GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

OBJECT DETECTION AND TRACKING (LIVE DETECTION)

A Report for the Evaluation 3 of Project 2

Submitted by

SAURABH PATEL

(1613105106)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING WITH SPECIALIZATION OF CLOUD COMPUTING AND VIRTUALIZATION

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of
Dr. K SAMPATH KUMAR, M.E., Ph.D.,
Professor

APRIL / MAY- 2020

GALGOTIAS UNIVERSITY

SCHOOL OF COMPUTING AND SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report **“OBJECT DETECTION AND TRACKING (LIVE DETECTION)”** is the bonafide work of **“SAURABH PATEL (1613105106)”** who carried out the project work under my supervision.

SIGNATURE OF HEAD	**SIGNATURE OF SUPERVISOR**
Dr. MUNISH SHABARWAL, PhD (Management), PhD (CS)	Dr. K SAMPATH KUMAR, M.E., Ph.D.,
Professor & Dean,	**Professor**
School of Computing Science & Engineering	**School of Computing Science & Engineering**

Table of Contents

ABSTRACT

Object detection and tracking are important and challenging tasks in many computer vision applications such as surveillance, vehicle navigation, and autonomous robot navigation. Video surveillance in a dynamic environment, especially for humans and vehicles, is one of the current challenging research topics in computer vision. It is a key technology to fight against terrorism, crime, public safety and for efficient management of traffic. The work involves designing of the efficient video surveillance system in complex environments. In video surveillance, detection of moving objects from a video is important for object detection, target tracking, and behavior understanding. Detection of moving objects in video streams is the first relevant step of information and background subtraction is a very popular approach for foreground segmentation. In this thesis, we have simulated different background subtraction methods to overcome the problem of illumination variation, background clutter and shadows. Detecting and tracking of human body parts is important in understanding human activities. Intelligent and automated security surveillance systems have become an active research area in recent time due to an increasing demand for such systems in public areas such as airports, underground stations and mass events. In this context, tracking of stationary foreground regions is one of the most critical requirements for surveillance systems based on the tracking of abandoned or stolen objects or parked vehicles. Object tracking based techniques is the most popular choice to detect stationary foreground objects because they work reasonably well when the camera is stationary and the change in ambient lighting is gradual, and they also represent the most popular choice to separate foreground objects from the current frame. Surveillance networks are typically monitored by a few people, viewing several monitors displaying the camera feeds. It is very difficult for a human operator to effectively detect events as they happen. Recently computer vision research has to address ways to automatically some of this data, to assist human operators. The techniques studied, implemented and presented are all premeditated in detail and then put into practice in this thesis.

List of Figures

List of Tables

Chapter 1

1. Introduction

1.1 Problem Statement

The basic concept behind object detection in videos engrosses the verification of the presence of an object in image sequences and possibly locating it in particular for recognition. Object tracking is to monitor an objects spatial and temporal changes during a video sequence, including its presence, position, size, shape, etc. This is done by solving the temporal correspondence problem, the problem of matching the target region in successive frames of a sequence of images taken at closely spaced time intervals. These two processes are closely related. Detection is the basis for tracking and it usually starts with detecting objects, while detecting an object repeatedly in subsequent image sequence is often necessary to help and verify tracking. The general problem of motion understanding and tracking of the moving objects is one of the most interestingly used areas of computer vision. Tracking is the problem of generating an inference about the motion of an object, given a sequence of images. Several image-based motion tracking systems have been developed in the past. These systems include one from the MIT AI lab [1] [2], the W^4 System of UMCP [3], and one from CMU [4]. However, these systems are computationally intensive and generally require very high performance computers to achieve real-time tracking. For instance, the tracking system of MIT AI lab used an SGI O2 workstation with R10000 processor to process images of 160x120 pixels at a frame rate up to 13 frames per second. Some systems used multiple cameras, each covering a fixed field of view. Some other systems used adaptive and model-based algorithms that required extensive training for recognizing specific objects and/or scenes. Digital video dispensation is becoming widely used in many aspects of our nowadays life. The availability of high-computation-power systems make it possible the processing of huge amount of raw data to achieve substance based functionalities, such as search and manipulation of objects, semantic description of scenes, detection of unusual events, and recognition of objects. Video tracking is a vital and active research area in computer vision.In its simplest form, tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. In other words, a tracker assigns consistent labels to the objects in question in different frames. Object tracking no doubt is an exigent problem. Impenetrability in tracking can arise due to abrupt motion of object, changing directions and appearances of the object and the scene, non rigid or articulated object structures as birds and human beings, object- to-object and object-to-scene occlusions, and camera motion [5]. Tracking is generally performed in the milieu of higher-level applications that require the location and/or shape of the object in every frame. Normally, assumptions are made to constrict and lessen the tracking problems.

1.2 Scope and Objective

Moving object detection is important in many real-time image processing applications such as autonomous robotics, traffic control, and driver assistance and surveillance systems. Usually high resolution gray-scale images must be processed; since each image pixel may belong to a moving object, pixel-wise processing is required.

Now days, video surveillance is an important and challenging field in computer vision for both indoor and outdoor environments. Organizations which need a surveillance system can easily get low priced surveillance cameras but they still need many security agents to keep a constant watch on all monitors. This approach is not efficient, and in fact most of the time video tapes or files are replayed a number of times to check on a particular event after it has happened thus the automation of this system is highly desired.The project Moving object tracking from video sequences is an attempt to study some algorithms, which are robust for the tracking of mobile but non rigid objects from the image sequences precisely called video. The Figure 1-1 shows the basic working grid of the processes involved in the system as Figure 1-1 also includes pre and post processing sequences as well which includes noise removal, image enhancement issues, organization and classification etc. but these issues are not of primary importance as far as this project is concerned so are not addressed in detail in this project.

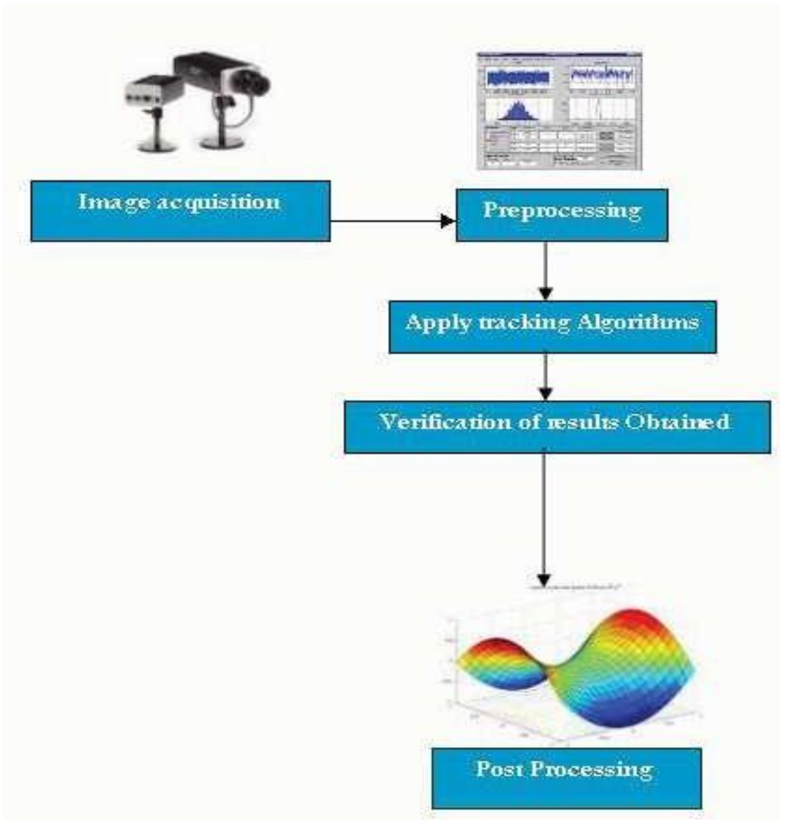


Figure 1-1- Block Diagram of system

1.3 Applications

Object detecting and tracking has a wide variety of applications in computer vision such as video compression, video surveillance, vision-based control, human-computer interfaces, medical imaging, augmented reality, military applications, traffic monitoring and robotics. Bayesian classification methods have been broadly used in an assortment of image processing applications, including medical image analysis. The basic procedure is to combine data-driven knowledge in the likelihood terms with clinical knowledge in the prior terms to classify an image into a pre-determined number of classes. Major relevance lies in tissue tracking and for Brain MRI classifications [6]. Moreover, augmented reality also offers diverse application fields for tracking yet accurate and real time tracking problem is not solved still this field is emerging day by day .Most commonly known application is Amines which work on the principle of virtual reality .Another such system developed on the basis of tracking usage in Augmented reality is Knowledge-based Augmented Reality for Maintenance Assistance (KARMA)[7] and A Mobile Augmented Reality Systems for Exploring the Urban Environment (MARS)[8]. Additionally, it provides input to higher level vision tasks, such as 3D reconstruction and 3D representation. It also plays an important role in video database such as content-based indexing and retrieval. In today s technologically emerging world this very technology of moving object detection is used for the movement assistance of disables in their homes. Video tracking is also used in automated digital recordings of animal behavior which includes conditional training/reward or punishment based on position in arena (e.g. demand-feeders, shuttle box etc.) such kind of system in practice is recently developed by *Qubit Systems* as Video tracking software . Video Tracking is usually taken in context of Real-time tracking of x, y coordinates and it is one of the most widely applicable usages of tracking draw on the control of experiments in addition to many other conventions.

1.4 Thesis Outline

Chapter 1 includes the introduction of the problem with its objective and scope along with applications. The concept of object detection and its methods along with the method studied and implemented which is background subtraction is all engrossed in Chapter 2. Object tracking in detail and its techniques are described in chapter 3. Similarly, chapter 4 depicts the modus operandi used for object tracking during this project along with their experimental results including Template matching and Fast Mean Shift. The Kalman filter and it necessary details along with implementation complexities, algorithm and results are described in chapter 5 .It also includes the comprehensive comparison of techniques used. In the end whole of the work is concluded and few future work aspects are mentioned in chapter6.

Chapter 2

2. Object Detection

One of the most difficult problems in image processing is detecting particular objects in an image. For a human observer it is very easy to identify any object, however it is far more difficult for a machine. Numerous methods exist for detecting objects of known type in a particular environment or image. However, in many cases, the visual characteristics of the objects are unknown, or it is necessary to detect objects that are very different from each other. This kind of method has applications in the domain of robotics, particularly for robots that are designed to operate in a hostile or unknown environment.

2.1 Introduction of Object Detection

Videos are the sequences of frames that run fast enough to give an effect of continuity as human eye perceives the frame sequences moving with a particular speed as a video. As far as object detection is concerned techniques of image processing are applied to the frames in order to identify any change so as to state the motion of object detected through the change observed in the two consecutive frames after attaining the results of the image processing techniques used for identification purpose. A surveillance system can be implemented in three steps. The first step consists of *detecting* the objects in motion .Then *tracking* them and finally High-level interpretation of the ongoing events. First step of detecting the object in question is described in detail with the experimental results in order to elaborate the mechanism in Chapter 2. Object detection is the first step in the motion tracking phenomena. The object detection is performed through background subtraction algorithm in this project though many other detection methods have been already developed and are widely in use.

2.2 <u>Background Subtraction</u>

Identification of moving objects from a sequence of frames is a primary and critical task in many computer-vision applications. A common approach is to carry out background subtraction, which identifies the specific moving objects from the segment of a video frame that differs distinctly from a background model. There are a number of challenges in developing a good background subtraction algorithm. First, it must be robust against changes in illumination. Second, it should avoid detecting non- stationary background objects such as moving leaves, rain, snow, and shadows cast by moving objects. Other object detection methods include:

1. Pixels based method
2. Optical Flow method
3. Color based method
4. Gradient based method
5. Frame differencing
6. Median filter
7. Linear predictive
8. Non parametric method
9. Mixture of Gaussian

2.2.1 <u>Pixels-based Method</u>

One of the first methods described in literature was from Nagasaka et al [9] in 1991. Shot changes are detected using a simple global inter frame difference measure, defined as:

Detection if:

$$\left(\left|\sum_{i=1}^{X}\sum_{j=1}^{Y}P(I_t,i,j)-\sum_{i=1}^{X}\sum_{j=1}^{Y}P(I_{t-1},i,j)\right|\right)>T \qquad (2.1)$$

resulting in operations O(*P*) per frame (as the second term of the difference has been already obtained after the processing of the previous frame I $_{t\ 1}$).
Frame difference method has been studied and used in this project.

2.2.2 <u>Optical Flow</u>

The optical flow is the disarticulation field allied to each of the pixels in a sequence. Such displacement field results from the apparent motion of the image brightness in time. For the computation of optical flow it is assumed that image brightness is continuous and differentiable as many times as needed in both the spatial and temporal domain. Estimating the optical flow is fundamental problem in low- level vision, and can be undoubtedly serve for many applications in image sequence processing. Most of the algorithms for the estimation of the optical flow concentrate on the goal of estimating the motion field between succeeding images in a sequence, disregarding the estimates obtained for the previous image pair.

If the apparent brightness of moving objects remain constant then the image brightness *E* over time is given by

$$\frac{dE}{dt} = 0 \qquad (2.2)$$

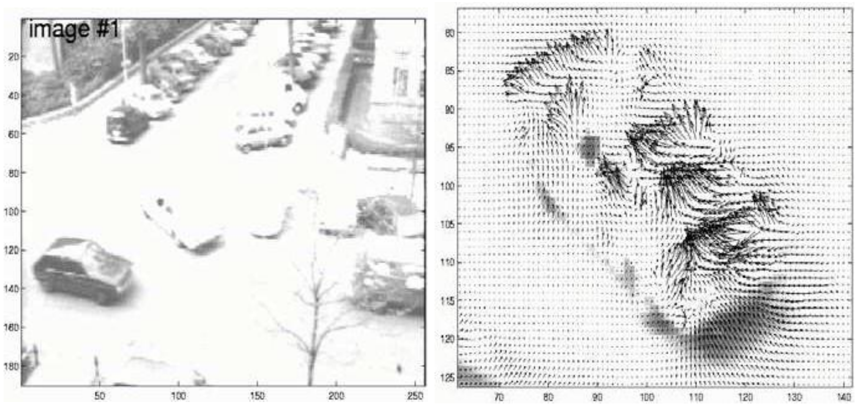


Figure 2-1: Application in Visual Surveillance: Optical Flow computed on Carthe Hamburg taxi sequence

Since image brightness *E* is regarded as a function of both spatial coordinates of the image plane, *x* and *y*, and of time, that is,

$$E = E(x, y, t) \qquad (2.3)$$

2.2.3 Color based Method

This method is based on background modeling and subtraction using both color and edge information. Both the color and edge models and subtraction are computed separately [10]. For storage of results confidence maps are used representing how confident the method is in recognizing that a pixel is a foreground object. Color and intensities of the previous image are compared with every new coming image and this difference signifies the motion. Background subtraction is done by performing the color-based subtraction and the edge-based subtraction separately and then combining the results [10].
Color-based subtraction is performed by subtracting the current image from the mean image in each color channel. For each pixel, the confidence is computed as

$$C_M = \begin{cases} 0 & D < m_c\sigma \\ \dfrac{D - m_c\sigma}{M_c\sigma - m_c\sigma} \times 100 & m_c\sigma \le D \le M_c\sigma \\ 100 & D > M_c\sigma \end{cases} \qquad (2.5)$$

A significant change in any color channel indicates motion and thus a foreground region is detected.

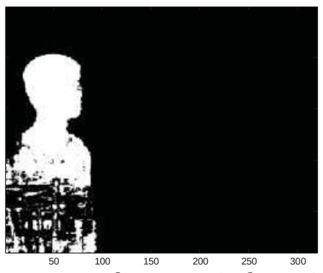


Figure 2-2 : Shows the color confidence maps for a certain frame in the sequence used

2.2.4 Gradient based method

As the back ground subtraction method has some recognized problems for instance quick illumination changes, shadows, reflection of the objects and orientation of the moving bodies .The gradients of image are moderately less perceptive to abrupt changes in illumination and thus can easily be combined with color information to perform background subtraction.

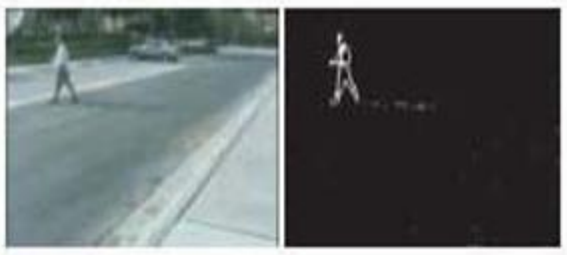


Figure 2-3: Gradient based background subtraction in a certain frame

2.2.5 Frame Differencing

Frame differencing method is used for this purpose results after applying the simple algorithm for subtraction of successive frames from the previous one. Frame differencing questionably the simplest background modeling technique, frame differencing uses the video frame at time t - 1 as the background model for the frame at time t. Since it uses only a single previous frame, frame differencing may not be able to identify the result.

$$|\text{frame } i - \text{frame } i\text{-}1| > \text{Threshold}$$

2.2.6 Experimental Results

The results achieved after applying the background subtraction algorithm using Frame Differencing technique are illustrated as under.

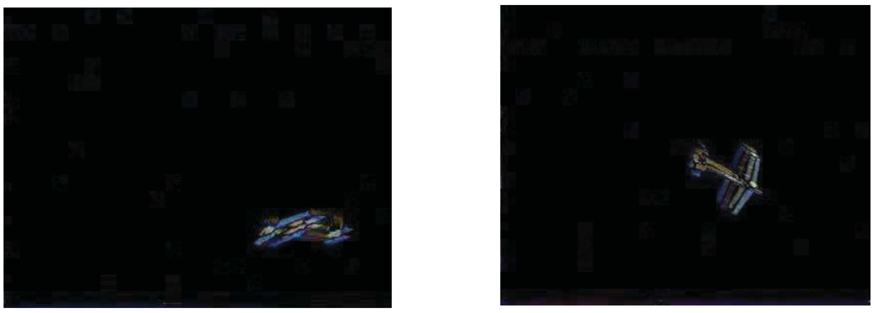


Figure 2-4: The video results showing background subtraction in Frame 10 and 15

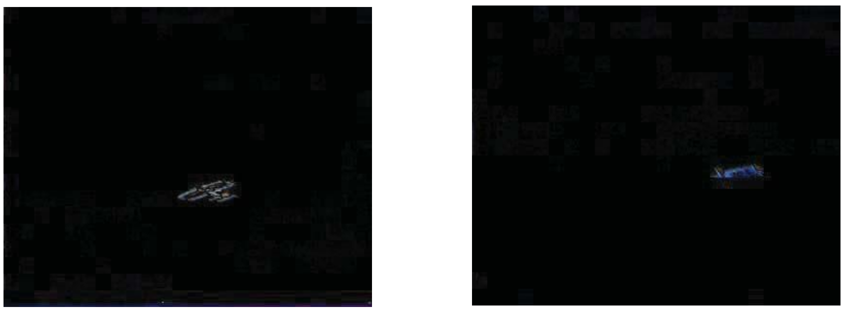


Figure 2-5: The video results showing background subtraction in Frame 42 and 45

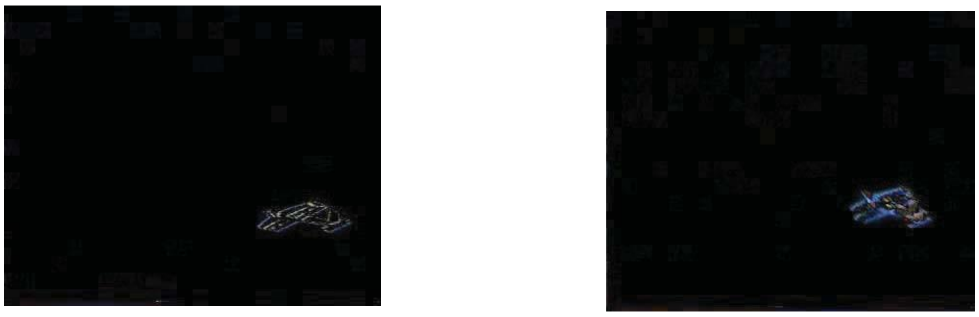


Figure 2-6: The video results showing background subtraction in Frame 86 and 92

In few cases where the background illumination or brightness becomes higher up to a certain level that it becomes detectable after subtraction like the object it also appears as shown in figure 2-4.

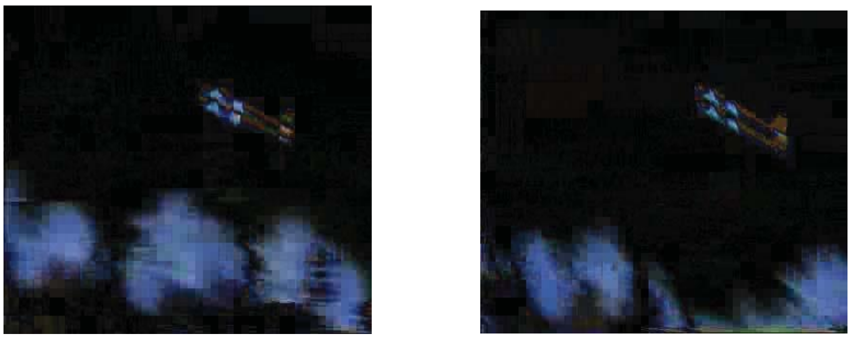


Figure 2-7: Video Results showing high illumination in the background in frame 56 and 58

So, the experimental results shown in Figure 2-1,2-2,2-3 show the object in question which is a plane clearly detected from the background .Similarly, Figure 2-4 shows another dimension of the background subtraction algorithm that it has to be robust enough in order to overcome the raising illumination and brightness factors .

2.3 <u>Conclusion</u>

Object detection is the lying at the start in the hierarchy of motion based tracking mechanism. It basically deducts the background and focuses the object of interest from the scene. Many methods can be used for background subtraction depending on color, edge and such kind of features .Some of the methods are described in brief in the preceding chapter .It is effectively used in real time applications as well .It is also feasible in correcting images deserts caused by inappropriate illumination and brightness effects .Thus background subtraction is one of the widely used and important technique as far as object detection in concerned.

Chapter 3

3. Object Tracking

The aim of object tracking is to establish a correspondence between objects or object parts in consecutive frames and to extract temporal information about objects such as trajectory, posture, speed and direction. Tracking detected objects frame by frame in video is a significant and difficult task. It is a crucial part of smart surveillance systems since without object tracking the system could not extract cohesive temporal information about objects and higher level behavior analysis steps would not be possible [12].

3.1 Object Tracking Introduction

The detection and classification of moving objects is an important area of research in computer vision. The problem assumes immense importance because of the fact that our visual world is dynamic and we constantly come across video scenes that contain a finitely large number of moving objects. To segment, detect, and track these objects from a video sequence of images is possibly the most important challenge that the vision experts confront today.

3.2 Model of Tracking System

A tracking system may be modeled as a three-state sequential machine. The functional hierarchy goes as Locking state**,** Tracking State and Recovery State.

3.2.1 Locking state

Initially the system is in locking state, when the camera is in search mode, i.e., searching for targets. During this state the processing is carried out on the whole image frame. The system will partition the image frame captured by camera into a number of moving objects. The history of these objects is extracted by checking the trajectory followed by the objects, and confirmation of the moving object is carried out in automatic mode. Once the target is confirmed the control of the system is transferred to tracking state.

3.2.2 Tracking state

This stage should use computationally inexpensive techniques. Current location extracted by locking state is used for processing. Next position of the target is identified, and that positional information is stored in history database. If the target does not exist in the predicted window area, then the system control is transferred to recovery state.

3.2.3 Recovery state

Quite often the moving object of interest may be lost temporarily or permanently. In this state if the target is lost, the system will try to recover the target from low-resolution image. If the target is recovered in a few frames, then the system will transfer control to tracking state; otherwise it remains in recovery state till its predefined time expires. After the time is elapsed, control transfers to locking state.

3.3 Different Approaches for Object Tracking

Different methods have been used for moving object tracking given as under.

1. Correlation based method
2. Feature based method
3. Histograms method
4. Gradient based method
5. Contour based method
6. Kernel based method
7. Kalman Filter
8. Extended Kalman Filter
9. Particle Filter

3.3.1 <u>Correlation based method</u>

The correlation based method simply operates on the principal of intriguing correspondence between the previous frame and every up coming frame. This mechanism finally results in the correlation vector of the previous and new frame which in fact gives the measure of relationship between both frames. Thus the difference indicates the motion and determines the new direction for the object of interest. Relatively similar approach is followed in template matching which is described in detail in chapter 3.In template matching the correlation is taken between the template and every up coming frame in the sequence which gives the similarity ratio in the form of a single point which is then resolved by plummeting the components around it to make it template for the next frame. Correlation value differs between -1 to 1 so whenever correlation methods is used in tracking some threshold is set by the .This threshold servers as a reference as how much the images of previous ad current frame match each other or differ from each other which finally determines the motion of the object if interest.

3.3.2 <u>Feature-based Methods</u>

In feature-based object detection, standardization of image features and registration (alignment) of reference points are important. The images may need to be transformed to another space for handling changes in illumination, orientation and size. One or more features are extracted and the objects of interest are modeled in terms of these features. Object detection and recognition then can be transformed in to a graph-matching problem.
All the methods we have already presented were using features, but they can be qualified of trivial features. This method considers more sophisticated ones. We consider:

- The moments computed on the image
- The contour lines extracted from the image

3.3.3 Histogram-based Method

It is also possible to compare two images based on global features instead of local features (pixels). Histogram is a global image feature widely used in image processing. The main advantage of histogram-based methods is their global aspect. So These methods are more robust to camera or object motion. The main drawback appears when we compare two different images having a similar histogram. It will often results in missing a shot change. Different uses of the histogram can be distinguished. Some methods only compute differences between histograms and then the quality of the result is linked to the kind of histogram considered. A first extension is the use of weighted differences between histograms. Another approach consists in the definition of an intersection operator between histograms or the definition of different distances or similarity measures.

Object tracking is a wide field in which many methods can be implemented to achieve results some of the other methods studied and implemented during this project tenure are described in forth coming chapters.

3.4 Design Considerations in Object Tracking

The following design considerations may be incorporated in an object tracking system.

3.4.1 Stationary Background

When the scene contains multiple objects, the background is stationary while all or part of the objects in the foreground may be in motion.

3.4.2 Target size variation

The target size reduces as the target moves further away from the camera. Thus a scaling mechanism needs to be incorporated during the process of tracking.

3.4.3 Occlusion or Temporary loss of target

During the tracking phase the target may be temporarily lost as it goes behind another object. This is known as occlusion. In such cases the system will recover the target automatically.

3.4.4 Target Model

The model of the target needs to be incorporated. In case of human tracking, for example, a human figure may be modeled as an ensemble of several ellipses, where each ellipse represents the individual body parts like head, torso, hands, and legs, etc. The color, shape, intensity, and other attributes of the object may vary while the object is in motion, and yet the tracker should be able to track correctly.

3.4.5 Automatic Target detection

The tracker should be able to detect all the new targets automatically and start tracking them.

3.4.6 <u>Real time</u>

The tracking algorithm should be computationally simple and optimum so that the tracking can be implemented in real time.

3.4.7 <u>Target trajectory</u>

The target may or may not follow a particular trajectory. There may be abrupt changes in the target path.

3.4.8 <u>Target speed</u>

Speed of the target can change abruptly; it may be constant, increasing, or decreasing.

3.5 <u>Conclusion</u>

Object tracking is described in this chapter along with a introduction to several methods in use. Object tracking is meant to follow the position and direction of the object of interest in a sequence .Many multipurpose applications are there in present day world regarding object tracking which makes it a flourishing field in research and development .It is equally applicable for the systems based on real time estimations as in defense sector and surveillance systems .

Chapter 4

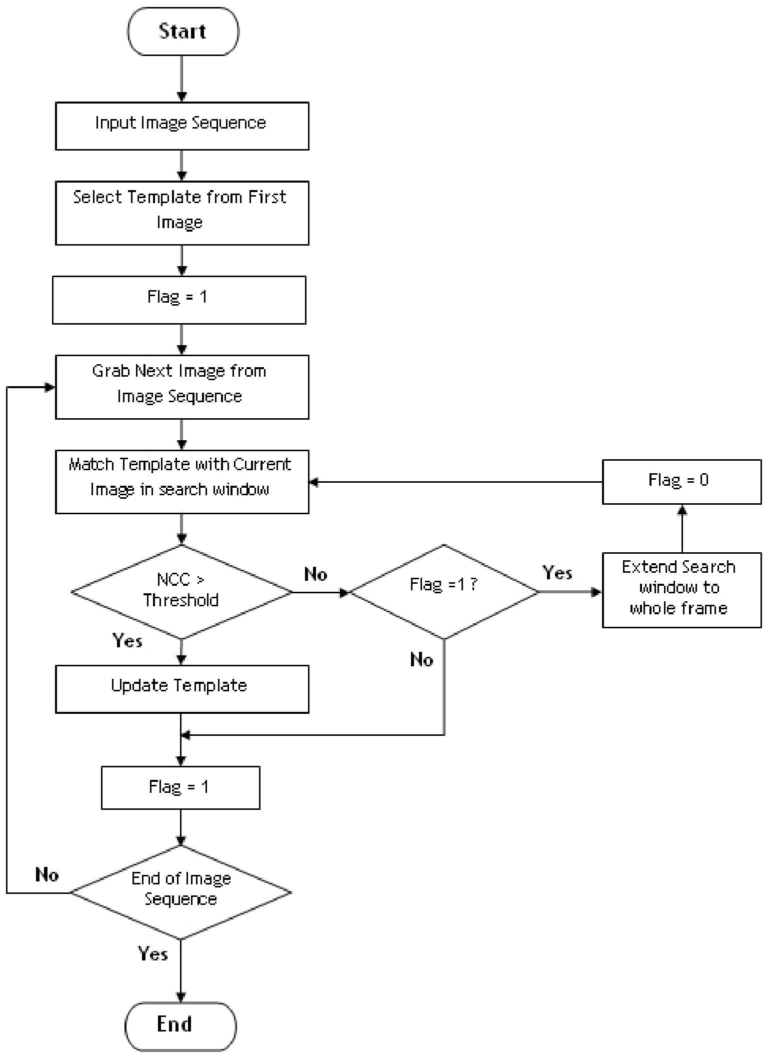
4. Object Tracking Modus Operandi

Object Tracking methodologies vary from mathematics to applications to a wide extent. This variation offers a wide spectrum of ample techniques to be used for object tracking .This is one of the most flourishing fields of research in computer vision specifically from surveillance and defense point of view. As the applications of tracking are in highly sensitive domains which require extreme accuracy and precision .This demand of accurateness increases when real time scenarios are under consideration as in defense sector missile tracking, jet tracking etc is concerned.

4.1 Template Matching

Template matching is a simple task of performing a normalized cross-correlation between a template image (object in training set) and a new image to classify. Template-based tracking using the sum-of-squared differences (SSD) is a classic technique for maintaining the location of a target throughout an image sequence.
The idea of template-based tracking is to track a moving object by defining a region of pixels belonging to that object and, using local optimization methods, to estimate the transformation parameters of that region between the reference image and the new image. The reference image can be fixed as the first frame or chosen to be the previous frame.
The goal of template-based tracking is to maintain a model of the target in terms of a 2D template of image intensities and compute the target location in a new image frame by comparing the new data with that of the template. The data are usually compared using a low-order parametric motion model such as translation or affine, and the optimal location is computed using either discrete correlation search or non-linear function optimization. Template-based matching algorithm is usually simple, effective and computationally efficient.

4.1.1 <u>Flow Chart representation</u>



Start
Input Image Sequence
Select Template from First Image
Flag = 1
Grab Next Image from Image Sequence
Match Template with Current Image in search window
Flag = 0
NCC > Threshold
No
Flag =1 ?
Yes
Extend Search window to whole frame
Yes
No
Update Template
Flag = 1
No
End of Image Sequence
Yes
End

Figure 4-1: Flow chart representation of Template based Target Tracking

4.1.2 <u>Experimental Results</u>

After applying the algorithm of template based tracking the results achieved are as following:

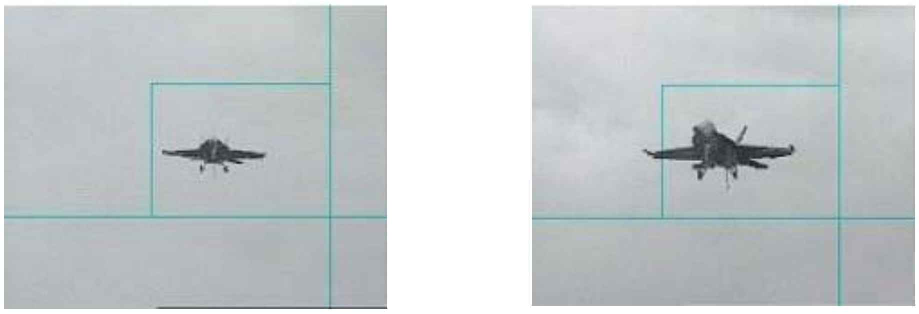


Figure 4-2: Video results showing Target tracking using template matching in frame 7 and 11

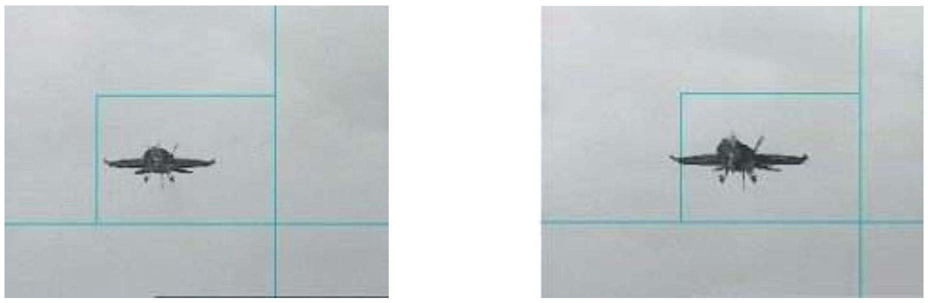


Figure 4-3: Video results showing Target tracking using template matching in frame 52 and 75

4.1.3 Complications using Template Matching Technique

There is a possibility, especially in air borne objects, that there may be significant change in target shape or orientation in very next frame as shown in following Figure 4-4. In this case, tracker starts futile tracking and correlation value drops due to which template is not updated and in next 2 or 3 frames tracker completely looses the object.

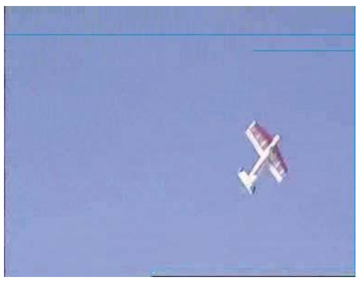


Figure4-4: Change in orientation of the object

The template selected for tracking purpose through template based tracking using correlation approach is shown in figure 4-5.

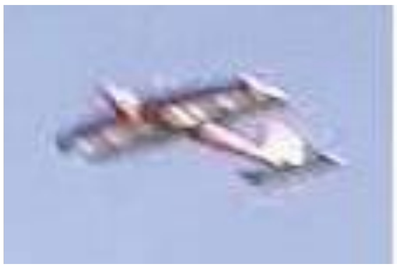


Figure 4-5: Template selected for Tracking

Hence, in such a condition when the orientation changes are gigantic for the tracking algorithms to keep pace with results in occlusions. Occlusion is a very common problem in tracking domain regarding directional changes. Here in Template matching technique used the template is updated by every next frame (image) .In such a situation one other problem may arise when the object in question leaves the screen area for sometimes and enters back after few frames .During that time in the absence of object the algorithm keeps on selecting template from the background and when the object re enters the area of concern it doesn t match with that previously selected template and thus the system fails.

Some results have been taken from a video in order to show this phenomenon shown in Figure 4-6

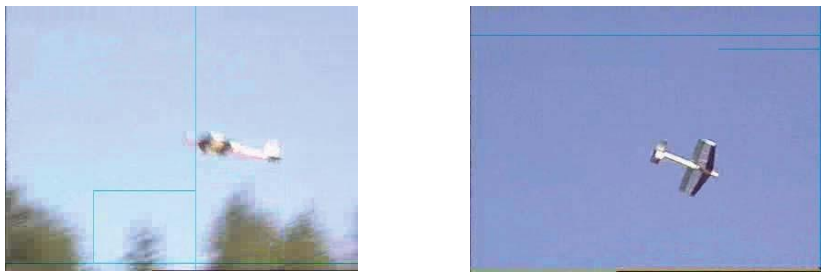


Figure 4-6: Tracker losing the object

4.1.4 <u>Template Selection</u>

A critical question in template-based tracking is how to select the template. One approach is to use the appearance of the target in the first image frame, with the template remaining constant throughout the sequence. The advantage of this approach is that the tracker always uses data, which is known to be trustworthy. However, the drawback is that the algorithm does not adapt to changes in the appearance of the target over time. The target is likely to be lost when it rotates out of plane or undergoes non-rigid transformations. An alternative approach is to use the appearance of the target in the previous image frame, so that the tracker always adapts to changes in appearance. The disadvantage of this approach is that the tracker tends to drift away from the original target over time, since there is no guarantee that the newly computed location of the target is without error.

This *reference frame* dilemma is not limited to template-based tracking: Histogram trackers are also faced with the choice of which image to use as a reference, with many of them selecting the first frame. The template used for experimentation is shown in Figure 4-7. Now this template is selected in a position when it is evidently visible and enlarged from the normal view which serves positively while using template based tracking. It helps the algorithm not to fail when in instances during the video the camera enlarges an object though it s not the mere solution to deal with zooming but it is an edge to let the algorithm work successfully in such conditions.

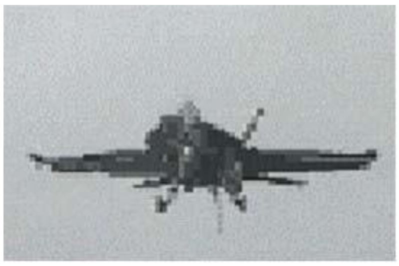


Figure 4-7: Template selected for tracking

Thus template selection is an issue which holds prime importance while implementing and template based tracking .The algorithm used in this project works on adaptive Template selection means every previous frame becomes template for the next frame to be matched with still it needs a previously selected template to instigate the process. Success of the progression largely depends on the kind of template selected for the very first time .Template matching is weaker in this case to deal with rapidly altering orientations and gives rise to occlusion and one major factor responsible for the phenomenon can also be template selection. As, any absurd value of cross correlation obtained can head the algorithm to failure. It also varies a lot for rigid and non rigid bodies as here only rigid bodies are of prime concern but if human or birds is to be tracked using template based method it becomes very critical when it comes to template selection. As, non rigid bodies change their shape and orientation at every next move for example a human walks and moves his arms which changes his posture and orientation similarly birds flutter all the time to keep their balance maintained in the air which makes them vary their shape at each flutter .
So, overall template selection is a core concern while dealing with template based tracking methodology.

4.1.5 <u>**Advantages of Template Matching**</u>

Some advantages of template matching are its simplicity and its suppleness. Its correlation-based algorithm is uncomplicated to implement and it allows us to use templates that have either been produced using a segment of the image or derived.

4.1.6 <u>**Disadvantages of Template Matching**</u>

On the other hand, one of the main disadvantages of template matching is its computational cost since correlating requires the "scrutiny" of each pixel in the template over several windows in the image. Another disadvantage of template-based tracking is that if shape or orientation of target is changed then tracker starts loosing target and eventually results in futile tracking.

4.1.7 <u>**Conclusion**</u>

Template Matching serves as a vigorous motion detection technique unless the intrusion of any occlusion phenomena or target loss from the region of interest .It serves as a technique which can be used effectively in present day tracking systems by making the template adaptive. Though there are complications in the methodology still template matching is a widely used and accepted technique. This technique is growing.

4.2 Mean Shift

The Mean shift algorithm is a nonparametric technique to locate density extrema or modes of a given distribution by an iterative procedure. The method was originally proposed by Fukunaga and Hostetler [13]. Cheng [14] generalized the method and pointed out, that the mean shift algorithm is a mode-seeking process on the density function surface. Comaniciu and Meer [15] proved the convergence of the iterated mean shift procedure on discrete data, proposed several extensions and presented its benefits for practical applications. Belenzai and Bernhard [16] stretch the conservative step of mean shift method and introduced fast mean shift method.

4.2.1 Mean Shift Tracking

mean shift vector for discrete data can be represented in general by the difference between the weighted mean computed with kernel profile *g(x)* and **x** (the kernel or window center)

$$m(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x - x_i}{h}\right\|\right)} - x \qquad (4.2.1)$$

…

Where n is the number of data points and h is the size of the kernel [15]. The difference image is the outcome of a change detection process by forming the difference between the current image and next image of an image sequence. The difference image generated from given image sequence is contains large number of high-intensity peaks or modes. Our principal objective is to find modes representing our desired object. The search process is facilitated by information of expected scaling of object $\{H(y_i),W(y_i)\}$ at a given vertical image location y_i, which can be obtain by a rough calibration of the image sequence. H and W are the height and the width of the object in pixels. Mode detection within the difference image I is performed by the following steps. The difference image intensity maximum is mapped to unit intensity and its entire range is scaled proportionally. A sample set of n points is defined by locating local maxima. Local maxima are found by:

- Locating the global intensity maximum and adding it to the list of sample set
- Resetting the difference image intensity around the found matrix within a window of size $\{0.5H(y_i), 0.5W(y_i)\}$
- Repeating the maximum search of step (1) until the found maximum drops below a threshold T_1.

The points of the sample set are subsequently used in a mean shift procedure. The final result does not depend critically on T_1. A very low value just increase the run time and generate more outliers, which have to be eliminated after computation of mean shift vector component.The mean shift procedure is applied to the points of the sample set with a window size of *$\{H(y_i), W(y_i)\}$* according to the local scaling. For the two-dimensional probability distribution in the difference image, the mean shift vector *(m_x, m_y)* computation using uniform kernel can be defined as:

$$m_x(x, y) - \frac{\sum_{i=1}^{n} x_i I(x_i, y_i)}{\sum_{i=1} I(x_i, y_i)} x \qquad (4.2.2)$$

$$m_y(x, y) - \frac{\sum_{i=1}^{n} y_i I(x_i, y_i)}{\sum_{i=1} I(x_i, y_i)} y \qquad (4.2.3)$$

Starting out from the points of the sample set, the mean shift vector is computed repeatedly until convergence, locating the closest mode typically within 3 to 4 iterations. Note that the mean shift vector computation requires the computation of the zeroth and first moments of the distribution within the window. The computed zeroth moment can be interpreted as the probability density sampled at distinct locations *(x,y)* of the convergence path:

$$p(x, y) - \frac{1}{W(y)H(y)} \sum_{i=1}^{n} I(x_i, y_i) \qquad (4.2.4)$$

The above formula yields a relative measure for the presence of an object-moving region.

The set of probability measures $\{ p(x, y), \ldots\ldots p(x, y)_k\}$ computed for the points of the convergence path is used later on to validate object location. The size of the sample set is usually on the order of several hundred points, which represents considerable computational complexity when targeting real-time operation. To achieve faster mode seeking fast variant of mean shift computation using integral images is described in next section. The convergence points of individual mean shift procedures are linked together forming the centers of detected clusters. Linking is carried out analogously by merging all points, which are closer in *x*- and *y*-direction.

4.2.2 <u>**Fast Shift Method**</u>

Belenzai and Bernhard [17] proposed Fast Mean Shift Method to stretch the conservative step of Mean Shift method. The convergence speed of Fast Mean Shift is faster than that of Mean Shift. The motivation comes of P. "
Simard [18]. The authors point out that in the case of linear operations (e.g. $f.g$), any invertible linear operation can be applied to f or g if its inverse is applied to the result. For example in the case of convolution, if the derivative operator is applied both to the image and the kernel the result must then be double integrated.

$$f * g - \iint (f' * g') \tag{4.2.5}$$

The authors go on to show that convolution can be significantly accelerated if the derivatives of f and g are sparse (or can be made so). A similar insight is that an invertible linear operation can be applied to f if its inverse is applied to g:

$$(f'') * (\iint g) - f * g \tag{4.2.6}$$

Viewed in this framework computation of the rectangle sum can be expressed as a dot product, $i.x$, where i is the image and x is the rectangle. This operation can be rewritten as

$$i.x - (\iint i).x'' \tag{4.2.7}$$

The double integral of the image (first along rows and then along columns) is in fact the integral image. The second derivative of the rectangle (first in row and then in column) yields four delta functions at the corners of the rectangle.

4.2.3 Fast shift means Calculation

The integral image also called summed area table [18] provides an efficient means to compute area integral for a given image. The integral image at a location *(x,y)* contains the cumulative sum of pixels located to the left and above *(x,y)* including the pixel *(x,y)*:

$$I_{int}(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y') \qquad (4.2.8)$$

$$I_{int}(x, y) = I_{int}(x, y-1) + I_{int}(x-1, y) + I(x, y) - I_{int}(x-1, y-1) \qquad (4.2.9)$$

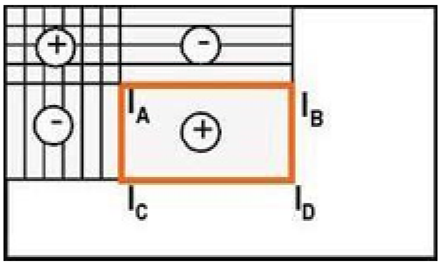


Figure 4-8: Integral Image

where *I(x,y)* is an arbitrary image and $I_{int}(x, y)$ is the integral image i.e. the two-dimensional cumulative distribution function of *I(x,y)*. Fast computation of the integral image can be performed in a single pass [18]. Using the integral image, the area sum of a rectangular region within the original image can be efficiently computed by the following step:

$$S_{area} - I_{int}(x-1,y-1) + I_{int}(x+W-1,y+H-1) - I_{int}(x-1,y+H-1) - I_{int}(x+W-1,y-1) \qquad (4.2.10)$$

$$S_{area} = I_A + I_D \quad I_B \quad I_C \qquad (4.2.11)$$

where S_{area} is the area sum within the rectangle with upper left corner *(x,y)* with *(w,h)* parameters for width and height.

Starting from a location x the local mean shift vector represents an offset to x'_{i} which is a translation towards the nearest mode along the direction of maximum increase in the underlying density function. The local density is estimated within the local neighborhood of kernel by kernel density estimation. Efficient computation of the mean shift vector can be achieved by computing integral images of the first moments:

$$I^{x}_{int} - \sum_{x' \leq x, y' \leq y} x' I(x', y') \qquad (4.2.12)$$

$$I^{y}_{int} - \sum_{x' \leq x, y' \leq y} y' I(x', y') \qquad (4.2.13)$$

The above quantities can be computed in the same pass as the zeroth moment (4.2.8).

Fast computation of new location vectors x' and y' can be performed as:

$$x' = \sum_{\leq x, y \leq y} \frac{K'' x' I(x', y')}{K\, I(x, y)} = \frac{S_{area,int}(I^{x})}{S_{area}(I_{int})} \qquad (4.2.14)$$

$$y' = \sum_{x' \leq x, y' \leq y} \frac{K'' y' I(x', y')}{K\, I(x, y)} = \frac{S_{area,int}(I^{y})}{S_{area}(I_{int})} \qquad (4.2.15)$$

Where K'' represents the second derivative of the kernel K, differentiated with respect to each dimension of the image space i.e. the *x*- and *y-coordinated*. Since the Kernel K is uniform, its second derivative becomes sparse containing only four impulse functions at its corner. Thus, evaluating a convolution takes only the summation of four corner values in the given integral image.

Now using the integral images of the zeroth and the first moments the mean shift vector components can be expressed as:

$$m_x(x,y) = \frac{S_{area}(I^x_{int})}{S_{area}(I_{int})} - x \qquad (4.2.16)$$

$$m_y(x,y) - \frac{S_{area}(I^y_{int})}{S_{area}(I_{int})} - y \qquad (4.2.7)$$

The numerator and denominator of the above expressions denote the area sums computed over the window rectangle *(W(y),H(y))* using the pre computed integral images for zeroth (I_{int}) and first (I^x_{int}, I^y_{int}) moments. Given that a single sum computation based on integral image requires three arithmetic operations and four array accesses, the fast computation of mean shift vector **m**(*x,y*) takes nine arithmetic operations and twelve array accesses only. In addition, the computational complexity is independent of the window size.

4.2.4 Flow Chart Representation

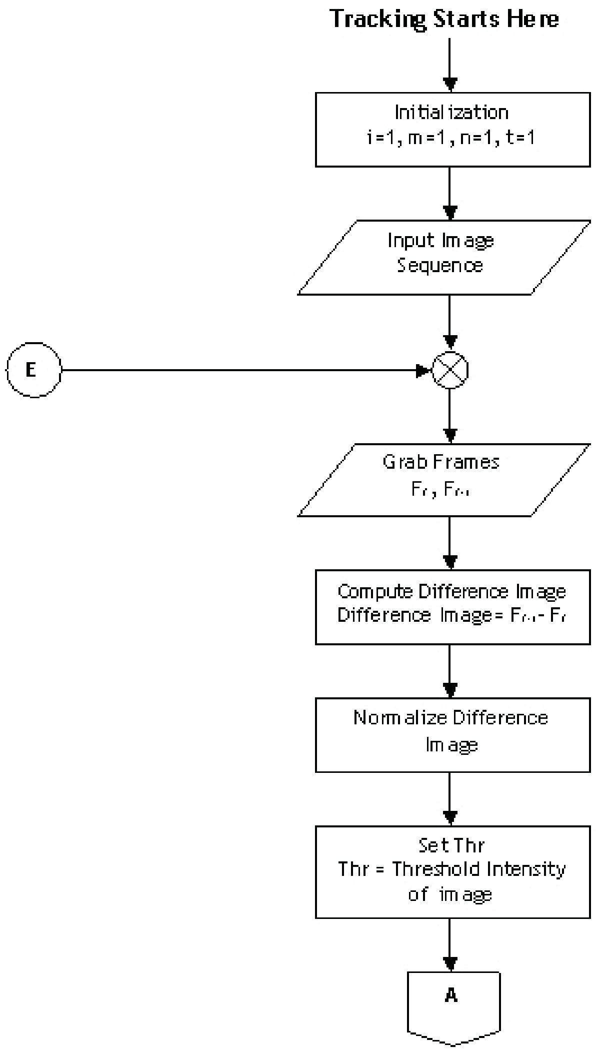


Figure 4-9: Flow Chart representation of Fast Mean Shift Algorithm

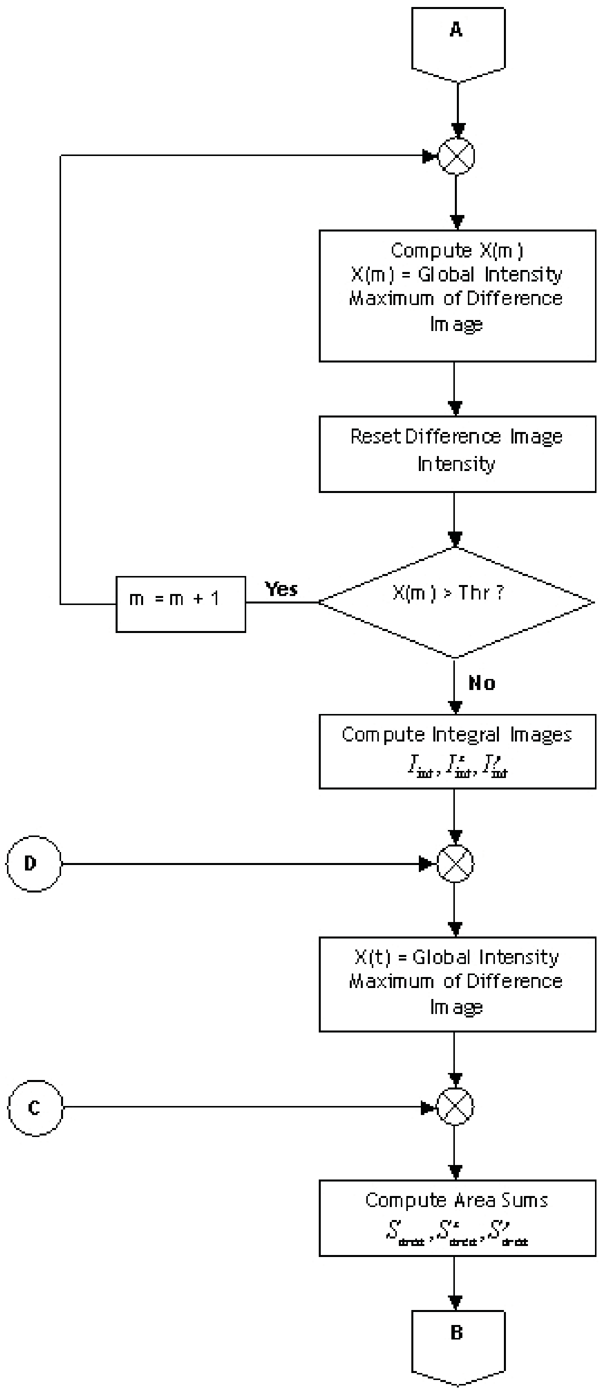

A
Compute X(m)
X(m) = Global Intensity Maximum of Difference Image
Reset Difference Image Intensity
X(m) > Thr ?
Yes
m = m + 1
No
Compute Integral Images
$I_{int}, I^{x}_{int}, I^{y}_{int}$
D
X(t) = Global Intensity Maximum of Difference Image
C
Compute Area Sums
$S_{area}, S^{x}_{area}, S^{y}_{area}$
B

Figure 4-10 : Flow Chart representation of Fast Mean Shift Algorithm (cont-I)

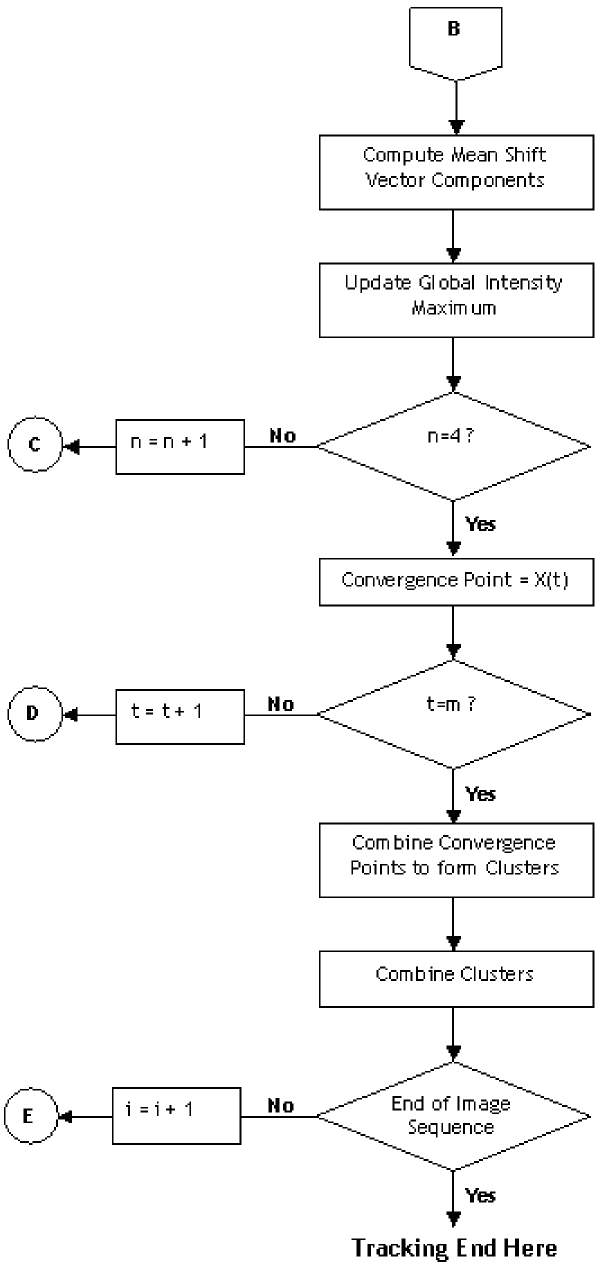


Figure 4-11: Flow Chart representation of Fast Mean Shift Algorithm (cont-II)

4.2.5 <u>Experimental Results</u>

Tracking Results of fast mean shift algorithm for two video sequences are shown below in Figure 4-12 and Figure 4-13. Tracked object is illustrated by a rectangle. Stable results are obtained using fast mean shift algorithm, which shows excellent mode sensitivity of the method even in the case of change of size of image.

Tracking Results with Video Sequence 1

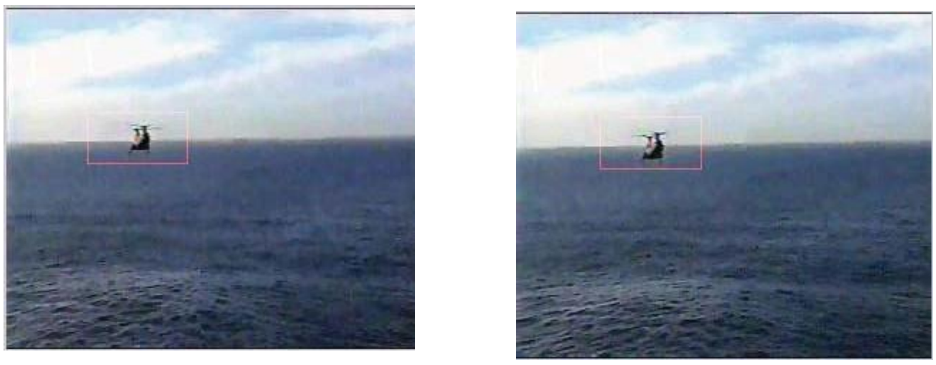


Figure 4-12: Results of Fast Mean Shift based tracking in Frame number 45 and 56

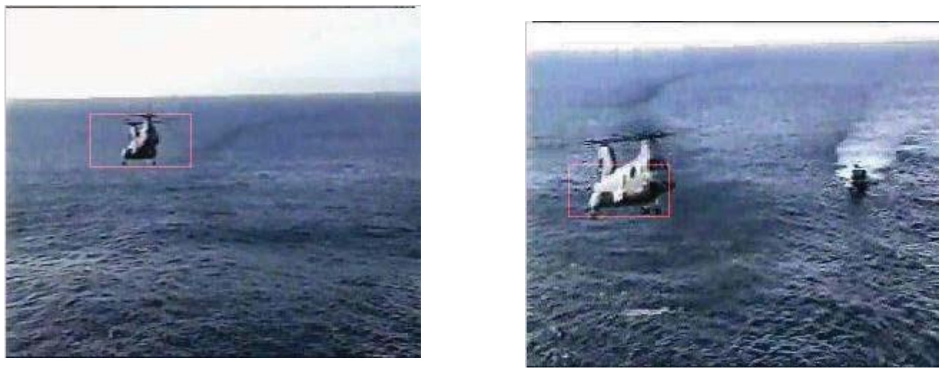


Figure 4-13 : Results of Fast Mean Shift based tracking in Frame number 67 and 113

Tracking Results with Video Sequence 2

These are the results of Tracking with Fast Mean Shift technique using video sequence 2.

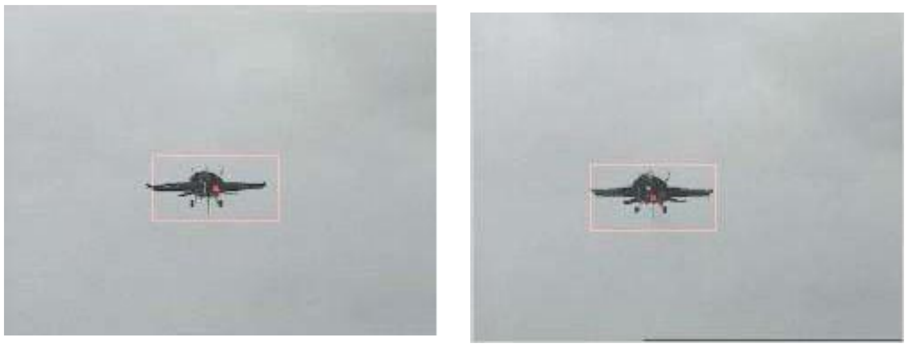


Figure 4-14: Results of Fast Mean Shift based tracking in frame number 13 and 25

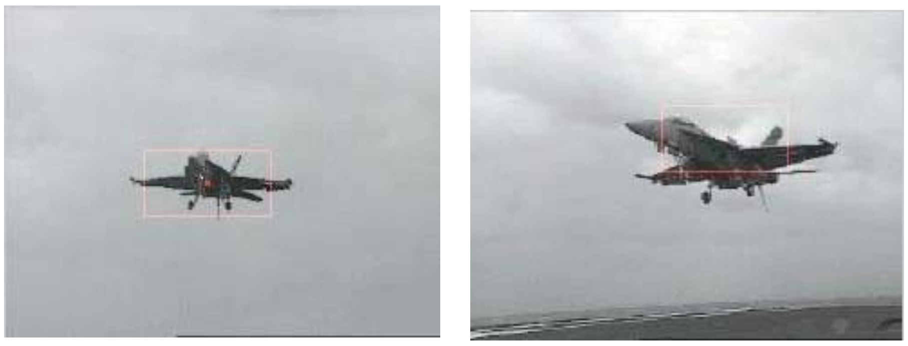
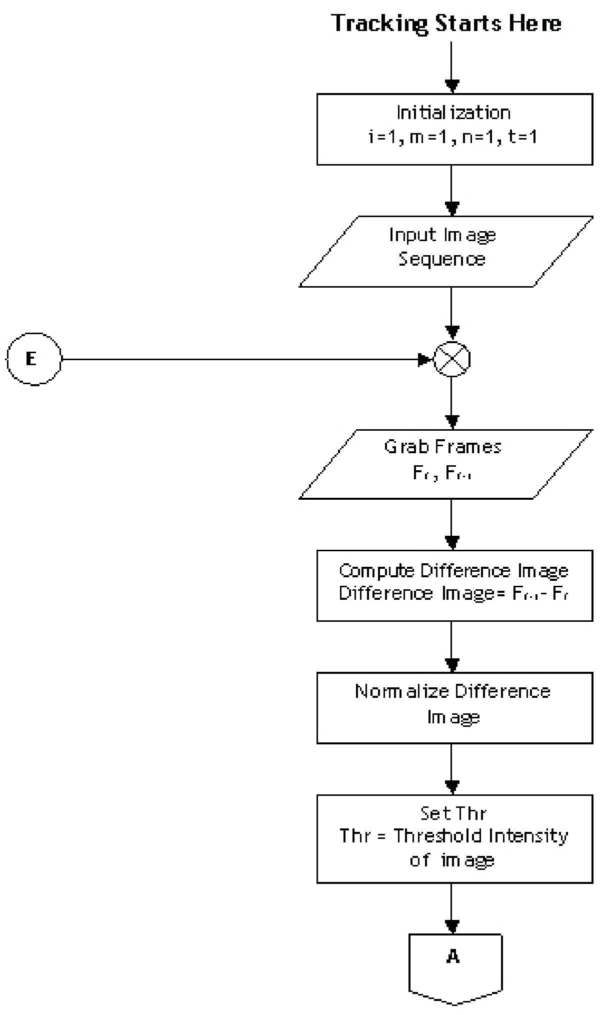


Figure 4-15: Results of Fast Mean Shift based tracking in frame number 69 and 94

4.2.6 Flow Chart Representation of Fast Mean shift



Tracking Starts Here
Initialization i=1, m=1, n=1, t=1
Input Image Sequence
E
Grab Frames F_t, F_{t-1}
Compute Difference Image
Difference Image= F_{t-1}- F_t
Normalize Difference Image
Set Thr
Thr = Threshold Intensity of image
A

Figure 4-16: Flow Chart representation of Fast Mean Shift Algorithm

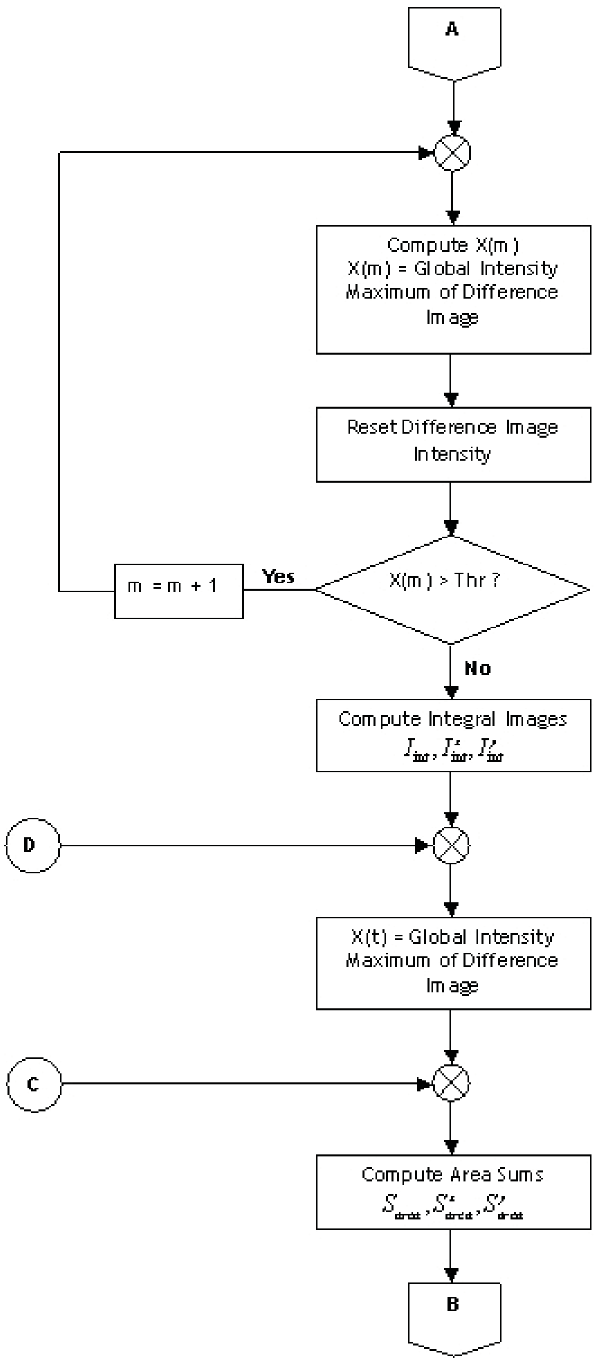


Figure 4-17: Flow Chart representation of Fast Mean Shift Algorithm (cont-I)

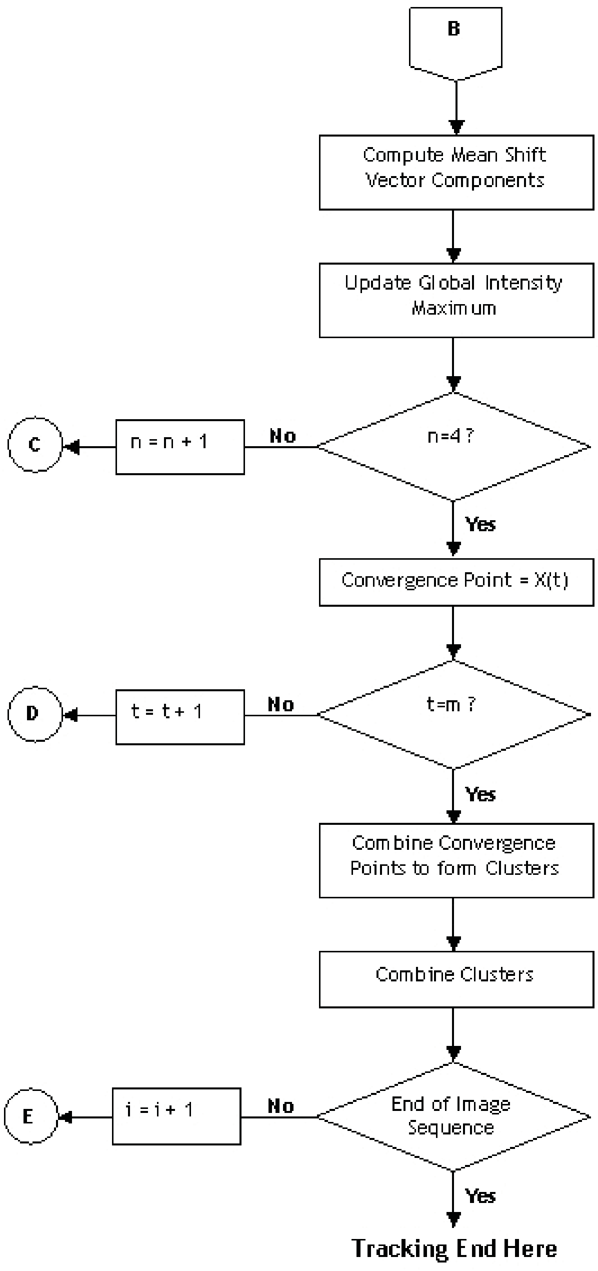


Figure 4-18: Flow Chart representation of Fast Mean Shift Algorithm (cont-II)

4.2.7 Advantages

Mean shift offers following advantages

4.2.7.1 It is application sovereign tool.

4.2.7.2 Mean shift is also appropriate for real data analysis.

4.2.7.3 It does not presume any former shape on data clusters.

4.2.7.4 It can easily handle random feature spaces.

4.2.8 Limitations

Though mean shift algorithm is an efficient approach to track objects but also it has some limitations.

4.2.8.1 The window sizing is not trivial.

4.2.8.2 Inappropriate window size can cause modes to be amalgamated or generate some additional shallow modes.

4.2.9 Conclusion

The Fast Mean Shift is no doubt an encroachment in methodology of Mean shift technique .The mathematical variations in the sequence of derivatives and integrals makes the algorithm robust and reduces the time aspect .Fast Mean shift algorithm with all its complex mathematics with stands the occlusion and target loss problem concretely .Whereas, complete disappearance of target from the region of interest may lead to the use of template support so as to retain the robustness of this system.

Chapter 5

5. Kalman Filter

The Kalman filter provides a recursive solution to the linear optimal filtering problem. It applies to motionless as well as active environments and can be used for prediction of next state/states, optimal state estimation, and noise filtering problems and data fusion. . Kalman filter is basically a deposit of mathematical equations that puts into practice a predictor-corrector type estimator that is optimal in the sense that it minimizes the estimated error covariance as when some presumed conditions are met. The solution provided by the Kalman filter is recursive that each updated estimate of the state is computed from the previous estimate and the new input data, so only the previous estimate requires storage. In addition to eliminating the need for storing the entire past observed data, the Kalman filter is computationally more efficient than computing the estimate directly from the entire past observed data at each step of the filtering process.

5.1 Why Kalman Filter?

Although, Kalman filter is computationally somewhat more multifarious than the other filters still there are reasons to use it. As far as the computational complexity is concerned, recent advancements in computer technology has nullified this drawback of the Kalman filter. Kalman has certain distinct features that are not provided by any other tracking filter. Some of these features are listed below.

5.1.1 Prediction Accuracy

In the process of computing filter weights, calculations for the accuracy of Kalman filter prediction are made [20]. The priori covariance matrix of the Kalman filter provides this information. This prediction information is needed in weapon delivery system. If predicted position of the target is known accurately then it is enough to kill the target [13]. It is also needed to accurately predict where a SCUD or any other ballistic missile would land. It is also used to determine that from where artillery shell was launched and then same information is used to destroy the attacker as well.

5.1.2 Optimal Filtering

Kalman filter make optimal use of the target measurements by adjusting the filter weights to take into account the accuracy of the *nth* measurement [20]. If the target measurement was more accurate, then weights will be automatically adjusted in such a way that more weight will be given to measurement than prediction.

5.1.3 Priori Information

Kalman filter optimally make use of prior information [20]. This is especially useful when using two separate devices for searching and tracking. Data from the searching devices can be optimally used to initialize the filter weights that will result in small transient in tracking filter.

5.1.4 Target Dynamics

Kalman filter is model-based predictor. It uses target dynamic model to predict the next state of the target. Target dynamic model allows direct filter update rate [20]. It is also possible to use more than one dynamic model in the filter. This helps to track the more complex maneuvers made by the target. Now days, it has been widely accepted that accurate targeting tracking requires multiple target models. Interacting Multiple Model (IMM) has become a generally accepted best method for multiple models filtering [21]. By probabilistically combining predictions of these models (typically by Kalman), a best guesstimate of target state can be made.

5.2 Models for Kalman Filter

There are two types of tracking models for Kalman filter

- Constant velocity model
- Constant Acceleration model

5.2.1 Constant velocity model

In constant velocity model, velocity of the target is assumed to be constant between successive samples and not the whole length of time. This model is described by the following equations [20].

$$x_{n1} \quad x_n + T \quad \dot{x} \qquad (5.1)$$

$$x_{n+1} = \dot{x}_n \qquad (5.2)$$

where T is sampling time. Accuracy of this model improves with decrease in sampling time. Therefore, smaller the sampling time better will be results. This model is especially suitable for slow moving objects. It is also possible to model a fast charging target with these dynamics if sampling time is appreciably small.

In state space, this model is represented by pure inertia model. Since there is no inpu
for the dynamics, so it is given by the following state space model.

$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{bmatrix} x \\ v \end{bmatrix} \tag{5.3}$$

From this model, state matrix is obtained.

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

The state transition matrix for this model can be found using the formulae [18].

$$\phi(t + T) = I + TA + \frac{T^2}{2!}A^2 + \frac{T^3}{3!}A^3 + \ldots \tag{5.4}$$

Using above formulae, state transition matrix is obtained is:

$$\phi\,(t \quad T) = \begin{pmatrix} 1 & T \\ 0 & 1 \end{pmatrix} \tag{5.5}$$

This can be discretezed to following state space model.

$$\begin{bmatrix} x_{n-1} \\ v_{n-1} \end{bmatrix} \begin{pmatrix} 1 & T \\ 0 & 1 \end{pmatrix} \begin{bmatrix} x_n \\ v_n \end{bmatrix} \tag{5.6}$$

This model is known as Constant Velocity Model. Since, there is no input that can influence the target motion, input coupling matrix is not present in this model. Only possible input for this model can be a random noise. This random noise is introduced in the velocity variable of the target. Model that incorporates this random component in velocity variable is known as “Constant Velocity Model with Random Walk.”

With the help of the above discretized state space model, it is possible to generate the state of the target after T units of time. Trajectories shown Fig 5-1 was generated using the constant velocity model for the target s range and range rate.

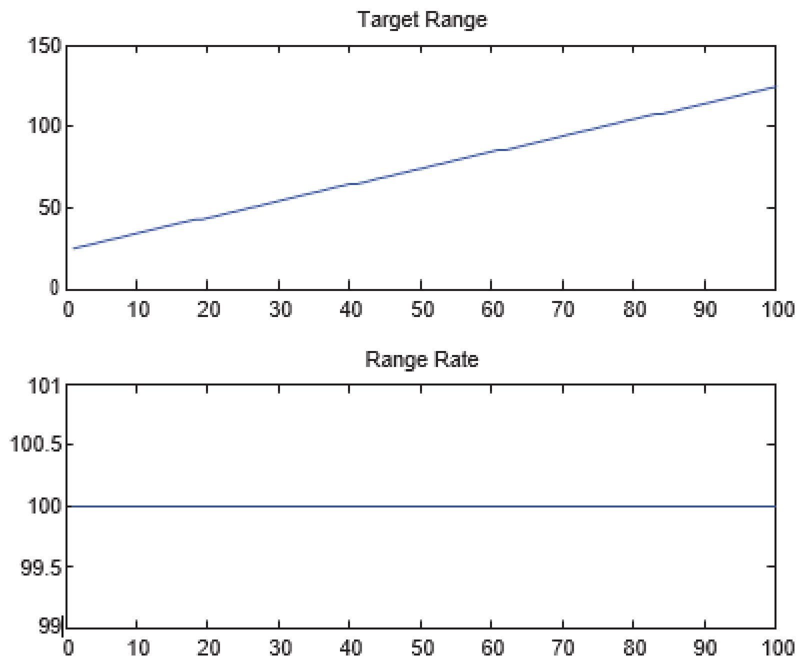


Figure 5-1Constant Velocity Model

From Fig 5-1, it can be observed that range of the target is linearly increasing with time but the velocity of’the target does not change and remain constant at a fixed value. This model of the target range is applicable in many situations e.g. a emissary aero-plane on surveillance or an underwater mole vehicle will follow a linear range model. Constant velocity assumption is somewhat more simplified. This model can be improved by introducing a random change in the velocity of the target. This model is known as constant velocity model with a random walk . Trajectories shown in Fig 5-2 were generated using constant velocity with random walk model for the target s range and range rate. A random variable was generated between 0.5 and +0.5 using MATLAB. This random value was added to the target velocity to introduce a random change in target s range rate or velocity.

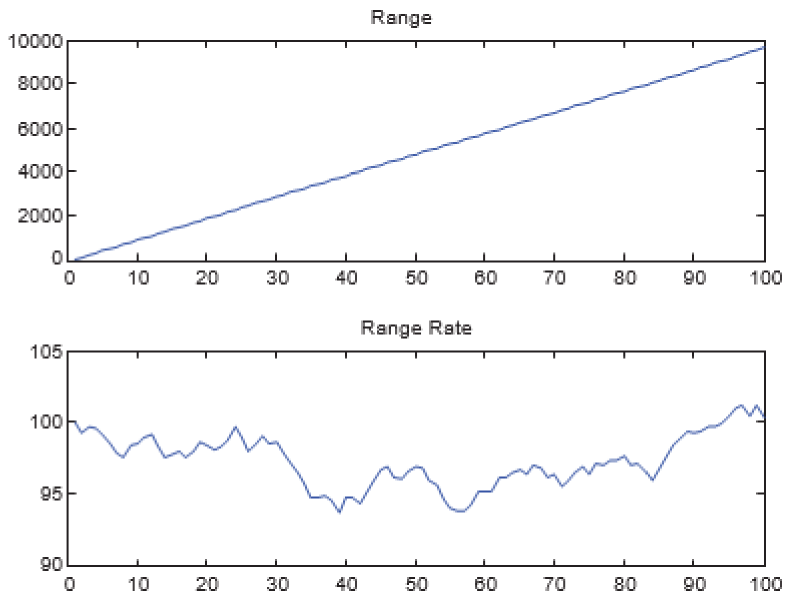


Figure 5-2 Random Walk Model

Again, it can be seen that still the range of the target is changing linearly but now the velocity or range rate of this is not constant. Range rate is having random variation between 100 and 95. This model is more realistic than the previous one as velocity of an object seldom remains constant. There is always a fluctuation in velocity. This random walk model was used, most of the time, to generate the data for the target azimuth and elevation angles.

5.2.2 <u>Constant Acceleration Model</u>

This model assumes the acceleration of the target to be constant during the sampling time only and not the entire duration of time. Velocity of the target is taken to be variable. This model is good for accelerating bodies. This is a third order model that introduces one additional variable.

This model can be represented by the following equations.

$$\begin{vmatrix} \dot{x} \\ \dot{v} \\ \dot{a} \end{vmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{bmatrix} x \\ v \\ a \end{bmatrix} \qquad (5.7)$$

where x, v and a are the position, velocity and acceleration of the target. The state matrix A obtained from this model is:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Thus, the state transition matrix can be obtained using formulae [21].

$$\phi(t+T) = I + TA + \frac{T^2}{2!}A^2 + \frac{T^3}{3!}A^3 + ... \qquad (5.8)$$

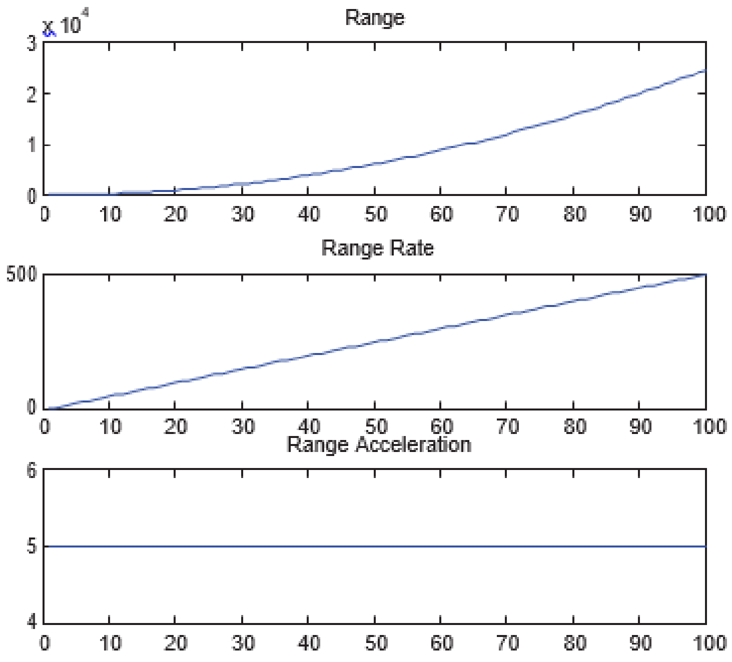
State transition matrix thus obtained is:

$$\phi(t+T) = \begin{pmatrix} 1 & T & \frac{T^2}{2} \\ 0 & 1 & T \\ 0 & 0 & 1 \end{pmatrix} \qquad (5.9)$$

Using this state transition matrix, this model is discredited into following state space model [23].

$$\begin{bmatrix} x_{n+1} \\ v_{n+1} \\ a_{n+1} \end{bmatrix} = \begin{pmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x_n \\ v_n \\ a_n \end{bmatrix} \qquad (5.10)$$

This model was used generate the trajectories for the target range, range rate and range acceleration show in Fig 5-2.



It can be observed from the Fig 5-3 that range of target is curving up. This parabolic trend in the range is due to the fact that target is moving with an acceleration. It is also apparent that velocity of the target is linearly increasing with time. This linear behavior velocity shows that target is moving with a constant acceleration. Finally, in the graph for acceleration of the body, a straight line is observed. This is obvious that straight line is also a sign of constant acceleration. In real world, it is almost impossible for a target to move with a constant acceleration.

A more realistic model would be to use a random acceleration, rather than a constant one. This model in which acceleration was not held constant; instead, a random variable was added to the acceleration component is called constant acceleration model with a random walk . Trajeçtories shown in Fig 5-4 have been produce using the random wall model. A 10% random variation was introduced into the acceleration variable while velocity variable was kept intact.

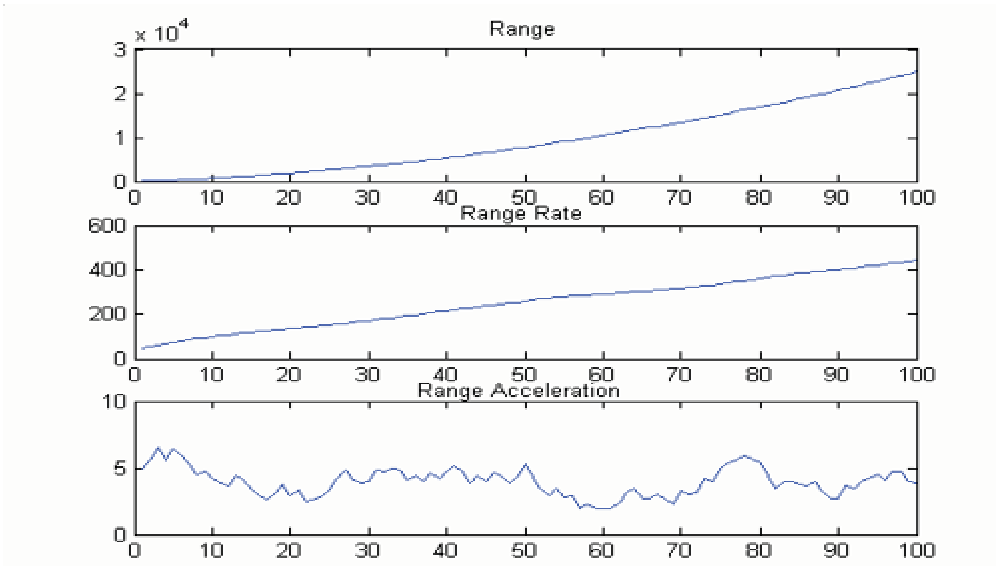


Figure 5-4 Constant Acceleration Model with Random Walk

From Fig 5-4, again a parabolic trend is observed in the range. This is due to the acceleration of the target. However, range rate now is not as linear as before, though trend is still increasing. Major deviation from the previous model is in range acceleration. Rather than being constant, now it is fluctuating about a constant value.

It is also possible to randomly introduce variation both in velocity and acceleration simultaneously. Fig 5-4 shows the scenario where 10% variation in acceleration and 20% random change was incorporated.

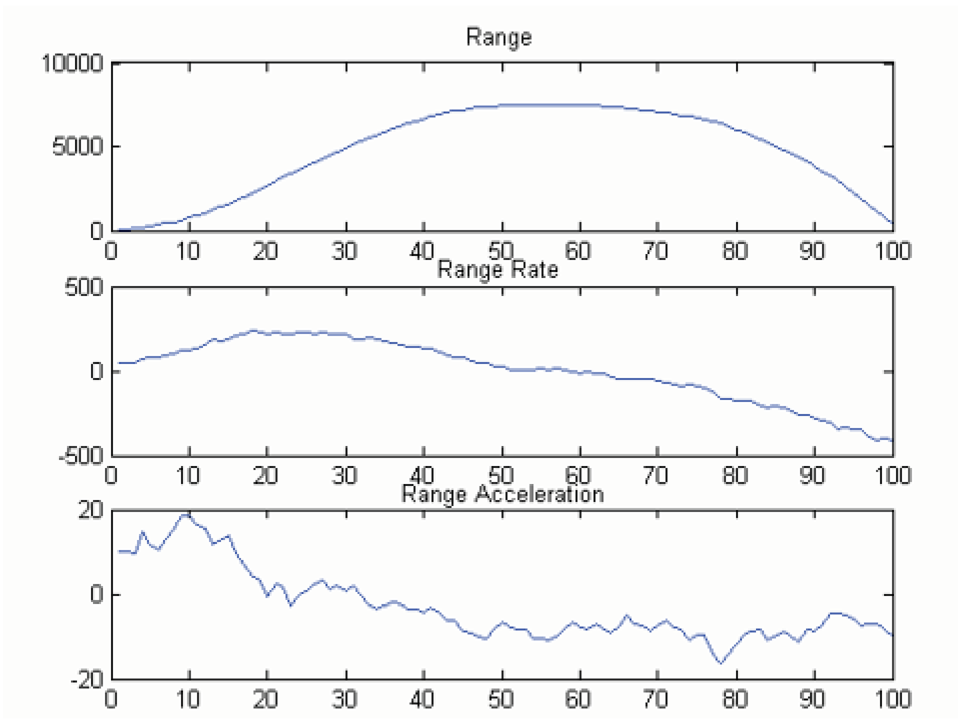
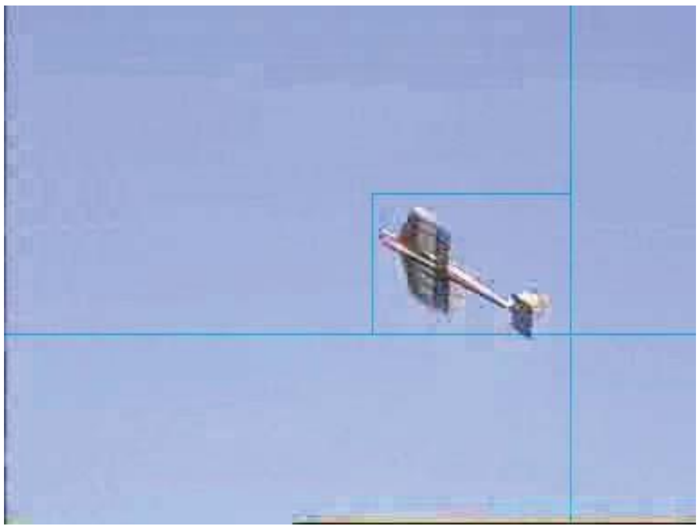


Figure 5-5 Random Velocity Random Acceleration Model

In Fig 5-5, negative values of velocity and acceleration can be seen. This represents the case when the target is moving towards the sensor. In that case, velocity and acceleration of the target will have negative values. This is obvious from the range. It can be seen that first, range is increasing when velocity and acceleration were positive. When velocity became negative, then target range decreases which shows the target s motion towards the sensor. It also indicates a circular motion of the target.

5.3 Experimental Results

The experimental results shown below in figure 5-6 onwards prove the accuracy of the algorithm followed and implemented in Matlab 7.0.



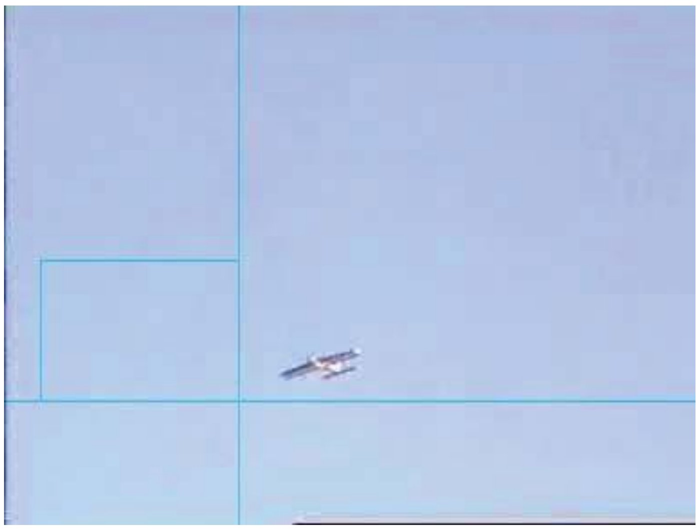
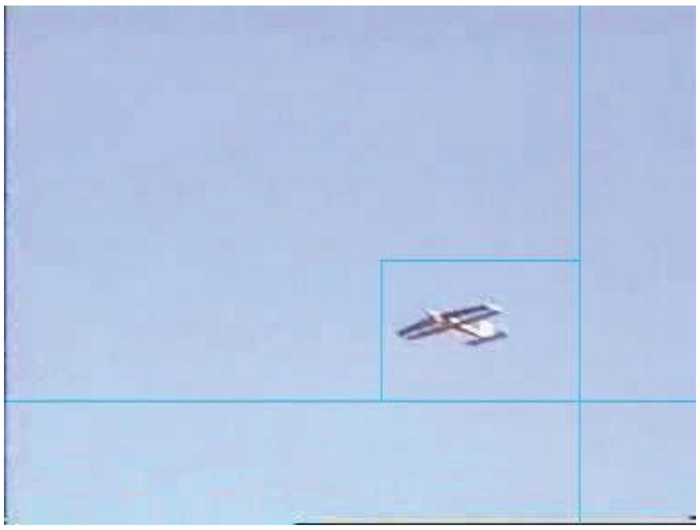


Figure 5-6 Tracking Results using Kalman Filter Showing frame number 10 and 20

Here in figure 5-7 the frames shown depict very precise tracking in which object is completely engrossed by the tracker.



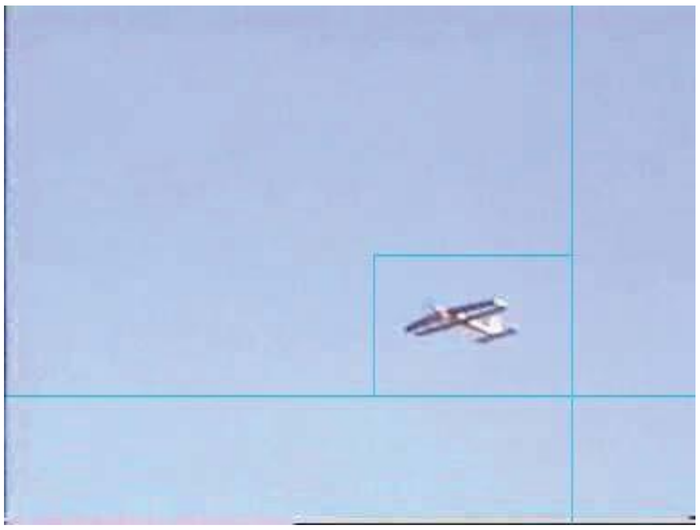
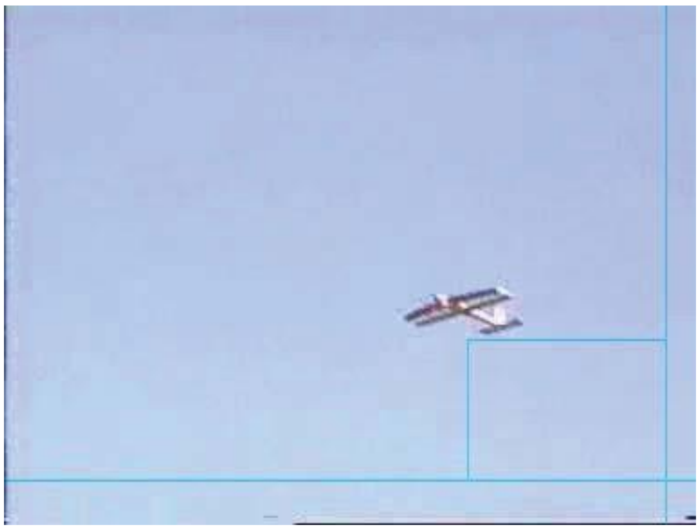


Figure 5-7 Tracking Results using Kalman Filter Showing frame number 42 and 43

Similarly in figure 5-8 the tracking sequence is shown where as in frame number 45 its just about to be grabbed by the tracker.



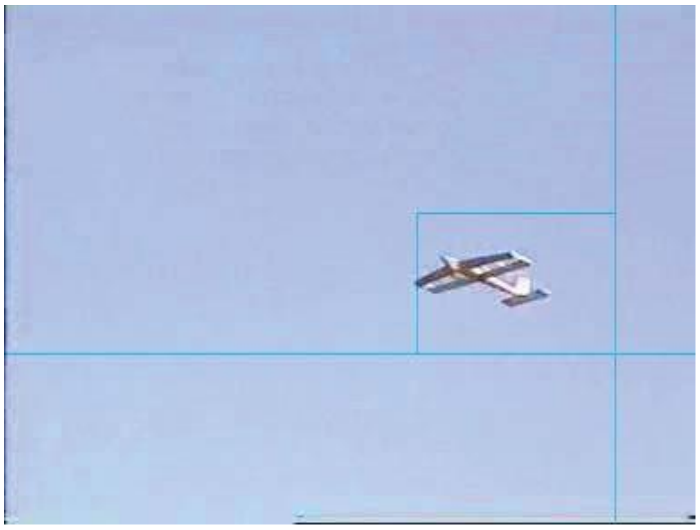
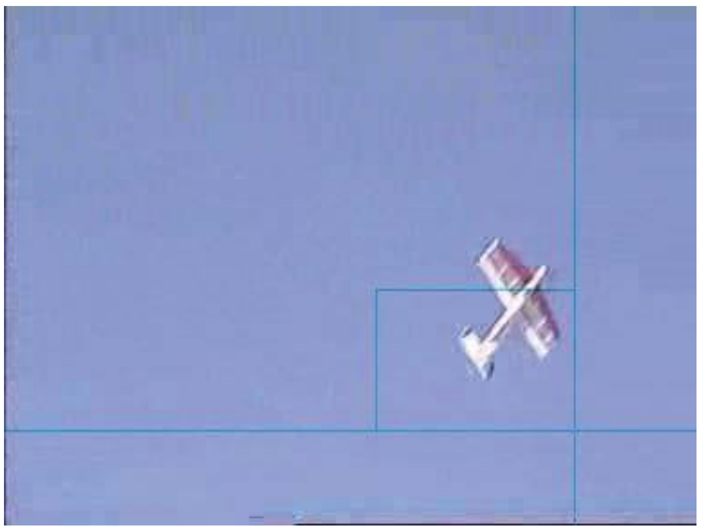


Figure 5-8 Tracking Results using Kalman Filter Showing frame number 45 and 52

Figure 5-9 shows the orientation changes of the air borne object used and Kalman filter s ability to deal with such sudden directional changes.



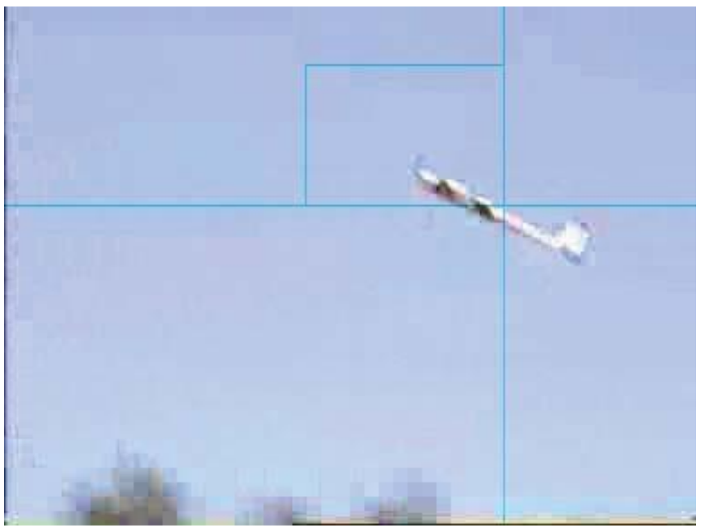


Figure 5-9 Orientation changes and Kalman Filter s'Tracking ability shown in frame number 143 and 150

Such orientation changes as shown in figure 5-10 are a little difficult to handle by the tracking techniques as we are not dealing with such maneuvers in this very project so tracking algorithm works well.

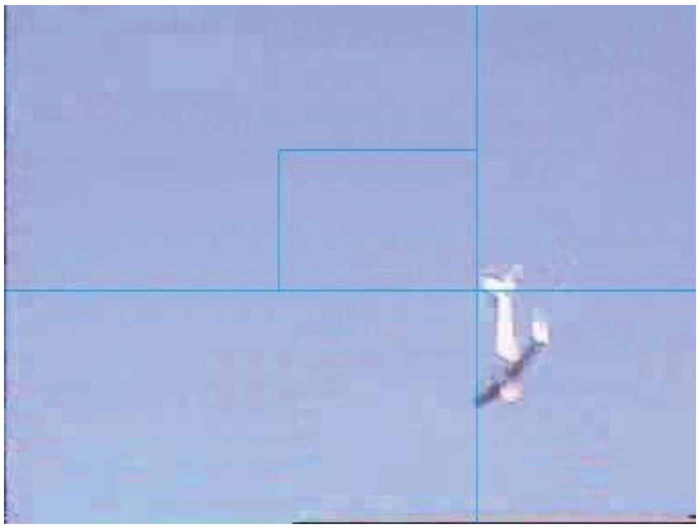


Figure 5-10 Orientation changes in frame number 179

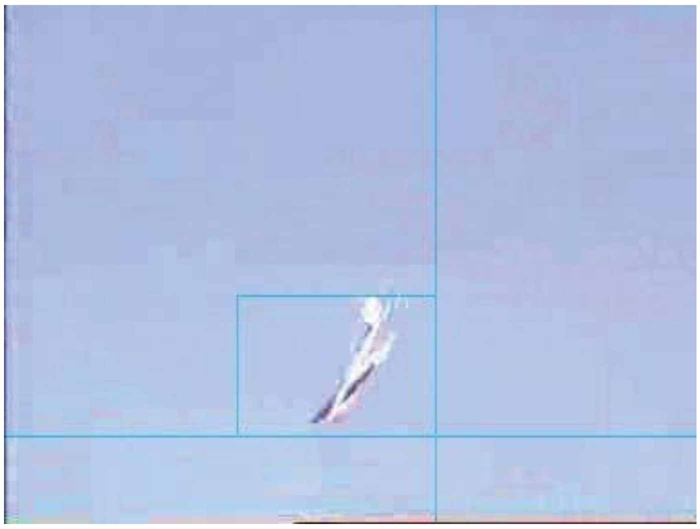


Figure 5-11 Tracking Results using Kalman Filter Showing frame number 155

Thus these experimental results make it clear that Kalman Filter is predictive in nature and predicts the next move of the object of interest even if the object suffers obstacles. This property makes Kalman Filter worth implementation.

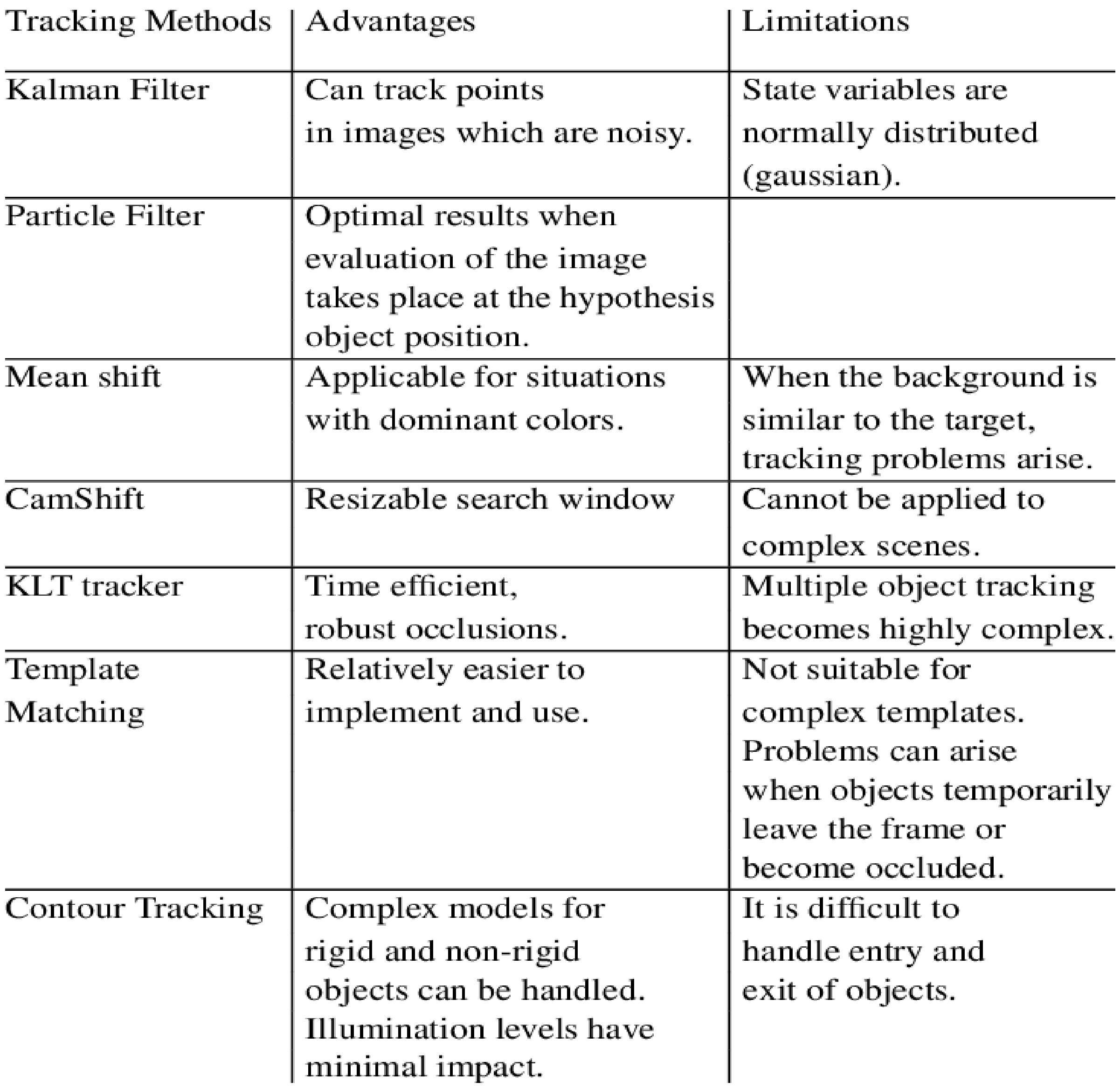
5.4 Comparison between Techniques Implemented

Moving object tracking is one of the most fertile areas in research sector. Techniques available so far are competent still new methods are searched for to enhance the features, reduce time, and improve efficiency and augment precision. Tracking is widely used in many industries now a days but most of its usage is in defense and surveillance sectors .Both of these fields are highly sensitive and receptive and require high accuracy .For improving these features new methods are developed and their competence level is compared .

The techniques used in this very project are described in detail in chapter 4 and 5. All the techniques used show appropriate results which are also shown in experimental results session in each chapter. Now a detailed comparison of features and results is presented as under in order to make the difference clearer.

The techniques are compared below in Table 5-1 showing their major differences. The methods differ from each other in mathematical background and implementation method .Kalman Filter is a set of equations which are known as predictor corrector equations where as fast mean shift is a convolution between the double derivative of kernel and double integral of image .Where as template matching has a template in .jpeg format which is an image of object in question and used for comparison with the very first frame and later on every previous frame serves as template for the next one. This makes the template matching technique adaptive and also drives it closer to occlusion problem.

Fast Mean shift is a faster and better technique, far more robust than template matching .In spite of all these differences all these methods are still prevalent and used in many fields. The comprehensive difference is listed below in Table 5-1. So, there were few differences mentioned above in table 5-1 .All these techniques in their place serve the purpose of moving object tracking very appropriately. Several short comings of template matching are removed by mean shift and similarly to attain perfection and accuracy Kalman filter is implemented next .Use of Kalman filter ensures the predictive behavior which serves as an important feature for tracking field. As in case of any kind of collision of objects or distraction of objects or any sudden change in directional attributes Predictive behavior saves the technique from being failed.



Tracking Methods	Advantages	Limitations
Kalman Filter	Can track points in images which are noisy.	State variables are normally distributed (gaussian).
Particle Filter	Optimal results when evaluation of the image takes place at the hypothesis object position.	
Mean shift	Applicable for situations with dominant colors.	When the background is similar to the target, tracking problems arise.
CamShift	Resizable search window	Cannot be applied to complex scenes.
KLT tracker	Time efficient, robust occlusions.	Multiple object tracking becomes highly complex.
Template Matching	Relatively easier to implement and use.	Not suitable for complex templates. Problems can arise when objects temporarily leave the frame or become occluded.
Contour Tracking	Complex models for rigid and non-rigid objects can be handled. Illumination levels have minimal impact.	It is difficult to handle entry and exit of objects.

Table 5.1 Difference between Implemented techniques

5.5 Conclusion

Kalman filter is correctly called a predictor and corrector as it predicts the next moves of the object in question and that is why it is also used to identify the flood ̓s course which is highly unexpected phenomena. The comparison described also represents the accuracy level of the techniques implemented and their differences as well.

Chapter 6

6. Other Methods for Object Detection and Tracking

6.1 Texture Analysis

Major goals of texture research in computer vision are to understand, model and process texture, and ultimately to simulate human visual learning process using computer technologies.

A typical computer vision system can be divided into components such as the ones show in Fig 4.7. Texture analysis might be applied to various stages of the process. At the preprocessing stage, images could be segmented into contiguous regions based on texture properties of each region; At the feature extraction and the classification stages, texture features could provide cues for classifying patterns or identifying objects.

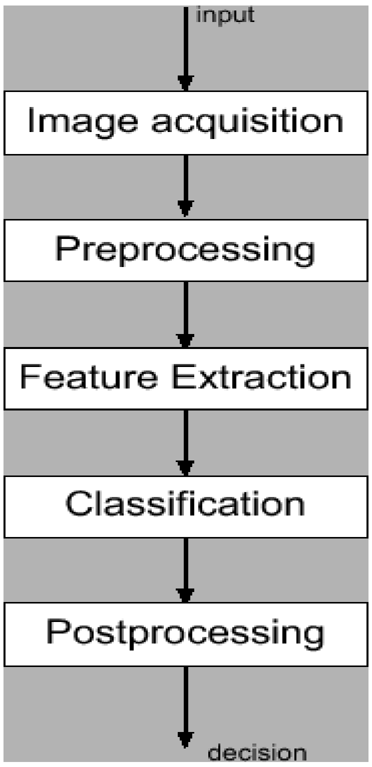


Figure 6-1: The components of a typical computer vision system.

As a fundamental basis for all other texture-related applications, *texture analysis* seeks to derive a general, efficient and compact quantitative description of textures so that various mathematical operations can be used to alter, compare and transform textures. Most available texture analysis algorithms involve extracting texture features and deriving an image coding scheme for presenting selected features. These algorithms might differ in either which texture features are extracted or how they are presented in the description. For example, a statistical approach describes a texture via image signal statistics which reflect nondeterministic properties of spatial distribution of image signals. A spectral method extracts texture features from the spectral domain . A structural approach considers a texture as a hierarchy of spatial arrangements of well-defined texture primitives. A probability model describes the underlying stochastic process that generates textures. Several representative works on texture analysis.

Four major application domains related to texture analysis are texture classification, texture segmentation, shape from texture, and texture synthesis . Below, each domain is described briefly.

6.1.1 Texture Classification

Texture classification assigns a given texture to some texture classes [Two main classification methods are *supervised* and *unsupervised* classification.

Supervised classification is provided examples of each texture class as a training set. A supervised *classifier* is trained using the set to learn a characterisation for each texture class. Unsupervised classification does not require prior knowledge, which is able to automatically discover different classes from input textures. Another class is *semi-supervised* with only partial prior knowledge being available.

The majority of classification methods involve a two-stage process. The first stage is feature extraction, which yields a characterisation of each texture class in terms of feature measures. It is important to identify and select distinguishing features that are invariant to irrelevant transformation of the image, such as translation, rotation, and scaling. Ideally, the quantitative measures of selected features should be very close for similar textures. However, it is a difficult problem to design a universally applicable feature extractor, and most present ones are problem dependent and require more or less domain knowledge.

The second stage is classification, in which classifiers are trained to determine the classification for each input texture based on obtained measures of selected features. In this case, a classifier is a function which takes the selected features as inputs and texture classes as outputs.

In the case of supervised classification, a k-nearest neighbour (k-NN) classifier is usually applied ,which determines the classification of a texture by computing distances to the k nearest training cases. The distances are computed in a multi-dimensional *feature space* constructed by selected texture features. Euclidean distance, Chi-square distance, and Kullback-Leibler distance are mostly used as distance metrics for distributions and thus similarity metrics for textures. A Bayesian classifier that performs classification via probabilistic inference is also frequently used. A general two-class Bayesian classifier can be specified by the following Bayes formula ,

$$Pr(w_j|\mathbf{F}) = \frac{Pr(\mathbf{F}|w_j)Pr(w_j)}{Pr(\mathbf{F})}$$

Here, wj;j =1,2 denote two categories; the prior Pr(wj) is the unconditional probability of the category w; the prior P(f|w) is called the likelihood of w with respect to a set of feature measures $\mathbf{F}$. The formula leads to a posterior P(w|f) that gives the probability of a category w given the feature measures $\mathbf{F}$. Essentially, this Bayesian classifier converts the prior knowledge, i.e. P(w) and P(F|w), into a posterior P(w|f) that describes the probability of classifying a texture into a particular class w given the evidence (observed features $\mathbf{F}$). For instance, based on the classifier, a texture could be assigned to a particular category, if the posterior probability is greater than some threshold . A Bayesian classifier needs prior knowledge about textures and is based on a probability model, while a $k-NN$ classifier makes no assumption on textures and is a non-parametric approach.

Leung and Malik developed a state-of-the-art feature-based method for classifying 3D textures under varying viewpoint and illumination. In feature extraction, the method applies a filter bank onto the training textures for each material with known viewpoints and illumination. A *k*-mean clustering algorithm is exploited to identify *k* clusters from the vector space concatenating all filter responses. Cluster centres are the representative *textons* of each material and act as feature descriptors. The textons of all materials together create a global texton dictionary, so that each material is represented by a particular probability density function,i.e. the distribution of texton frequencies, with respect to the dictionary. For a novel texture to be classified, the distribution of texton frequencies with respect to the texton dictionary is computed, for a $k-NN$ classifier to assign the texture to a class with the nearest distribution of texton frequencies. Figure 4.3.1 shows examples of textures and their corresponding texton distribution with respect to a texton dictionary .

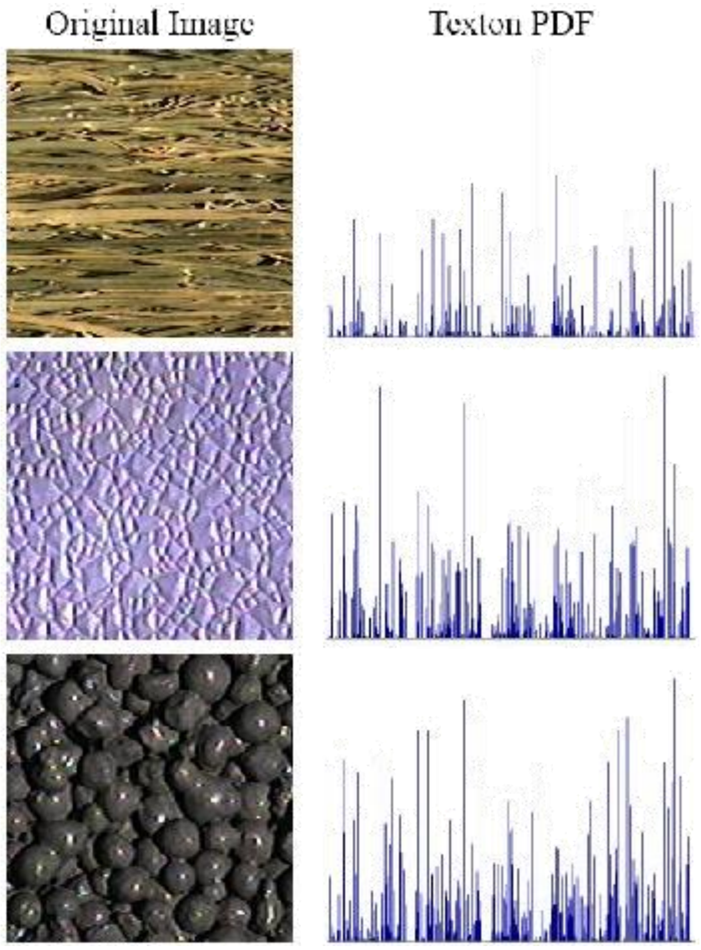
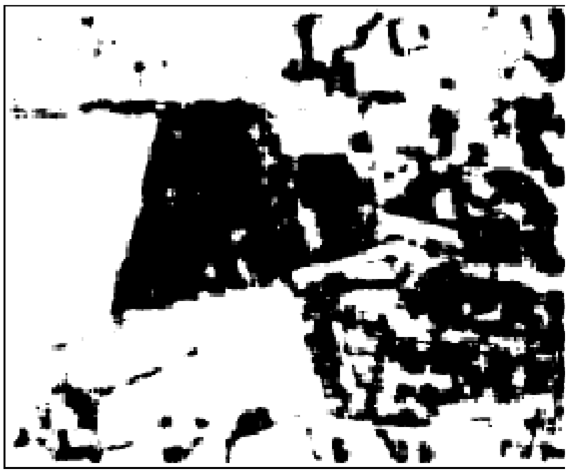
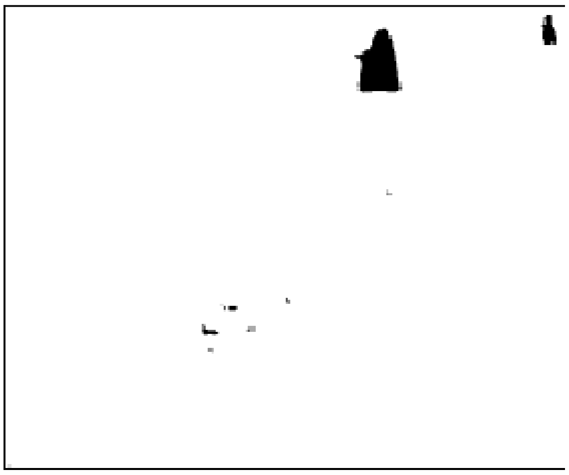


Figure 6-2: Textures and their corresponding texton distributions .

Texture classification can sort image data into more readily interpretable information, which is used in a wide range of applications such as industrial inspection, image retrieval , medical imaging and remote sensing .

(a) (b)

(c) (d)

Figure 6-3: Segmentation of an aerial image by texture properties . (a) input aerial photo; (b) region map:`field'; (c) region map: `residential area'; (d) region map: `vegetation area'.

6.1.2 <u>Texture Segmentation</u>

Texture segmentation partitions an image into a set of disjoint regions based on texture properties, so that each region is homogeneous with respect to certain texture characteristics. Results of segmentation can be applied to further image processing and analysis, for instance, to object recognition. Similar to classification, segmentation of texture also involves extracting features and deriving metrics to segregate textures. However, segmentation is generally more difficult than classification, since boundaries that separate different texture regions have to be detected in addition to recognising texture in each region.

Texture segmentation could also be supervised or unsupervised depending on if prior knowledge about the image or texture class is available. Supervised texture segmentation identifies and separates one or more regions that match texture properties shown in the training textures. Unsupervised segmentation has to first recover different texture classes from an image before separating them into regions. Compared to the supervised case, the unsupervised segmentation is more flexible for real world applications despite that it is generally more computationally expensive.

Partitioning an image into homogeneous regions is very useful in a variety of applications of pattern recognition and machine leaning. For example, in remote sensing and GIS analysis, texture segmentation could be applied to detect landscape change from an aerial photo. Figure <u>4.9</u> illustrates such an application of texture segmentation which finds different ground objects, such as rural, industrial residential areas, based on their distinct texture properties appeared in the image

6.1.3 <u>Shape from Texture</u>

Shape from texture is the problem of estimating a 3D surface shape by analysing texture property of a 2D image. Weak homogeneity or isotropy of a texture is likely to provide a shape cue . For instance, texture gradient is usually resulted from perspective projection when the surface is viewed from a slant, which infers the parameters of surface shape or the underlying perspective transformation. Therefore, via a proper measure of texture gradient, a depth map and the object shape could be recovered.

Shape from texture have been used for recovering true surface orientation, reconstructing surface shape, and inferring the 3D layout of objects, in many applications . For example, the plane vanish line could be computed from texture deformation in an image , which could be used to affine rectify the image.

6.1.4 <u>Texture Synthesis</u>

In computer graphics, texture synthesis is a common technique to create large textures from usually small texture samples, for the use of texture mapping in surface or scene rendering applications. A synthetic texture should differ from the samples, but should have perceptually identical texture characteristics . The main advantage of texture synthesis in this case is that it can naturally handle boundary condition and avoid verbatim repetitions. In computer vision, texture synthesis is of interest also because it provides an empirical way to test texture analysis. Because a synthesis algorithm is usually based on texture analysis, the result justifies effectiveness of the underlying models. Compared to texture classification and segmentation, texture synthesis poses a bigger challenge on texture analysis because it requires a more detailed texture description and also reproducing textures is generally more difficult than discriminating them.

Other applications of texture synthesis include image editing , image completion , and video synthesis etc.

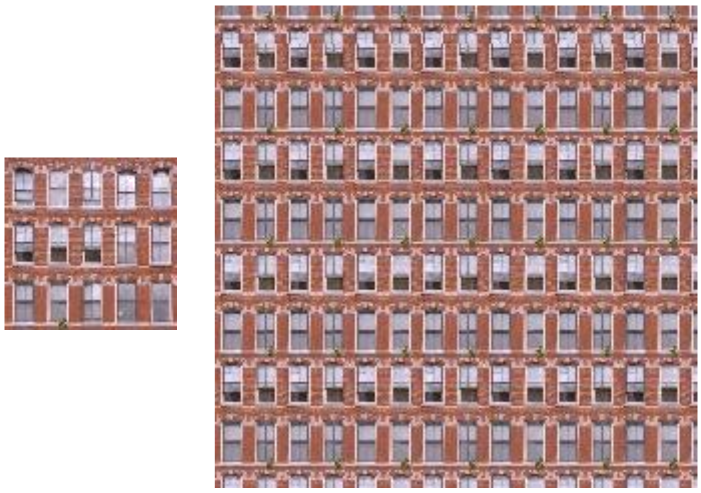

Sample Image Synthetic texture

Figure 6-4: Texture synthesis by example: Given a sample texture, to generate a new image having the same visual textural appearance. The synthetic texture is generated using the method described in this thesis.

6.2 Focusing Analysis

The method discussed here[3] operates in the Fourier transform domain using the exact form of the double-square-root (DSR) operator. Mathematical details of this method are found in Section E.7. Figure 6-5 summarizes the main computational steps involved in this migration velocity analysis based on wavefield extrapolation.

1. Starting with the prestack data in midpoint y, off-set h, and two-way event time t in the unmigrated position, represented by the wavefield $P(y, h, \tau = 0, t)$ at the surface $\tau = 0$, perform 3-D Fourier transform. The variable τ is associated with the direction of wave extrapolation, and is related to depth z by $\tau = 2z/\upsilon$, where υ is the medium velocity.
2. Specify an extrapolation velocity function that only varies vertically, $\upsilon(\tau)$ and apply the extrapolation operator $\exp(-i\omega DSR\tau/2)$ to compute the extrapolated wavefield in the transform domain $P(k_y,k_h,\tau,\omega)$ from the surface wavefield in the transform domain $P(k_y,k_h,\tau = 0,\omega)$.
3. To obtain the zero-offset image, sum over the offset wavenumber, and thus obtain $P(k_y,h = 0, \tau, \omega)$.
4. Apply 2-D inverse Fourier transform to obtain the zero-offset image $P(y, h = 0,\tau,t)$. The image below a midpoint y is contained in the $t - \tau$ plane.
5. Perform mapping of the variables as described in Section E.6 from τ *to* υ. The velocity information is given by the envelope of the velocity volume of data $P(y, h =0, \tau = t, \upsilon)$.

We now demonstrate the procedure outlined in Figure 6-5 using a synthetic data set. Figure 6-6 shows two common-offset sections over a number of point scatterers buried in a constant-velocity earth, where υ = 3000 m/s. Using a constant velocity for extrapolation, υ_e = 3000 m/s, the $t - \tau$ *image plane* was produced for each midpoint. Two such planes corresponding to midpoints 1 and 5 denoted in Figure 6-6 are shown in Figure 6-7. The υ–τ planes (Figure 6-8) then were generated from the $t - \tau$ image planes by the mapping procedure described in Section E.7. Peak amplitudes for all events occur at the correct medium velocity (3000 m/s). We expect the diffraction events in Figure 6-5 to migrate to the apexes beneath midpoint 1, where the point scatterers are located. Note that in Figure 6-7, almost all the energy is in the image plane corresponding to midpoint 1; just five midpoints away, at midpoint 5, the migrated energy is very low.

How do we interpret the $t - \tau$ image planes? If we used the true medium velocity in downward extrapolation, then, according to the imaging principle, we would see all the events along the diagonal $\tau = t$, the *image line*, on the image plane. This happens in Figure 6-7, because a 3000-m/s extrapolation velocity was used, which is just the velocity used in generating the model in Figure 6-5. Any displacement of peak energy from the $t = \tau$ image line means that the velocity value used for downward extrapolation differs from that of the event. This displacement is also the basis for mapping from the $t - \tau$ image plane to the $\upsilon - \tau$ plane by equation (E-77).

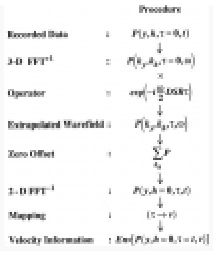


Figure 6-5 A flowchart of an algorithm for focusing analysis.

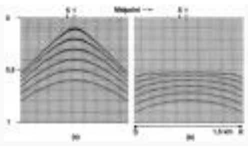


Figure 6-6 Common-offset data derived from a constant-velocity earth model consisting of six point scatterers beneath midpoint 1, where (a) is zero-offset and (b) is far offset.

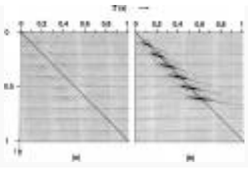


Figure 6-7 Image planes corresponding to midpoints 1 and 5 as indicated in Figure 6-6, where (a) is CMP 5 and (b) is CMP 1.

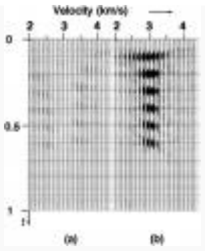


Figure 6-8 The $\upsilon - \tau$ planes corresponding to midpoints 1 and 5 derived from the image planes in Figure 6-7, where (a) is CMP 5 and (b) is CMP 1.

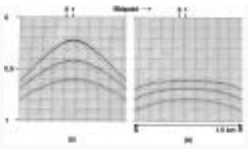


Figure 6-9 Common-offset data based on a horizontally layered earth model containing three point scatterers located beneath midpoint 1 on the boundaries between constant-velocity layers, where (a) is zero-offset and (b) is far offset.

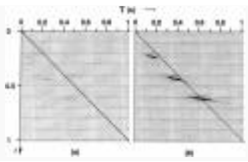
- 

Figure 6-10 Image planes corresponding to midpoints 1 and 5 as indicated in Figure 6-9, where (a) is CMP 5 and (b) is CMP 1.

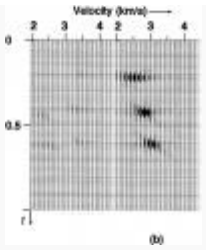
- 

Figure 6-11 The $\upsilon - \tau$ planes corresponding to midpoints 1 and 5 derived from the image planes in Figure 6-10, where (a) is CMP 5 and (b) is CMP 1.

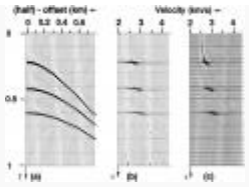
- 

Figure 6-12 (a) CMP gather at location 1 as indicated in Figure 6-7; (b) and (c) are velocity spectra derived from this gather by the methods of Figures 6-4 and 6-5, respectively.

This mapping is investigated further with the modeled data set shown in Figure 5.4-27, in which velocity increases with depth. In Figure 6-10, note that the top and middle events fall to the left of the image line suggesting that the velocity used in extrapolation (υ_e = 3000 m/s) is greater than the velocities associated with these events. The bottom event falls on the image line, implying that its velocity is nearly the same as the extrapolation velocity. These observations are confirmed in the corresponding $\upsilon - \tau$ planes in Figure 6-10. While true stacking velocity values for the three events are 2700, 2850, and 3000 m/s, the velocities interpreted from Figure 6-10 are about 2500, 2800, and 3000 m/s. Thus, the migration-based velocity estimate for the shallow event is in error by approximately 8 percent.

To determine the reason for the velocity error, we will consider a migration-based velocity analysis of our synthetic data example that does not involve the approximate mapping step. Figure 6-11 shows a CMP gather from midpoint 1 in the zero-dip region of the depth-variable velocity model associated with the constant-offset sections in Figure 6-10. The migration velocity analysis on this gather (Figure 6-12) was done by extrapolating the surface wavefield $P(k_h,\ \omega,\ \tau = 0)$ repeatedly with different constant velocities in steps of $\Delta\tau = \Delta t$ (the sampling rate). The zero-offset trace from each attempt was collected after this effort, abandoning the rest of the migrated CMP gather.

Interpretation of the velocity analysis in Figure 6-12 reveals correct stacking velocities for the three events, including the shallowest. Clearly the error observed in Figure 6-9 is attributable to the mapping (equation (E-100). Note that the error does not occur because of depth variability of velocity, but instead, because the single extrapolation velocity used differed from the medium velocity. The conventional velocity analysis for midpoint 1 of this model data set is shown in Figure 6-12 for comparison. Note the familiar NMO stretching that is apparent in the shallow event. In other respects, both the results (Figures 6-11 and 6-12) are comparable.

The departure of an event on the $t - \tau$ image plane from the $t = \tau$ image line is measured by the quantity $\Delta\tau$ as depicted in Figure 5.4-31a. In some practical implementations, the $t - \tau$ image plane is mapped onto the plane of $\Delta\tau$ versus τ as depicted in Figure 6-12 to determine the rms velocity $\upsilon(\tau)$ for time migration from the extrapolation velocity $\upsilon_e(\tau)$. An event with a velocity error $\upsilon(\tau) - \upsilon_e(\tau)$ is represented by an energy maximum either to the left or to the right of the $\Delta\tau = 0$ line. The $\delta\tau(\tau)$ trend can be picked and translated into a velocity trend as depicted in Figure 6-12. This type of analysis has come to be called focusing analysis in the industry (Faye and Jeannaut, 1986). It has been used in some cases erroneously to estimate and update velocity-depth models used for depth migration. The method can only provide plausable velocity update within the framework of time migration.

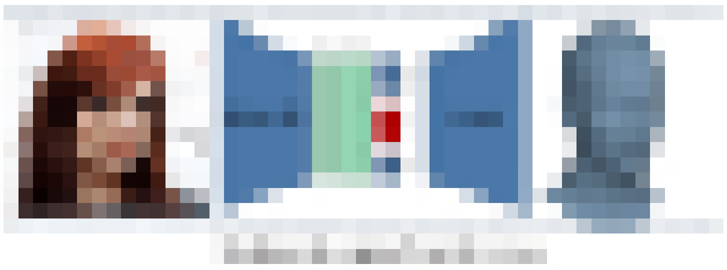
Figure 6-10 is a CMP stack from offshore Texas. A 7000-ft portion (64 midpoints each with 48 offset traces) of the profile was used for migration velocity analysis. For computational efficiency, the data were windowed into 1024-ms time gates with 50 percent overlap. The image planes for one particular midpoint are shown in Figure 6-11. Different extrapolation velocities picked from a specified regional velocity function are used in each time gate. The velocity scan used in mapping is then carried out within a corridor around this function. Because different extrapolation velocities are used in successive segments, a given event appears at different values of τ in adjacent time segments.

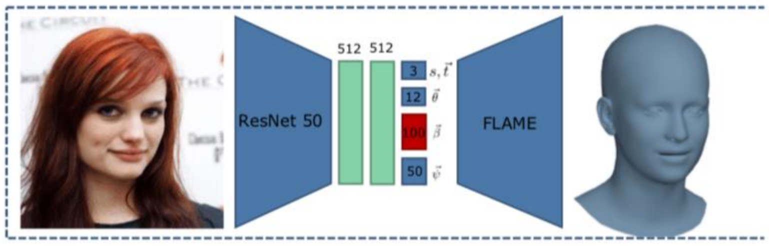
The resulting velocity analysis for the central midpoint is shown in Figure 6-12. In conventional practice, to improve the quality of velocity picks, velocity analyses from a number of neighboring CMP gathers often are summed. Figure 6-12 shows the result of stacking velocity analysis for data from the six adjacent CMP gathers indicated in Figure 6-12. For the migration-based method, the $\upsilon - \tau$ planes corresponding to these gathers were summed. The result is shown in Figure 6-12. The most obvious difference between the two results is the lack of shallow information in the migration-based $\upsilon - \tau$ plane. This shortcoming is attributed to spatial aliasing and lack of long-offset data in the shallow time gate. The problem can be eliminated partly by increasing the length of the time gate used in the velocity analysis. With the shortcut time-windowing approach described above, the shallowest time segment did not include the large-offset data necessary for velocity resolution. Because the events have dip, the derived migration velocities are lower (by up to 4.5 percent) than the velocities derived from the stacking velocity analysis.

The velocity analysis described in this section does not handle lateral variations in velocity. It is based on a Fourier-transform domain formulation with only vertically varying velocity used in extrapolation. This method may be particularly efficient for the dip-corrected velocity estimate needed for time migration.

6.3 <u>3D face Algorithm</u>

The estimation of 3D face shape from a single image must be robust to variations in lighting, head pose, expression, facial hair, makeup, and occlusions. Robustness requires a large training set of in-the-wild images, which by construction, lack ground truth 3D shape.” (MPIIS).
In a new paper accepted at CVPR 2019, researchers from the Max Planck Institute for Intelligent Systems introduce RingNet, an end-to-end trainable network which learns to compute 3D face shape from a single face image without 3D supervision. The researchers also built a new benchmark dataset and a 3D reconstruction benchmark challenge, NoW, both of which have been open-sourced on <u>Github</u>.
The Max Planck Institute responded to Synced questions regarding their new paper, RingNet and the open challenge.
How would you describe RingNet?
RingNet is an end-to-end trainable network that enforces shape consistency across face images of the subject with varying viewing angles, light conditions, resolution, and occlusion. It is able to learn 3D face geometry from 2D images, but it only need single image for inference.





Ring element that outputs a 3D mesh for an image.

Figure 6-13 Ring Element Output
Why does this research matter?
The idea of RingNet is quite general even if it is only used for faces. One can potentially use this idea for other 3D reconstruction purposes. In this work, our researchers also introduce a 3D reconstruction benchmark challenge NoW and an evaluation metric to provide the research community with quantitative feedback which was lacking in this field. The aim is to encourage other researchers to participate in this challenge and go beyond visual comparisons.
Since people can reconstruct a 3D face from single images with neck and full head, the technique can potentially be used for the animation industry or different face apps. There could be many interesting applications by combining RingNet and <u>VOCA project</u> (voice

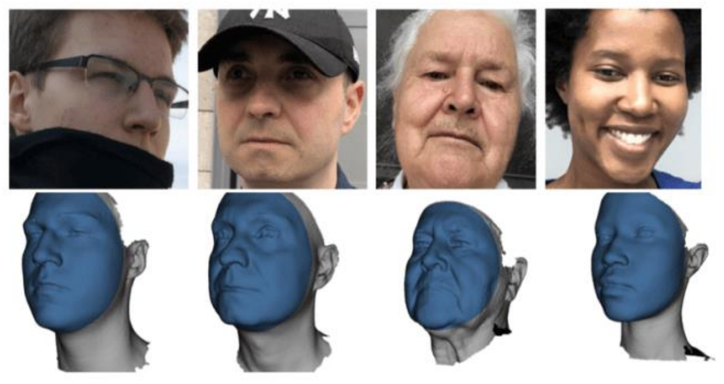
driven face animation model), for example, Using RingNet to prepare a template mesh for VOCA, then animate it with audio, i.e., a talking head from a face image.

Could you describe the Challenge NoW in more detail?

The goal of this benchmark is to measure the accuracy and robustness of 3D face reconstruction methods under variations in viewing angle, lighting, and common occlusions by a standard evaluation metric.

The NoW Dataset introduced to run the challenge contains 2054 2D images of 100 subjects, captured with an iPhone X, and a separate 3D head scan for each subject. The head scans serve as ground truth for the evaluation. The subjects were selected to contain variations in age, BMI, and sex (55 female, 45 male).

The challenge for all categories is to reconstruct a neutral 3D face given a single monocular image. Note that facial expressions are present in several images, which requires methods to disentangle identity and expression to evaluate the quality of the predicted identity.



The NoW dataset includes a variety of images take in different conditions (top) and high-resolution 3D head scans (bottom). The dark blue region is the part we considered for face challenge.

Figure 6-14 (a) Blur 2D image (b) Perfect 3D image

Can you identify any bottlenecks in the research?

A bottleneck of the research topic is people tend to rely on only 2D landmarks. This certainly constrains the quality of the 3D reconstruction to some extent. Using dense correspondences should be able to push the limit to a new level.

Why do we need 3D when 2D is already looking good?

People may find some 2D face animations (like the Obama lip sync, Kumar, Rithesh, et al 2017) are already quite realistic. Although their results can look good by learning from huge datasets, we are lacking the ability to manipulate these 2D models accurately. Additionally, looking good is not enough, we need to understand what’s really going on. We live in a 3D world, this is what’s really behind every 2D picture and movie frame. So without 3D information, we can’t ask a GAN which is only trained with 2D images and landmarks to maintain the face shape in each frame when it’s rotating. A 3D model also gets more correspondence than 2D for example regarding each pixel’s relevance. Nowadays, face tracking by 2D landmark localization performs pretty well, but landmarks alone can’t provide the dense correspondence between frames. This is the key motivation for making 3D reconstruction more accurate and robust.

6.4 Hurestic Algorithm

6.4.1 The "no machine learning" challenge

Inspired by Nick's post, I decided to challenge myself to explore if similar results could be achieved *without* the use of machine learning. It struck me that the bottles used in the original demo could be detected based on their colour or other characteristics along with some simple matching rules. This is known as an heuristic approach to problem solving.
Potential advantages of this include:

- Ease of development and conceptualisation
- Lower CPU and memory use
- Fewer dependencies

In terms of CPU and memory, on my i5 MacBook Pro, the IBM Cloud Annotations demo uses over 100% CPU and more than 1.5 *Gigabytes* of RAM. It also relies on a web browser and some heavy dependencies including Tensorflow, React.js, node.js and COCO-SSD itself.
The rules I set myself are:

1. Coke, Pepsi and Mountain Dew bottles must be labelled correctly
2. A rectangle should be drawn around each bottle as it moves
3. Minimal code
4. No machine learning techniques!

The original demo claims to use only 10 lines of code, however including boilerplate, the current demo is 107 lines of JavaScript. I think under 100 lines is a good aim for this task.

6.4.2 Approach and solution

Firstly, I decided to base my project in OpenCV since I have previously used it for work projects, it has relatively easy setup and is designed *specifically* for computer vision. OpenCV is written in C++ and has bindings in Python and JavaScript. I decided to go with the Python version for convenience.
I started with just recognising a Coke bottle. For this, a naïve solution would be to analyse the colours in a video frame and place a label where coke red is found. One problem here is that depending on the lighting conditions and camera colour accuracy, the bottle label is unlikely to be *exactly* RGB 244 0 0.
To solve this, we can use a HSV colour representation along with cv::inRange to find colours within the image that are within a given *range*. Think "shades of red". This gives us an image mask with all the red coloured areas white and everything else black. We can then use cv::findContours to supply a list of points that define each "red area" within the frame.

The basic code will look something like this:

```
mask = cv2.inRange(hsv, colour.lower, colour.upper)
conts, heirarchy = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
biggest = sorted(conts, key=cv2.contourArea, reverse=True)[0]
```

The third line of code sorts the detected "red" contours and returns the largest one. Done! …right? Well unfortunately not. Left like this, the program often finds Coke in the image even when there is none.

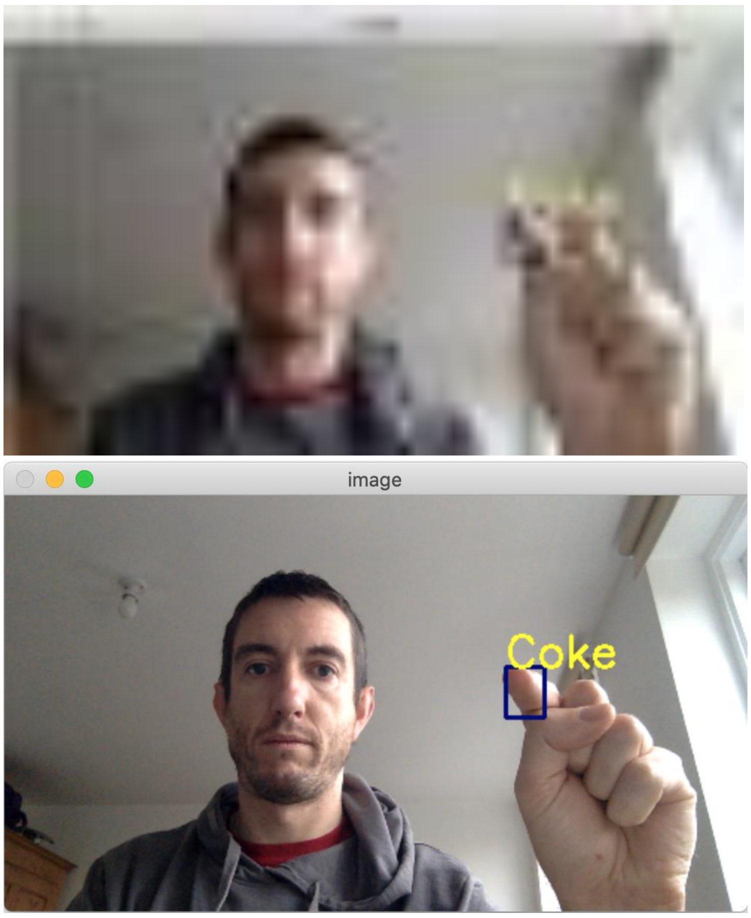


Figure 6-15 Coke false positive

To deal with this we need an additional heuristic. I found simply excluding any contour smaller than 50×50 worked well enough.

```
if w < 50 or h < 50:
    continue
```

Finally, for our detection system to work well, we need to exclude colours that are found "inside" other colours. For example both the Pepsi and Mountain Dew labels *contain* red, which will get detected as Coke unless we exclude it. So we add a special heuristic for Coke that ignores detection if it is within the vertical bounds of another bottle.

```
if name == "Coke":
    if any([contains_vertical(rects[n], rect) for n in rects]):
        continue
```

6.4.3 Demonstration

Putting it all together, here is a working demonstration of the final system.On my i5 MacBook Pro this runs smoothly at around 45% CPU with just over 50MB RAM. The full source code comes to 85 lines and is available here.

6.4.4 Limitations

One of the limitations of this colour-based approach is that it doesn’t place the bounding box around the bottle but only the coloured area. We could define additional rules to consider the colour above or below the detected region, or attempt to guess where the bounding box should be, but the code would quickly become complicated.

Another limitation is that whilst our system can recognise a Coke and a Pepsi bottle at the same time, it can’t detect two Coke bottles. We could add further heuristics to deal with this but I would question if an heuristic approach is the right choice if so much complexity needs to be added.

6.4.5 Deep learning vs heuristics

I have shown that it’s straightforward to build a heuristic detector with accuracy comparable to that of a deep learning-based system for a highly constrained task. Furthermore, the heuristic object detector is conceptually simpler, has fewer dependencies, takes significantly less CPU and uses an order-of-magnitude less memory.

However, the heuristic approach is not as robust or accurate as using deep learning. A deep learning system can trivially recognise multiple instances of the same object at different scales and rotations, depending on how it is trained. It can also do things like recognise partial objects even if key features are missing.

6.5 Conclusion

For me, this isn’t a clear win for deep learning and I think there still is a place for an heuristic approach. The more assumptions that can be made about the detection conditions (consistent background and / or scale, constrained object types, distinguishing features such as colour) the more appeal heuristics have. As a developer, I would consider a heuristic based solution if time and resources were tight and the input constraints were clearly defined. If I wanted increased robustness and flexibility, I would opt for machine learning. Both approaches definitely have their place, and it’s a question of choosing the right tool for the job.

Chapter 7

7. GUI (Graphical User Interface)

7.1 What is liveness detection and why do we need it?

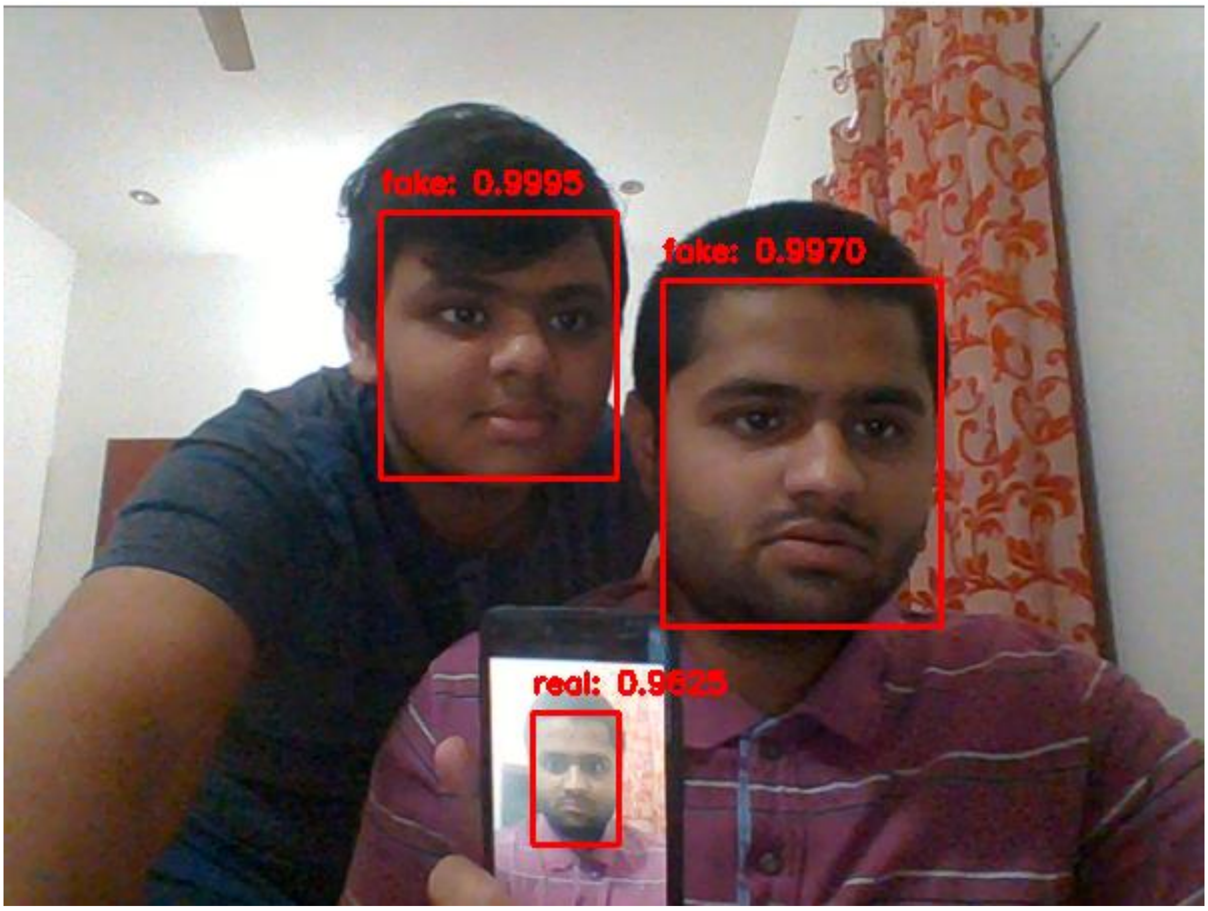


Figure 7.1: Liveness detection with OpenCV. On the left is a live (fake) video of me with my brother and on the right we can see I am holding my Phone (real).

Face recognition systems are becoming more prevalent than ever. From face recognition on our iPhone/smartphone, to face recognition for mass surveillance in China, face recognition systems are being utilized everywhere.

However, face recognition systems are easily fooled by "spoofing" and "non-real" faces.

Face recognition systems can be circumvented simply by holding up a photo of a person (whether printed, on a smartphone, etc.) to the face recognition camera. In order to make face recognition systems more secure, we need to be able to detect such fake/non-real faces — **liveness detection is the term used to refer to such algorithms.**

There are a number of approaches to liveness detection, including:

- **Texture analysis,** including computing **Local Binary Patterns** (LBPs) over face regions and using an SVM to classify the faces as real or spoofed.
- **Frequency analysis,** such as examining the Fourier domain of the face.
- **Variable focusing analysis,** such as examining the variation of pixel values between two consecutive frames.
- **Heuristic-based algorithms,** including **eye movement, lip movement,** and **blink detection.** These set of algorithms attempt to track eye movement and blinks to ensure the user is not holding up a photo of another person (since a photo will not blink or move its lips).
- **Optical Flow algorithms,** namely examining the differences and properties of optical flow generated from 3D objects and 2D planes.
- **3D face shape,** similar to what is used on Apple's iPhone face recognition system, enabling the face recognition system to distinguish between real faces and printouts/photos/images of another person.
- **Combinations of the above,** enabling a face recognition system engineer to pick and choose the liveness detections models appropriate for their particular application.

A full review of liveness detection algorithms can be found in Chakraborty and Das' 2014 paper, **An Overview of Face liveness Detection**. For the purposes of today's tutorial, we'll be **treating liveness detection as a binary classification problem**. Given an input image, we'll train a Convolutional Neural Network capable of distinguishing real faces from fake/spoofed faces. But before we get to training our liveness detection model, let's first examine our dataset.

Our liveness detection videos

To keep our example straightforward, the liveness detector we are building in this blog post will focus on distinguishing **real faces** versus **spoofed faces on a screen.** This algorithm can easily be extended to other types of spoofed faces, including print outs, high-resolution prints, etc.

In order to build the liveness detection dataset, I:

1. Took my iPhone and put it in **portrait/selfie mode.**
2. **Recorded a ~25-second video of myself** walking around my office.
3. **Replayed the same 25-second video, this time facing my Phone towards my desktop** where I recorded the video replaying.
4. This resulted in **two example videos**, one for "real" faces and another for "fake/spoofed" faces.

5. Finally, I applied **face detection** to both sets of videos to extract individual face ROIs for both classes.

I have provided with both my real and fake video files in the **“Downloads”** section of the post.

We can use these videos as a starting point for our dataset but **I would recommend gathering more data to help make our liveness detector more robust and accurate.** With testing, I determined that the model is slightly biased towards my own face which makes sense because that is all the model was trained on. And furthermore, since I am white/Caucasian I wouldn’t expect this same dataset to work as well with other skin tones.

Ideally, we would train a model with **faces of multiple people** and **include faces of multiple ethnicities.** Be sure to refer to the “Limitations and further work“section below for additional suggestions on improving our liveness detection models. In the rest of the tutorial, we will learn how to take the dataset I recorded it and turn it into an actual liveness detector with OpenCV and deep learning.

7.1.1 Project structure

Go ahead and grab the code, dataset, and liveness model using the **“Downloads”** section of this post and then unzip the archive.Once we navigate into the project directory, we’ll notice the following structure:

```
liveness Detection with OpenCV
$ tree --dirsfirst --filelimit 10
.
├── dataset
│   ├── fake [150 entries]
│   └── real [161 entries]
├── face_detector
│   ├── deploy.prototxt
│   └── res10_300x300_ssd_iter_140000.caffemodel
├── pyimagesearch
│   ├── __init__.py
│   └── livenessnet.py
├── videos
│   ├── fake.mp4
│   └── real.mov
├── gather_examples.py
├── train_liveness.py
├── liveness_demo.py
├── le.pickle
├── liveness.model
└── plot.png
6 directories, 12 files
```

There are four main directories inside our project:

- dataset/

  : Our dataset directory consists of two classes of images:

  - Fake images of me from a camera aimed at my screen while playing a video of my face.
  - Real images of me captured from a selfie video with my phone.

- face_detector/

  : Consists of our pretrained Caffe face detector to locate face ROIs.

- pyimagesearch/

  : This module contains our LivenessNet class.

- videos/

  : I’ve provided two input videos for training our LivenessNet classifier.

Today we’ll be reviewing three Python scripts in detail. By the end of the post we’ll be able to run them on our own data and input video feeds as well. In order of appearance in this tutorial, the three scripts are:

1. gather_examples.py
   : This script grabs face ROIs from input video files and helps us to create a deep learning face liveness dataset.

2. train_liveness.py
   : As the filename indicates, this script will train our LivenessNet classifier. We’ll use Keras and TensorFlow to train the model. The training process results in a few files:

   - le.pickle
     : Our class label encoder.

   - liveness.model
     : Our serialized Keras model which detects face liveness.

   - plot.png
     : The training history plot shows accuracy and loss curves so we can assess our model (i.e. over/underfitting).

3. liveness_demo.py
   : Our demonstration script will fire up our webcam to grab frames to conduct face liveness detection in real-time.

7.1.2 Detecting and extracting face ROIs from our training (video) dataset

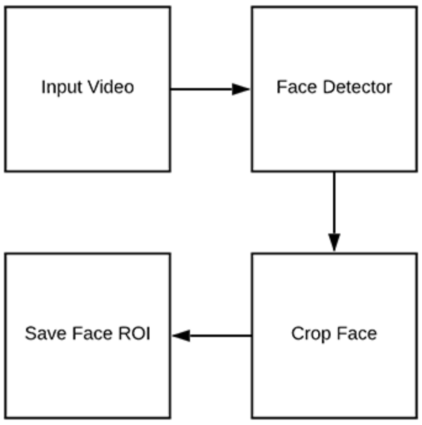


Figure 7.2: Detecting face ROIs in video for the purposes of building a liveness detection dataset.

Now that we’ve had a chance to review both our initial dataset and project structure, let’s see how we can extract both real and fake face images from our input videos.

The end goal if this script will be to populate two directories:

1. dataset/fake/ : Contains face ROIs from the *fake.mp4* file.
2. dataset/real/ : Holds face ROIs from the *real.mov* file.

Given these frames, we’ll later train a deep learning-based liveness detector on the images.

Open up the *gather_examples.py* file and insert the following code:

```
# import the necessary packages
import numpy as np
import argparse
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", type=str, required=True,
help="path to input video")
ap.add_argument("-o", "--output", type=str, required=True,
help="path to output directory of cropped faces")
ap.add_argument("-d", "--detector", type=str, required=True,
help="path to OpenCV's deep learning face detector")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
help="minimum probability to filter weak detections")
```

```
17 ap.add_argument("-s", "--skip", type=int, default=16,
18 help="# of frames to skip before applying face detection")
19 args = vars(ap.parse_args())
```

Lines 2-5 import our required packages. This script only requires OpenCV and NumPy in addition to built-in Python modules.

From there **Lines 8-19** **parse our command line arguments**:

- --input

  : The path to our input video file.

- --output

  : The path to the output directory where each of the cropped faces will be stored.

- --detector

  : The path to the face detector. We'll be using **OpenCV's deep learning face detector**. This Caffe model is included with today's ***"Downloads"*** for our convenience.

- --confidence

  : The minimum probability to filter weak face detections. By default, this value is 50%.

- --skip

  : We don't need to detect and store every image because adjacent frames will be similar. Instead, we'll skip *N* frames between detections. We can alter the default of 16 using this argument.

Let's go ahead and load the face detector and initialize our video stream:

```
21 # load our serialized face detector from disk
22 print("[INFO] loading face detector...")
23 protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
24 modelPath = os.path.sep.join([args["detector"],
25 "res10_300x300_ssd_iter_140000.caffemodel"])
26 net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
27
28 # open a pointer to the video file stream and initialize the total
29 # number of frames read and saved thus far
30 vs = cv2.VideoCapture(args["input"])
31 read = 0
32 saved = 0
```

Lines 23-26 load **OpenCV's deep learning face detector**.

From there we open our video stream on **Line 30**. We also initialize two variables for the number of frames read as well as the number of frames saved while our loop executes (**Lines 31 and 32**). Let's go ahead create a loop to process the frames:

```
34 # loop over frames from the video file stream
35 while True:
36 # grab the frame from the file
37 (grabbed, frame) = vs.read()
38
39 # if the frame was not grabbed, then we have reached the end
40 # of the stream
41 if not grabbed:
42 break
43
44 # increment the total number of frames read thus far
45 read += 1
46
47 # check to see if we should process this frame
48 if read % args["skip"] != 0:
49 continue
```

Our while loop begins on **Lines 35**. From there we grab and verify a frame (**Lines 37-42**).At this point, since we've read a frame, we'll increment our readcounter (**Line 48**). If we are skipping this particular frame, we'll continue without further processing (**Lines 48 and 49**).

Let's go ahead and detect faces:

```
51 # grab the frame dimensions and construct a blob from the frame
52 (h, w) = frame.shape[:2]
53 blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
54 (300, 300), (104.0, 177.0, 123.0))
55
56 # pass the blob through the network and obtain the detections and
57 # predictions
58 net.setInput(blob)
59 detections = net.forward()
60
61 # ensure at least one face was found
62 if len(detections) > 0:
63 # we're making the assumption that each image has only ONE
64 # face, so find the bounding box with the largest probability
65 i = np.argmax(detections[0, 0, :, 2])
66 confidence = detections[0, 0, i, 2]
```

In order to perform face detection, we need to **create a blob from the image** (**Lines 53 and 54**). This blob has a *300×300* width and height to accommodate our Caffe face detector. Scaling the bounding boxes will be necessary later, so **Line 52**, grabs the frame dimensions. **Lines 58 and 59** perform a forward pass of the blob through the deep learning face detector.

Our script *makes the assumption that there is only one face in each frame* of the video (**Lines 62-65**). This helps prevent false positives. If We're working with a video containing more than one face, I recommend that we adjust the logic accordingly. Thus, **Line 65** grabs the highest probability face detection index. **Line 66** extracts the confidence of the detection using the index.

Let's filter weak detections and write the face ROI to disk:

```
68 # ensure that the detection with the largest probability also
69 # means our minimum probability test (thus helping filter out
70 # weak detections)
71 if confidence > args["confidence"]:
72 # compute the (x, y)-coordinates of the bounding box for
73 # the face and extract the face ROI
74 box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
75 (startX, startY, endX, endY) = box.astype("int")
76 face = frame[startY:endY, startX:endX]
77
78 # write the frame to disk
79 p = os.path.sep.join([args["output"],
80 "{}.png".format(saved)])
81 cv2.imwrite(p, face)
82 saved += 1
83 print("[INFO] saved {} to disk".format(p))
84
85 # do a bit of cleanup
86 vs.release()
87 cv2.destroyAllWindows()
```

Line 71 ensures that our face detection ROI meets the minimum threshold to reduce false positives.From there we extract the face ROI bounding box coordinates and face ROI itself (**Lines 74-76**). We generate a path + filename for the face ROI and write it to disk on **Lines 79-81**. At this point, we can increment the number of saved faces. Once processing is complete, we'll perform cleanup on **Lines 86 and 87**.

7.1.3 Building our liveness detection image dataset

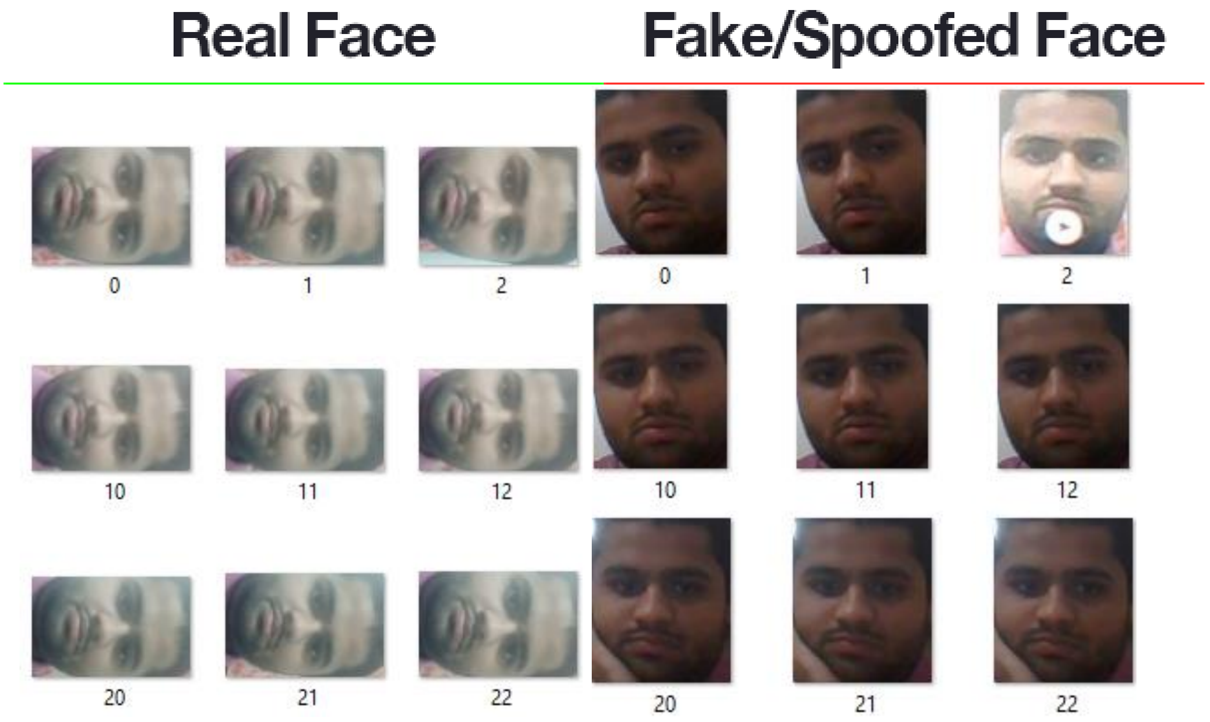


Figure 7.3: Our OpenCV face liveness detection dataset. We'll use Keras and OpenCV to train and demo a liveness model.

Now that we've implemented the gather_examples.py script, let's put it to work. Make sure we use the ***"Downloads"*** section of this tutorial to grab the source code and example input videos.

From there, open up a terminal and execute the following command to extract faces for our **"fake/spoofed"** class:

```
liveness Detection with OpenCV
$ python gather_examples.py --input videos/fake.mp4 --output dataset/fake \
--detector face_detector --skip 1
[INFO] loading face detector...
[INFO] saved datasets/fake/0.png to disk
[INFO] saved datasets/fake/1.png to disk
[INFO] saved datasets/fake/2.png to disk
[INFO] saved datasets/fake/3.png to disk
[INFO] saved datasets/fake/4.png to disk
[INFO] saved datasets/fake/5.png to disk
...
[INFO] saved datasets/fake/145.png to disk
[INFO] saved datasets/fake/146.png to disk
[INFO] saved datasets/fake/147.png to disk
[INFO] saved datasets/fake/148.png to disk
[INFO] saved datasets/fake/149.png to disk
```

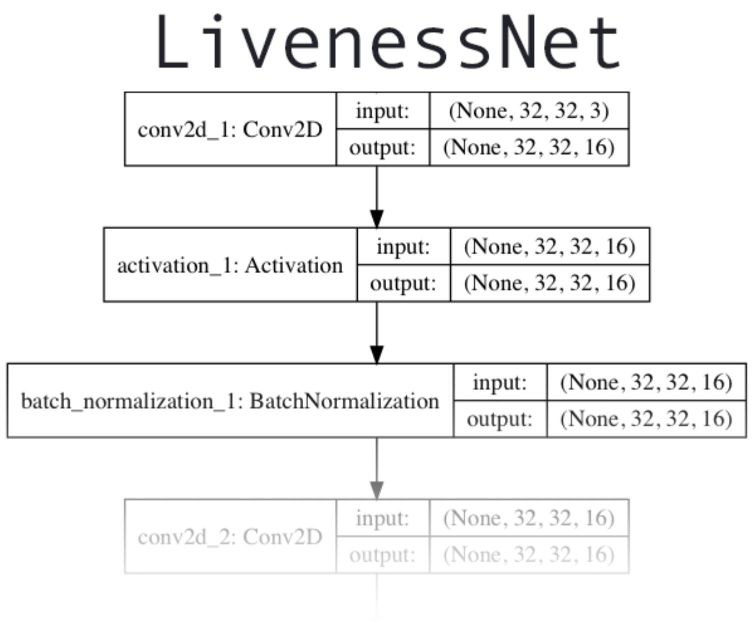
Similarly, we can do the same for the **“real”** class as well:

```
liveness Detection with OpenCV
$ python gather_examples.py --input videos/real.mov --output dataset/real \
--detector face_detector --skip 4
[INFO] loading face detector...
[INFO] saved datasets/real/0.png to disk
[INFO] saved datasets/real/1.png to disk
[INFO] saved datasets/real/2.png to disk
[INFO] saved datasets/real/3.png to disk
[INFO] saved datasets/real/4.png to disk
...
[INFO] saved datasets/real/156.png to disk
[INFO] saved datasets/real/157.png to disk
[INFO] saved datasets/real/158.png to disk
[INFO] saved datasets/real/159.png to disk
[INFO] saved datasets/real/160.png to disk
```

Since the “real” video file is longer than the “fake” video file, we’ll use a longer skip frames value to help balance the number of output face ROIs for each class. After executing the scripts we should have the following image counts:

- **Fake:** 150 images
- **Real:** 161 images
- ***Total:*** 311 images

Implementing “LivenessNet”, our deep learning liveness detector



LivenessNet
conv2d_1: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 16)
activation_1: Activation
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
batch_normalization_1: BatchNormalization
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
conv2d_2: Conv2D
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)

Figure 7.4: Deep learning architecture for LivenessNet, a CNN designed to detect face liveness in images and videos.

The next step is to implement "LivenessNet", our deep learning-based liveness detector. At the core, LivenessNet is actually just a simple Convolutional Neural Network.

We'll be *purposely* keeping this network as *shallow* and *with as few parameters as possible* for two reasons:

1. To reduce the chances of overfitting on our small dataset.

2. To ensure our liveness detector is fast, capable of running in real-time (even on resource-constrained devices, such as the Raspberry Pi).

Let's implement LivenessNet now — open up

```
1 # import the necessary packages
2 from keras.models import Sequential
3 from keras.layers.normalization import BatchNormalization
4 from keras.layers.convolutional import Conv2D
5 from keras.layers.convolutional import MaxPooling2D
6 from keras.layers.core import Activation
7 from keras.layers.core import Flatten
8 from keras.layers.core import Dropout
9 from keras.layers.core import Dense
10 from keras import backend as K
11
12 class LivenessNet:
13 @staticmethod
14 def build(width, height, depth, classes):
15 # initialize the model along with the input shape to be
16 # "channels last" and the channels dimension itself
17 model = Sequential()
18 inputShape = (height, width, depth)
19 chanDim = -1
20
21 # if we are using "channels first", update the input shape
22 # and channels dimension
23 if K.image_data_format() == "channels_first":
24 inputShape = (depth, height, width)
25 chanDim = 1
```

All of our imports are from Keras (**Lines 2-10**). For an in-depth review of each of these layers and functions, be sure to refer to ***Deep Learning for Computer Vision with Python***. Our LivenessNet class is defined on **Line 12**. It consists of one static method, build (**Line 14**). The build method accepts four parameters:

- width

  : How wide the image/volume is.

- height

  : How tall the image is.

- depth

  : The number of channels for the image (in this case 3 since we'll be working with RGB images).

- classes

  : The number of classes. We have two total classes: "real" and "fake".

Our model is initialized on **Line 17**.

The inputShape to our model is defined on **Line 18** while channel ordering is determined on **Lines 23-25.**

Let's begin adding layers to our CNN:

```
27 # first CONV => RELU => CONV => RELU => POOL layer set
28 model.add(Conv2D(16, (3, 3), padding="same",
29 input_shape=inputShape))
30 model.add(Activation("relu"))
31 model.add(BatchNormalization(axis=chanDim))
32 model.add(Conv2D(16, (3, 3), padding="same"))
33 model.add(Activation("relu"))
34 model.add(BatchNormalization(axis=chanDim))
35 model.add(MaxPooling2D(pool_size=(2, 2)))
36 model.add(Dropout(0.25))
37
38 # second CONV => RELU => CONV => RELU => POOL layer set
39 model.add(Conv2D(32, (3, 3), padding="same"))
40 model.add(Activation("relu"))
41 model.add(BatchNormalization(axis=chanDim))
42 model.add(Conv2D(32, (3, 3), padding="same"))
43 model.add(Activation("relu"))
44 model.add(BatchNormalization(axis=chanDim))
45 model.add(MaxPooling2D(pool_size=(2, 2)))
46 model.add(Dropout(0.25))
```

Our CNN exhibits VGGNet-esque qualities. It is very shallow with only a few learned filters. Ideally, we won't need a deep network to distinguish between real and spoofed faces.

The first CONV => RELU => CONV => RELU => POOL layer set is specified on **Lines 28-36** where batch normalization and dropout are also added. Another CONV => RELU => CONV => RELU => POOL layer set is appended on **Lines 39-46**. Finally, we'll add our FC => RELU layers:

```
48 # first (and only) set of FC => RELU layers
49 model.add(Flatten())
50 model.add(Dense(64))
51 model.add(Activation("relu"))
52 model.add(BatchNormalization())
53 model.add(Dropout(0.5))
54
55 # softmax classifier
56 model.add(Dense(classes))
57 model.add(Activation("softmax"))
58
59 # return the constructed network architecture
60 return model
```

Lines 49-57 consist of fully connected and ReLU activated layers with a softmax classifier head. The model is returned to the training script on **Line 60**.

7.1.4 Creating the liveness detector training script

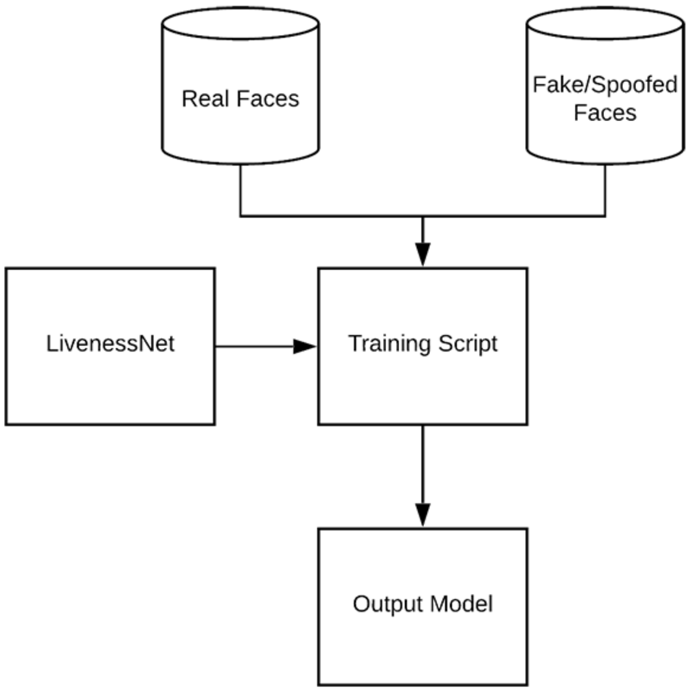


Figure 7.5: The process of training LivenessNet. Using both "real" and "spoofed/fake" images as our dataset, we can train a liveness detection model with OpenCV, Keras, and deep learning.

Given our dataset of real/spoofed images as well as our implementation of LivenessNet, we are now ready to train the network.

Open up the *train_liveness.py* file and insert the following code:

```
# set the matplotlib backend so figures can be saved in the background
import matplotlib
matplotlib.use("Agg")

# import the necessary packages
from pyimagesearch.livenessnet import LivenessNet
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.utils import np_utils
```

```
13 from imutils import paths
14 import matplotlib.pyplot as plt
15 import numpy as np
16 import argparse
17 import pickle
18 import cv2
19 import os
20
21 # construct the argument parser and parse the arguments
22 ap = argparse.ArgumentParser()
23 ap.add_argument("-d", "--dataset", required=True,
24 help="path to input dataset")
25 ap.add_argument("-m", "--model", type=str, required=True,
26 help="path to trained model")
27 ap.add_argument("-l", "--le", type=str, required=True,
28 help="path to label encoder")
29 ap.add_argument("-p", "--plot", type=str, default="plot.png",
30 help="path to output loss/accuracy plot")
31 args = vars(ap.parse_args())
```

Our face liveness training script consists of a number of imports (**Lines 2-19**). Let's review them now:

- matplotlib

  : Used to generate a training plot. We specify the "Agg" backend so we can easily save our plot to disk on **Line 3**.

- LivenessNet

  : The liveness CNN that we defined in the previous section.

- train_test_split

  : A function from scikit-learn which constructs splits of our data for training and testing.

- classification_report

  : Also from scikit-learn, this tool will generate a brief statistical report on our model's performance.

- ImageDataGenerator

  : Used for performing data augmentation, providing us with batches of randomly mutated images.

- Adam

: An optimizer that worked well for this model. (alternatives include SGD, RMSprop, etc.).

- paths

  : From my imutils package, this module will help us to gather the paths to all of our image files on disk.

- pyplot

  : Used to generate a nice training plot.

- numpy

  : A numerical processing library for Python. It is an OpenCV requirement as well.

- argparse

  : For processing **command line arguments**.

- pickle

  : Used to serialize our label encoder to disk.

- cv2

  : Our OpenCV bindings.

- os

  : This module can do quite a lot, but we'll just be using it for it's operating system path separator.

That was a mouthful, but now that we know what the imports are for, reviewing the rest of the script should be more straightforward. This script accepts four command line arguments:

- --dataset

  : The path to the input dataset. Earlier in the post we created the dataset with the gather_examples.py script.

- --model

  : Our script will generate an output model file — here we supply the path to it.

- --le

  : The path to our output serialized label encoder file also needs to be supplied.

- --plot

  : The training script will generate a plot. If we wish to override the default value of "plot.png" , we should specify this value on the command line.

This next code block will perform a number of initializations and build our data:

```
33 # initialize the initial learning rate, batch size, and number of
34 # epochs to train for
35 INIT_LR = 1e-4
36 BS = 8
37 EPOCHS = 50
38
39 # grab the list of images in our dataset directory, then initialize
40 # the list of data (i.e., images) and class images
41 print("[INFO] loading images...")
42 imagePaths = list(paths.list_images(args["dataset"]))
43 data = []
44 labels = []
45
46 for imagePath in imagePaths:
47 # extract the class label from the filename, load the image and
48 # resize it to be a fixed 32x32 pixels, ignoring aspect ratio
49 label = imagePath.split(os.path.sep)[-2]
50 image = cv2.imread(imagePath)
51 image = cv2.resize(image, (32, 32))
52
53 # update the data and labels lists, respectively
54 data.append(image)
55 labels.append(label)
56
57 # convert the data into a NumPy array, then preprocess it by scaling
58 # all pixel intensities to the range [0, 1]
59 data = np.array(data, dtype="float") / 255.0
```

Training parameters including initial learning rate, batch size, and number of epochs are set on **Lines 35-37**.From there, our imagePaths are grabbed. We also initialize two lists to hold our data and class labels (**Lines 42-44**).The loop on **Lines 46-55** builds our data and labels lists. The data consists of our images which are loaded and resized to be *32×32* pixels. Each image has a corresponding label stored in the labels list. All pixel intensities are scaled to the range *[0, 1]* while the list is made into a NumPy array via **Line 59**.

Now let’s encode our labels and partition our data:

```
61 # encode the labels (which are currently strings) as integers and then
62 # one-hot encode them
63 le = LabelEncoder()
64 labels = le.fit_transform(labels)
65 labels = np_utils.to_categorical(labels, 2)
66
67 # partition the data into training and testing splits using 75% of
68 # the data for training and the remaining 25% for testing
69 (trainX, testX, trainY, testY) = train_test_split(data, labels,
70 test_size=0.25, random_state=42)
```

Lines 63-65 one-hot encode the labels. We utilize scikit-learn to partition our data — 75% is used for training while 25% is reserved for testing (**Lines 69 and 70**). Next, we'll initialize our data augmentation object and compile + train our face liveness model:

```
72 # construct the training image generator for data augmentation
73 aug = ImageDataGenerator(rotation_range=20, zoom_range=0.15,
74 width_shift_range=0.2, height_shift_range=0.2, shear_range=0.15,
75 horizontal_flip=True, fill_mode="nearest")
76
77 # initialize the optimizer and model
78 print("[INFO] compiling model...")
79 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
80 model = LivenessNet.build(width=32, height=32, depth=3,
81 classes=len(le.classes_))
82 model.compile(loss="binary_crossentropy", optimizer=opt,
83 metrics=["accuracy"])
84
85 # train the network
86 print("[INFO] training network for {} epochs...".format(EPOCHS))
87 H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
88 validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
89 epochs=EPOCHS)
```

Lines 73-75 construct a data augmentation object which will generate images with random rotations, zooms, shifts, shears, and flips. To read more about data augmentation, read **my previous blog post**. Our LivenessNet model is built and compiled on **Lines 79-83**. We then commence training on **Lines 87-89**. This process will be relatively quick considering our shallow network and small dataset. Once the model is trained we can evaluate the results and generate a training plot:

```
91 # evaluate the network
92 print("[INFO] evaluating network...")
93 predictions = model.predict(testX, batch_size=BS)
94 print(classification_report(testY.argmax(axis=1),
95 predictions.argmax(axis=1), target_names=le.classes_))
96
```

```
97 # save the network to disk
98 print("[INFO] serializing network to '{}'...".format(args["model"]))
99 model.save(args["model"])
100
101 # save the label encoder to disk
102 f = open(args["le"], "wb")
103 f.write(pickle.dumps(le))
104 f.close()
105
106 # plot the training loss and accuracy
107 plt.style.use("ggplot")
108 plt.figure()
109 plt.plot(np.arange(0, EPOCHS), H.history["loss"], label="train_loss")
110 plt.plot(np.arange(0, EPOCHS), H.history["val_loss"], label="val_loss")
111 plt.plot(np.arange(0, EPOCHS), H.history["acc"], label="train_acc")
112 plt.plot(np.arange(0, EPOCHS), H.history["val_acc"], label="val_acc")
113 plt.title("Training Loss and Accuracy on Dataset")
114 plt.xlabel("Epoch #")
115 plt.ylabel("Loss/Accuracy")
116 plt.legend(loc="lower left")
117 plt.savefig(args["plot"])
```

Predictions are made on the testing set (**Line 93**). From there a classification_report is generated and printed to the terminal (**Lines 94 and 95**). The LivenessNet model is serialized to disk along with the label encoder on **Lines 99-104**. The remaining **Lines 107-117** generate a training history plot for later inspection.

7.1.5 Training our liveness detector

We are now ready to train our liveness detector. Make sure we've used the ***"Downloads"*** section of the tutorial to download the source code and dataset — from, there execute the following command:

```
liveness Detection with OpenCV
$ python train.py --dataset dataset --model liveness.model --le le.pickle
[INFO] loading images...
[INFO] compiling model...
[INFO] training network for 50 epochs...
Epoch 1/50
29/29 [==============================] - 2s 58ms/step - loss: 1.0113 - acc: 0.5862 -
val_loss: 0.4749 - val_acc: 0.7436
Epoch 2/50
29/29 [==============================] - 1s 21ms/step - loss: 0.9418 - acc: 0.6127 -
val_loss: 0.4436 - val_acc: 0.7949
Epoch 3/50
29/29 [==============================] - 1s 21ms/step - loss: 0.8926 - acc: 0.6472 -
val_loss: 0.3837 - val_acc: 0.8077
...
Epoch 48/50
29/29 [==============================] - 1s 21ms/step - loss: 0.2796 - acc: 0.9094 -
val_loss: 0.0299 - val_acc: 1.0000
Epoch 49/50
29/29 [==============================] - 1s 21ms/step - loss: 0.3733 - acc: 0.8792 -
val_loss: 0.0346 - val_acc: 0.9872
Epoch 50/50
29/29 [==============================] - 1s 21ms/step - loss: 0.2660 - acc: 0.9008 -
val_loss: 0.0322 - val_acc: 0.9872
[INFO] evaluating network...
precision recall f1-score support
fake 0.97 1.00 0.99 35
real 1.00 0.98 0.99 43
micro avg 0.99 0.99 0.99 78
macro avg 0.99 0.99 0.99 78
weighted avg 0.99 0.99 0.99 78
[INFO] serializing network to 'liveness.model'...
```

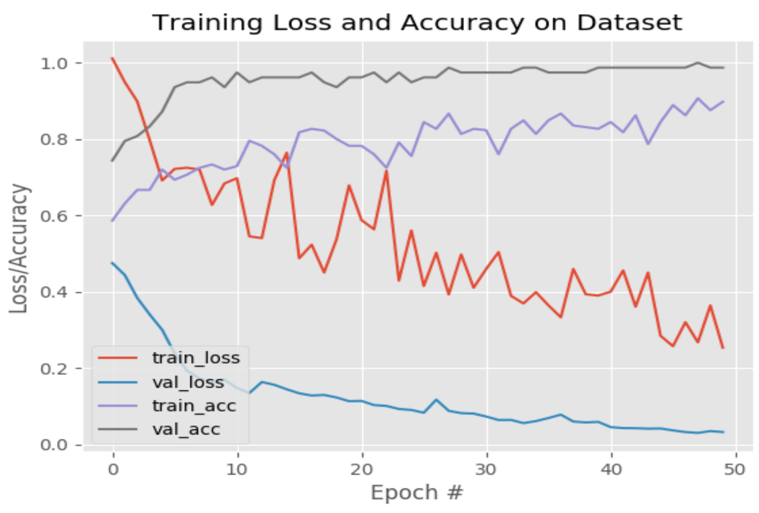


Figure 7.6: A plot of training a face liveness model using OpenCV, Keras, and deep learning. As our results show, we are able to obtain 99% liveness detection accuracy on our validation set!

7.1.6 Putting the pieces together: Liveness detection with OpenCV

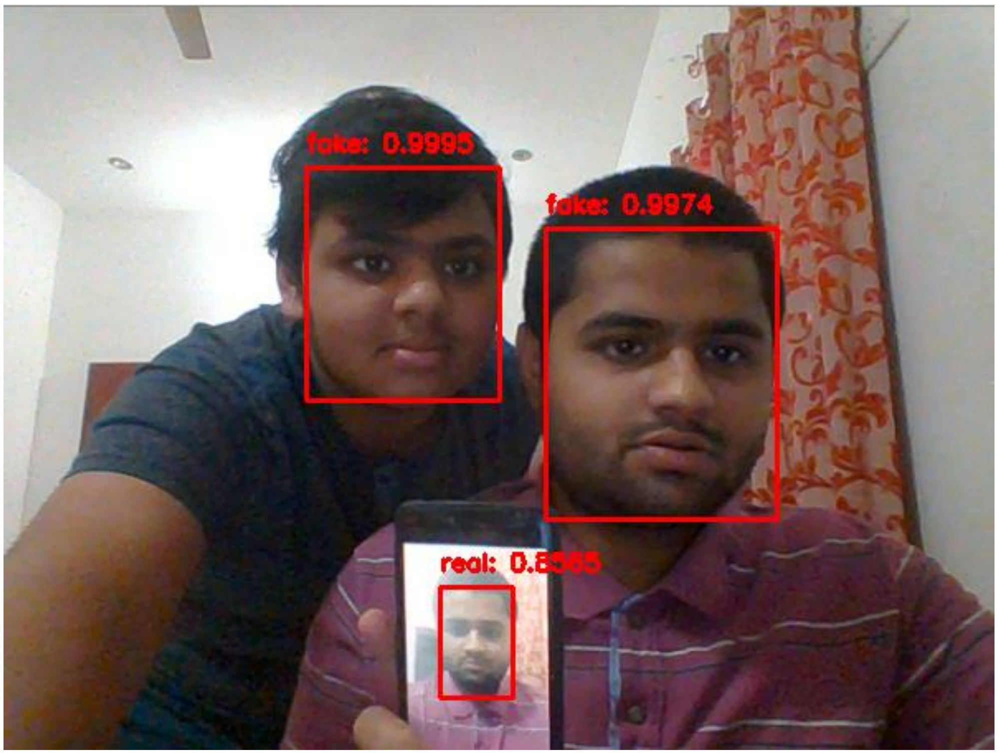


Figure 7.7: Face liveness detection with OpenCV and deep learning.

The final step is to combine all the pieces:

1. We'll access our webcam/video stream
2. Apply face detection to each frame
3. For each face detected, apply our liveness detector model Open up the liveness_demo.py and insert the following code:

```
# import the necessary packages
from imutils.video import VideoStream
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np
import argparse
import imutils
import pickle
import time
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", type=str, required=True,
16 help="path to trained model")
17 ap.add_argument("-l", "--le", type=str, required=True,
18 help="path to label encoder")
19 ap.add_argument("-d", "--detector", type=str, required=True,
20 help="path to OpenCV's deep learning face detector")
21 ap.add_argument("-c", "--confidence", type=float, default=0.5,
22 help="minimum probability to filter weak detections")
23 args = vars(ap.parse_args())
```

Lines 2-11 import our required packages. Notably, we'll use

- VideoStream

  to access our camera feed.

- img_to_array

  so that our frame will be in a compatible array format.

- load_model

  to load our serialized Keras model.

- imutils

  for its convenience functions.

- cv2

  for our OpenCV bindings.

Let's parse our command line arguments via **Lines 14-23:**

- --model

  : The path to our pretrained Keras model for liveness detection.

- --le

  : Our path to the label encoder.

- --detector

  : The path to OpenCV's deep learning face detector, used to find the face ROIs.

- --confidence

  : The minimum probability threshold to filter out weak detections.

Now let's go ahead an initialize the face detector, LivenessNet model + label encoder, and our video stream:

```
25 # load our serialized face detector from disk
26 print("[INFO] loading face detector...")
27 protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
28 modelPath = os.path.sep.join([args["detector"],
29 "res10_300x300_ssd_iter_140000.caffemodel"])
30 net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
31
32 # load the liveness detector model and label encoder from disk
33 print("[INFO] loading liveness detector...")
34 model = load_model(args["model"])
35 le = pickle.loads(open(args["le"], "rb").read())
36
37 # initialize the video stream and allow the camera sensor to warmup
38 print("[INFO] starting video stream...")
39 vs = VideoStream(src=0).start()
40 time.sleep(2.0)
```

The OpenCV face detector is loaded via **Lines 27-30**.From there we load our serialized, pretrained model (LivenessNet) and the label encoder (**Lines 34 and 35**).Our VideoStream object is instantiated and our camera is allowed two seconds to warm up (**Lines 39 and 40**). At this point, it's time to start looping over frames to detect real versus fake/spoofed faces:

```
42 # loop over the frames from the video stream
43 while True:
44 # grab the frame from the threaded video stream and resize it
45 # to have a maximum width of 600 pixels
46 frame = vs.read()
47 frame = imutils.resize(frame, width=600)
48
49 # grab the frame dimensions and convert it to a blob
50 (h, w) = frame.shape[:2]
51 blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
52 (300, 300), (104.0, 177.0, 123.0))
53
54 # pass the blob through the network and obtain the detections and
55 # predictions
56 net.setInput(blob)
57 detections = net.forward()
```

Line 43 opens an infinite whileloop block where we begin by capturing + resizing individual frames (**Lines 46 and 47**).After resizing, dimensions of the frame are grabbed so that we can later perform scaling (**Line 50**)Using OpenCV's `blobFromImage` function we generate a blob (**Lines 51 and 52**) and then proceed to perform inference by passing it through the face detector network (**Lines 56 and 57**).

```
59 # loop over the detections
60 for i in range(0, detections.shape[2]):
61 # extract the confidence (i.e., probability) associated with the
62 # prediction
63 confidence = detections[0, 0, i, 2]
64
65 # filter out weak detections
66 if confidence > args["confidence"]:
67 # compute the (x, y)-coordinates of the bounding box for
68 # the face and extract the face ROI
69 box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
70 (startX, startY, endX, endY) = box.astype("int")
71
72 # ensure the detected bounding box does fall outside the
73 # dimensions of the frame
74 startX = max(0, startX)
75 startY = max(0, startY)
76 endX = min(w, endX)
77 endY = min(h, endY)
78
79 # extract the face ROI and then preproces it in the exact
80 # same manner as our training data
81 face = frame[startY:endY, startX:endX]
82 face = cv2.resize(face, (32, 32))
83 face = face.astype("float") / 255.0
84 face = img_to_array(face)
85 face = np.expand_dims(face, axis=0)
86
87 # pass the face ROI through the trained liveness detector
88 # model to determine if the face is "real" or "fake"
89 preds = model.predict(face)[0]
90 j = np.argmax(preds)
91 label = le.classes_[j]
92
93 # draw the label and bounding box on the frame
94 label = "{}: {:.4f}".format(label, preds[j])
95 cv2.putText(frame, label, (startX, startY - 10),
96 cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
97 cv2.rectangle(frame, (startX, startY), (endX, endY),
98 (0, 0, 255), 2)
```

On **Line 60**, we begin looping over face detections. Inside we:

- Filter out weak detections (**Lines 63-66**).

- Extract the face bounding box coordinates and ensure they do not fall outside the dimensions of the frame (**Lines 69-77**).
- Extract the face ROI and *preprocess* it in the same manner as our training data (**Lines 81-85**).
- Employ our **liveness detector model to determine if the face is *"real"* or *"fake/spoofed"*** (**Lines 89-91**).
- **Line 91 is where we would insert our own code to perform face recognition** but only on real images. The pseudo code would similar to if label == "real": run_face_reconition() *directly after* **Line 91**).
- Finally (for this demo), we draw the label text and a rectangle around the face (**Lines 94-98**).

Let's display our results and clean up:

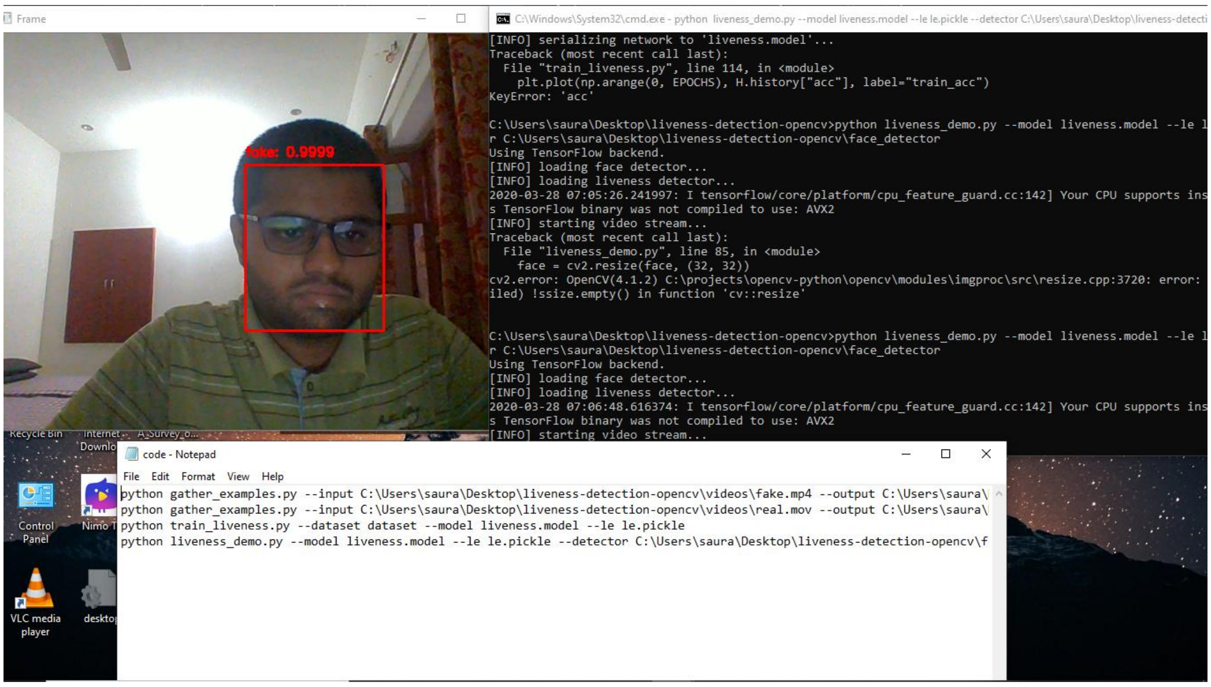
```
100 # show the output frame and wait for a key press
101 cv2.imshow("Frame", frame)
102 key = cv2.waitKey(1) & 0xFF
103
104 # if the `q` key was pressed, break from the loop
105 if key == ord("q"):
106 break
107
108 # do a bit of cleanup
109 cv2.destroyAllWindows()
110 vs.stop()
```

The ouput frame is displayed on each iteration of the loop while keypresses are captured (**Lines 101-102**). Whenever the user presses "q" ("quit") we'll break out of the loop and release pointers and close windows (**Lines 105-110**).

Deploying our liveness detector to real-time video

To follow along with our liveness detection demo make sure we have used the ***"Downloads"*** section of the blog post to download the source code and pre-trained liveness detection model. From there, open up a terminal and execute the following command:

```
liveness Detection with OpenCV
$ python liveness_demo.py --model liveness.model --le le.pickle \
--detector face_detector
Using TensorFlow backend.
[INFO] loading face detector...
[INFO] loading liveness detector...
[INFO] starting video stream...
```


Frame
fake: 0.9999
C:\Windows\System32\cmd.exe - python liveness_demo.py --model liveness.model --le le.pickle --detector C:\Users\saura\Desktop\liveness-detecti
code - Notepad
File Edit Format View Help
Recycle Bin
Internet Downlo
A_Survey_o...
Control Panel
Nimo T
VLC media player
deskto

```
[INFO] serializing network to 'liveness.model'...
Traceback (most recent call last):
  File "train_liveness.py", line 114, in <module>
    plt.plot(np.arange(0, EPOCHS), H.history["acc"], label="train_acc")
KeyError: 'acc'

C:\Users\saura\Desktop\liveness-detection-opencv>python liveness_demo.py --model liveness.model --le l
r C:\Users\saura\Desktop\liveness-detection-opencv\face_detector
Using TensorFlow backend.
[INFO] loading face detector...
[INFO] loading liveness detector...
2020-03-28 07:05:26.241997: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports ins
s TensorFlow binary was not compiled to use: AVX2
[INFO] starting video stream...
Traceback (most recent call last):
  File "liveness_demo.py", line 85, in <module>
    face = cv2.resize(face, (32, 32))
cv2.error: OpenCV(4.1.2) C:\projects\opencv-python\opencv\modules\imgproc\src\resize.cpp:3720: error:
iled) !ssize.empty() in function 'cv::resize'


C:\Users\saura\Desktop\liveness-detection-opencv>python liveness_demo.py --model liveness.model --le l
r C:\Users\saura\Desktop\liveness-detection-opencv\face_detector
Using TensorFlow backend.
[INFO] loading face detector...
[INFO] loading liveness detector...
2020-03-28 07:06:48.616374: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports ins
s TensorFlow binary was not compiled to use: AVX2
[INFO] starting video stream...
```

```
python gather_examples.py --input C:\Users\saura\Desktop\liveness-detection-opencv\videos\fake.mp4 --output C:\Users\saura\
python gather_examples.py --input C:\Users\saura\Desktop\liveness-detection-opencv\videos\real.mov --output C:\Users\saura\
python train_liveness.py --dataset dataset --model liveness.model --le le.pickle
python liveness_demo.py --model liveness.model --le le.pickle --detector C:\Users\saura\Desktop\liveness-detection-opencv\f
```

Fig 7.8 (Detection on Live Camera)

7.2 Conclusion

Using this liveness detector we can now **spot fake fakes and perform anti-face spoofing in our own face recognition systems.**

To create our liveness detector we utilized OpenCV, Deep Learning, and Python.

The first step was to gather our real vs. fake dataset. To accomplish this task, we:

1. First recorded a video of ourselves using our smartphone (i.e., “real” faces).
2. Held our smartphone up to our laptop/desktop, replayed the same video, and then *recorded the replaying* using our webcam (i.e., “fake” faces).
3. Applied face detection to *both sets of videos* to form our final liveness detection dataset.

After building our dataset we implemented, “LivenessNet”, a Keras + Deep Learning CNN.

This network is *purposely* shallow, ensuring that:

1. We reduce the chances of overfitting on our small dataset.
2. The model itself is capable of running in real-time (including on the Raspberry Pi).

Overall, our liveness detector was able to obtain **99% accuracy** on our validation set.

To demonstrate the full liveness detection pipeline in action we created a Python + OpenCV script that loaded our liveness detector and applied it to real-time video streams. As our demo showed, our liveness detector was capable of distinguishing between real and fake faces.

Chapter 8

8. Conclusion and Future Work

8.1 Conclusion

In this part of work, there is a meticulous depiction of the object detection and tracking phenomena in addition to their mechanisms .Object tracking is considered as the first step for tracking which actually serves to locate the temporal position of the object in question .The tracking methods described in this piece of work do not need any pre-locating trend of grabbing the object by object detection still the techniques for detection are studied so far and implemented. Object tracking techniques being implemented for this project do possess their own discrepancies in regard of motion based tracking complications but efforts are made to make the algorithms robust enough to withstand the impediments. In order to provide, vigorous modus operandi for object tracking all latest concepts are used in the implementations .All the techniques are tested on motion detection of air borne non rigid objects as jets to fully examine the heftiness with instantly changing orientations and it cogently showed the vigor of the algorithms implemented during this project phase by withstanding the occlusions to a large extent. At several places tracking snags became a problem like in template matching target loss hinders the algorithm to succeed at certain situations. Purposefully, several other techniques are implemented then which proved to be efficient and effective. So Fast mean shift and Kalman filter are implemented to have a strong savor of object tracking for airborne objects, the objects for which every next instant derives a new position varying widely from the previous one. Thus results prove the effectiveness of the techniques implemented in this project and their accomplishment is also obvious from the results provided in this thesis.

8.2 Future Work

FUTURE ENCHANCEMENTS

The object recognition system can be applied in the area of surveillance system, face recognition, fault detection, character recognition etc. The objective of this thesis is to develop an object recognition system to recognize the 2D and 3D objects in the image. The performance of the object recognition system depends on the features used and the classifier employed for recognition. This research work attempts to propose a novel feature extraction method for extracting global features and and obtaining local features from the region of interest. Also the research work attempts to hybrid the traditional classifiers to recognize the object. The object recognition system developed in this research was tested with the benchmark datasets like COIL100, Caltech 101, ETH80 and MNIST. The object recognition system is implemented in MATLAB 7.5

It is important to mention the difficulties observed during the experimentation of the object recognition system due to several features present in the image. The research work suggests that the image is to be preprocessed and reduced to a size of 128 x 128. The proposed feature extraction method helps to select the important feature. To improve the efficiency of the classifier, the number of features should be less in number. Specifically, the contributions towards this research work are as follows,

1. An object recognition system is developed, that recognizes the two-dimensional and three dimensional objects.
2. The feature extracted is sufficient for recognizing the object and marking the location of the object. X The proposed classifier is able to recognize the object in less computational cost.
3. The proposed global feature extraction requires less time, compared to the traditional feature extraction method.
4. The performance of the SVM-kNN is greater and promising when compared with the BPN and SVM.
5. The performance of the One-against-One classifier is efficient.
6. Global feature extracted from the local parts of the image.
7. Local feature PCA-SIFT is computed from the blobs detected by the Hessian-Laplace detector.
8. Along with the local features, the width and height of the object computed through projection method is used.

The methods presented for feature extraction and recognition are common and can be applied to any application that is relevant to object recognition.

The proposed object recognition method combines the state-of-art classifier SVM and k-NN to recognize the objects in the image. The multiclass SVM is used to hybridize with the k-NN for the recognition. The feature extraction method proposed in this research work is efficient and provides unique information for the classifier.

The image is segmented into 16 parts, from each part the Hu's Moment invariant is computed and it is converted into Eigen component. The local feature of the image is obtained by using the Hessian-Laplace detector. This helps to obtain the objects feature easily and mark the object location without much difficulty.

As a scope for future enhancement,

1. Features either the local or global used for recognition can be increased, to increase the efficiency of the object recognition system.
2. Geometric properties of the image can be included in the feature vector for recognition. 150
3. Using unsupervised classifier instead of a supervised classifier for recognition of the object.
4. The proposed object recognition system uses grey-scale image and discards the color information. The color information in the image can be used for recognition of the object. Color based object recognition plays vital role in Robotics Although the visual tracking algorithm proposed here is robust in many of the conditions, it can be made more robust by eliminating some of the limitations as listed below:

i. In the Single Visual tracking, the size of the template remains fixed for tracking. If the size of the object reduces with the time, the background becomes more dominant than the object being tracked. In this case the object may not be tracked.

ii. Fully occluded object cannot be tracked and considered as a new object in the next frame.

iii. Foreground object extraction depends on the binary segmentation which is carried out by applying threshold techniques. So blob extraction and tracking depends on the threshold value.

iv. Splitting and merging cannot be handled very well in all conditions using the single camera due to the loss of information of a 3D object projection in 2D images.

v. For Night time visual tracking, night vision mode should be available as an inbuilt feature in the CCTV camera.

To make the system fully automatic and also to overcome the above limitations, in future, multi- view tracking can be implemented using multiple cameras. Multi view tracking has the obvious advantage over single view tracking because of wide coverage range with different viewing angles for the objects to be tracked.

In this thesis, an effort has been made to develop an algorithm to provide the base for future applications such as listed below:

In this research work, the object Identification and Visual Tracking has been done through the use of ordinary camera. The concept is well extendable in applications like Intelligent Robots, Automatic Guided Vehicles, Enhancement of Security Systems to detect the suspicious behavior along with detection of weapons, identify the suspicious movements of enemies on boarders with the help of night vision cameras and many such applications. In the proposed method, background subtraction technique has been used that is simple and fast. This technique is applicable where there is no movement of camera. For robotic application or automated vehicle assistance system, due to the movement of camera, backgrounds are continuously changing leading to implementation of some different segmentation techniques like single Gaussian mixture or multiple Gaussian mixture models. Object identification task with motion estimation needs to be fast enough to be implemented for the real time system. Still there is a scope for developing faster algorithms for object identification. Such algorithms can be implemented using FPGA or CPLD for fast execution

An accurate and efficient object detection system has been developed which achieves comparable metrics with the existing state-of-the-art system. This project uses recent techniques in the elder of computer vision and deep learning. Custom dataset was created using labelling and the evaluation was consistent. This can be used in real-time applications which require object detection for pre-processing in their pipeline. An important scope would be to train the system on a video sequence for usage in tracking applications. Addition of a temporally consistent network would enable smooth detection and more optimal than per-frame detection. Object detection is a key ability for most computer and robot vision system. Although great progress has been observed in the last years, and some existing techniques are now part of many consumer electronics (e.g., face detection for auto-focus in smartphones) or have been integrated in assistant driving technologies, we are still far from achieving human-level performance, in particular in terms of open-world learning. It should be noted that object detection has not been used much in many areas where it could be of great help. As mobile robots, and in general autonomous machines, are starting to be more widely deployed (e.g., quad-copters, drones and soon service robots), the need of object detection systems is gaining more importance. Finally, we need to consider that we will need object detection systems for nano-robots or for robots that will explore areas that have not been seen by humans, such as depth parts of the sea or other planets, and the detection systems will have to learn to new object classes as they are encountered. In such cases, a real-time open-world learning ability will be critical.

References

[1] Xiao Wang, "Forward backward correlation for template based tracking" , *Thesis*, Graduate School of Clemson University, (May 2006).

[2] Rafik Bourezak and Guillaume-Alexander Bilodeau, "Object detection and tracking using iterative division and correlograms" , *IEEE 3rd Canadian Conference on Computer and Robotics Vision*, (2006).

[3] Cheung, S.-C. and C. Kamath, "Robust Background Subtraction with Foreground Validation for Urban Traffic Video," *EURASIP Journal on Applied Signal Processing*, vol. 14, pp 1-11,(2005).

[4] Y. Caron, N. Vincent, P. Makris, "Artificial object detection in natural environments" , *RFAI Team Publication, First Annual Meeting on Health, Science and Technology*, Ecole Doctorale SST, Tours (France), (May 2002).

[5]. Alper Yilmaz , Ohio State University , Omar Javed ,Object Video, Inc and Mubarak , Shah University of Central Florida Object Tracking: A Survey ,(2006)

[6] John Melonakos*a* and Yi Gao*a* and Allen Tannenbaum*a*, "Tissue Tracking: Applications for Brain MRI Classification" , Georgia Institute of Technology, 414 Ferst Dr, Atlanta, GA, USA;

[7] KARMA, Computer Graphics and UI Lab, Columbia University, (Jan 2000) URL: http://www.cs.columbia.edu/graphics/projects/karma/karma.html

[8] Feiner, S., MacIntyre, B., Höllerer, T., and Webster, T., "A touring machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment", *In Proc. ISWC '97 (Int. Symposium on Wearable Computers)*, Cambridge MA, (Oct 1997)

[9] A. Nagasaka and Y. Tanaka, "Automatic video indexing and full-video search for object appearances" .In *IFIP Working Conference on Visual Database Systems*, pages 113 127, Budapest,Hungary, (October 1991).

[10] Thuan D. Vong , Background Sub„traction Using Color and Gradient Information ,Department of Electrical a„nd Computer Engineering Clemson,University Clemson, SC 29632,(2004) .

[11] Omar Javed, Khurram Shafique and Mubarak Shah , A Hierarchical” Approach to Robust Background Subtraction using Color and Gradient Information , Computer Vision Lab, School of Electrical Engineering and Computer Science, University of Central Florida ,(2002).

[12] Y. Dedeoglu, Moving object “detection, tracking and classification for smart video surveillance , Bilkent University. ”

[13] K. Fukunaga and L. D. Hostetler, The esti“mation of the gradient of a density function, with applications in pattern recognition , *IE”EE Transactions on Information Theory*, vol.21, pp. 32-40, (1975).

[14] Y. Cheng, Mean shift, mod“e seeking and clustering , *IEEE Tra”nsactions on Pattern Analysis and Machine Intelligence*, vol.17, no.8, pp. 790-799, (1995).

[15] D. Comaniciu and P. Meer, Mean shift: “A robust approach toward feature space analysis , *IEEE Transactions o”n Pattern Analysis and Machine Intelligence*, vol.24, no.5, pp. 603-619, (2002).

[16] C. Belenzai, B. Fruhstuck and H. Bischof, in *Proceedings of IEEE International Conference on Image Processing*, (2004).

[17] P. Y. Simard, L. Bottou, P. Haffner and Y. LeCun, Boxlets: a Fa“st Convolution Algorithm for Signal Processing and Neural Networks , *Advances in Neura”l Information Processing Systems*, vol. 11, pp. 571 577, (1999).

[18] F. Crow, Summed-area tables for texture mapping , in *Proceedings of SIGGRAPH*, vol. 18, pp. 207-212, (1984).

[19] P. Viola and M. Jones, Rapid object “detection using a boosted cascade of simple features , in *IEEE Conference „on Computer Vision and Pattern Recognition*, Kauai, Hawaii, 2001, vol. 1, pp. 511-518.

[20] Eli Brookner, Consultant Scientist, Raytheon Comp. Sudbury, MA, Track“ing and Kalman Filtering Made Easy , John Wiley & S”ons, Inc.

[21] Samuel S. Blackman, Raytheon, Multiple Hypothesis Tracking for Multiple Target Tracking , *IEEE A&E Systems Magazines*, Vol. 19, No.1, (January 2004)

[22] Mikhel E. Hawkins, High Speed Target Tracking Using Kalman Filter and Partial Window Imaging , George Woodruff f School of Mechanical Engineering, Georgia Institute of Technology, (April 2002)

[23] H. L. Van Trees, "Detection, Estimation and Modulation Theory- Part I ", New York, Wiley, (1968).

[24] Erik Cuevas, Daniel Zaldivar and Raul Rojas, "Kalman Filter for Vision Tracking", Technical Report, Freie Universität Berlin, Institut für Informatik, Berlin, Germany