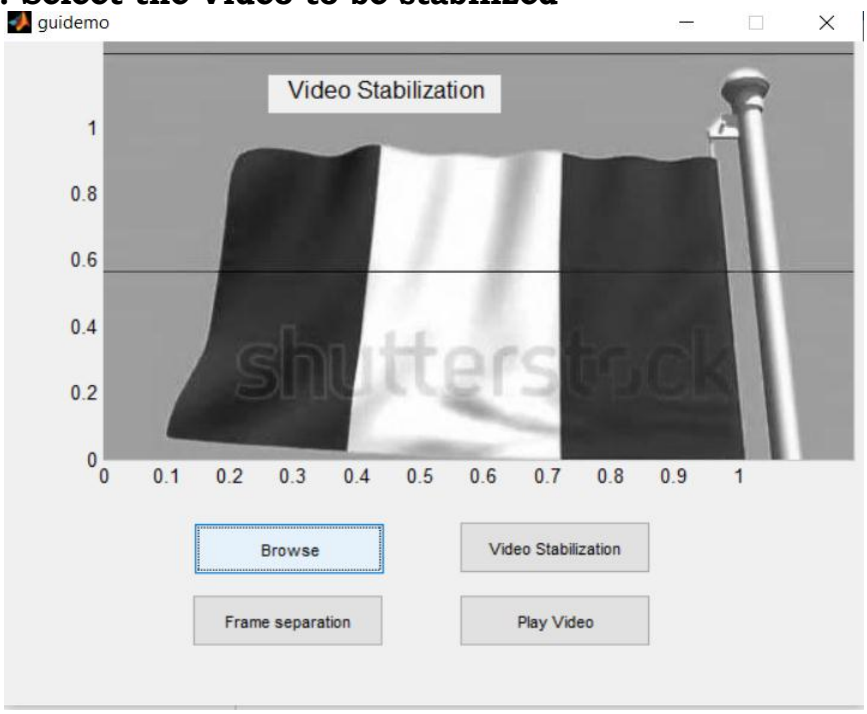


Fig 7.2

Step 3 : Frame Separation

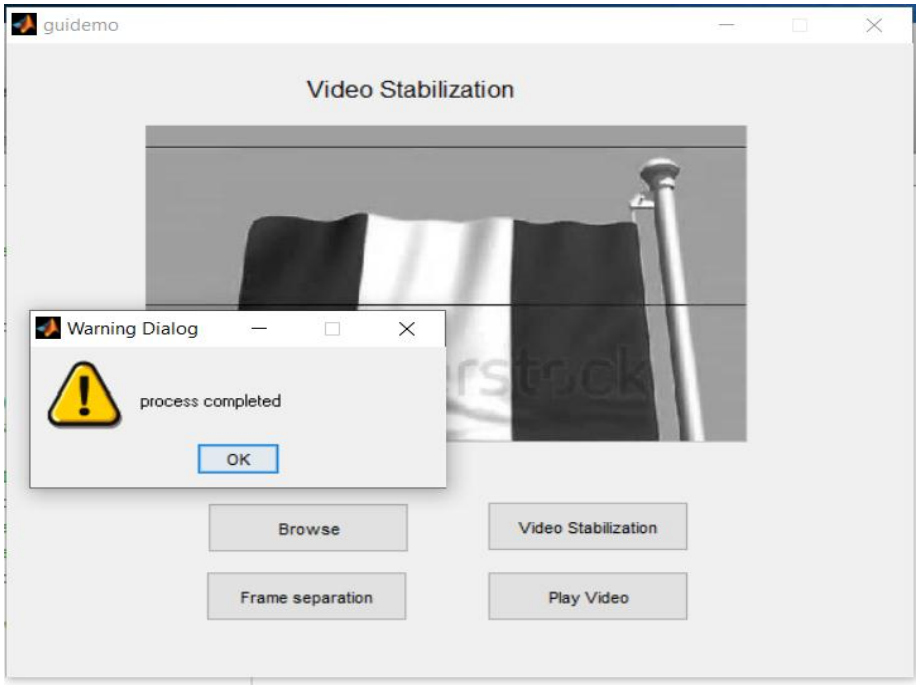


Fig 7.3

Step 4 : Video Stabilization Process

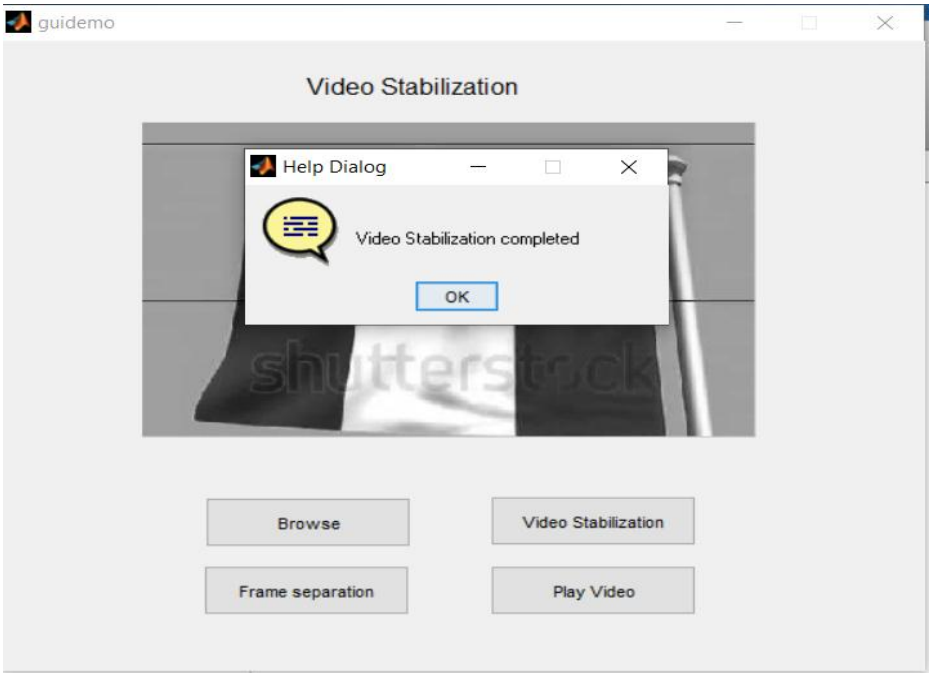


Fig 7.4

Step 5 : Play Stabilized Video

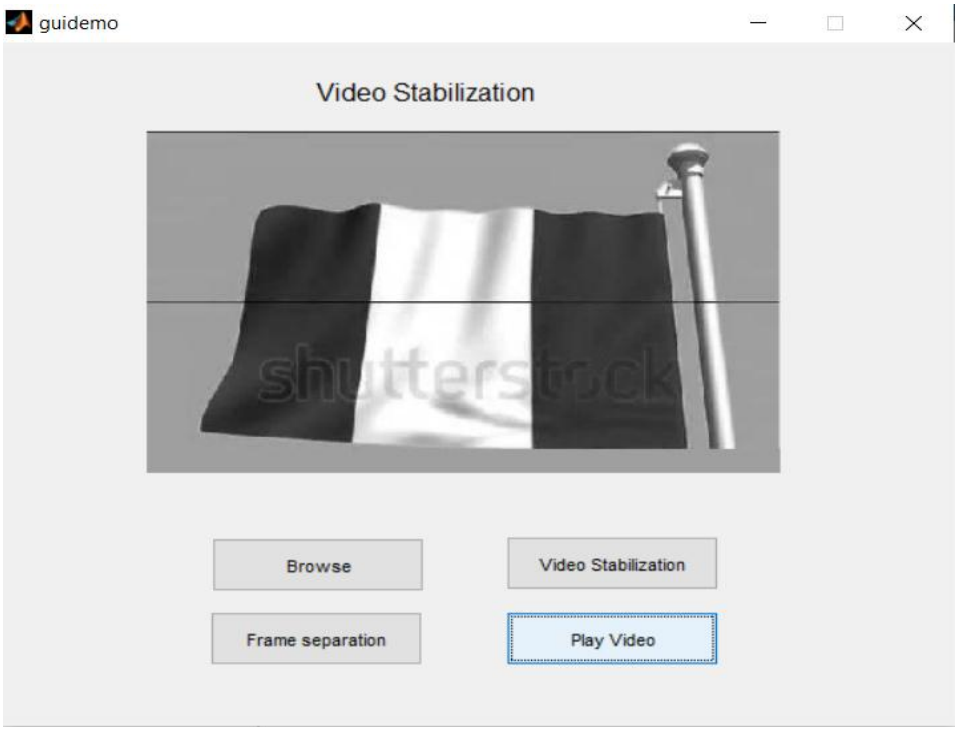


Fig 7.5

CHAPTER 8
CONCLUSION AND FUTURE SCOPE

8.1. Conclusion:

- The use of this method shows improved result in terms of stabilization. This technique is useful in enhancing the quality of low-grade video surveillance cameras.The video content has been stabilized.
- The degree of stabilization has been adjusted according to different demands.
- Able to handle stabilization of any unstabilized video as an input and having unlimited length videos.

8.2. Future Scope:

Fast detection technique is particularly helpful in identifying people, license plates, etc. from low-quality video surveillance cameras. In future, we can find better feature detector to overcome the consequences of extreme shaking of handheld camera in real time implementation for video stabilization.

Review of the various video filters, motion estimation, Video Processing Techniques used in Traffic Applications stabilization techniques are presented in the paper. Motion smoothening is the scope for the future the computation cost can also be reduced to improve the efficiency of the estimation and stabilization in future work.

REFERENCES

[1] Aleksandra Shnayderman, Alexander Gusev, and Ahmet M. Eskicioglu“An SVD-Based Grayscale Image Quality Measure for Local and GlobalAssessment “,IEEE 15(2), 2006.

[2]B.-Yu. Chen, K.-Yi. Lee, W.-T. Huang, J.-S. Lin, “Capturing Intention-based Full-Frame Video Stabilization,” Computer Graphics Forum, Vol. 27, No. 7, p.1805 - p.1814, 2008.

[3] C. Harris and M.J. Stephens, “A combined corner and edge detector”, Proc of Alvey Vision Conference, pp 147–152, 1988.

[4] Edward Rosten, Reid Porter and Tom Drummond, “FASTER and better: A machine learning approach to corner detection” in IEEE Trans. Pattern Analysis and Machine Intelligence, 2010, vol 32, pp. 105-119

[5] C. Morimato and R. Chelleppa, ”Evaluation of Image Stabilization algorithms”, Proc. of IEEE int. Conf. on Acoustics, speech and signal processing, Vol-5,pages 2789-2792, 1995.

[6]Jie Xu, Hua-wen Chang, Shuo Yang, and Minghui Wang “Fast Feature-Based Video Stabilization without Accumulative Global Motion Estimation” IEEE 2012.

[7]Ken-Yi Lee Yung-Yu Chuang Bing-Yu Chen Ming Ouhyoung “Video Stabilization using Robust Feature Trajectories” 2009 IEEE 12th International Conference on Computer Vision (ICCV).

[8]Keng-Yen Huang, Yi-Min Tsai, Chih-Chung Tsai, and Liang-Gee Chen “Feature-based Video Stabilization for Vehicular Applications” 2010 IEEE 14th International Symposium on Consumer Electronics.

[9] Labeeb Mohsin Abdullah, Nooritawati Md Tahir & Mustaffa Samad , “Video Stabilization based on Point Feature Matching Technique”, 2012 IEEE Control and System Graduate Research Colloquium (ICSGRC 2012).

[10]Xie Zheng, Cui Shaohui, Wang Gang, Li Jinlun “Video Stabilization System Based on Speeded-up Robust Features” International Industrial Informatics