ANOMALY DETECTION IN CROWDED PLACES

Project Submitted in partial fulfillment of the requirements for the

Degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS & COMMUNICATION ENGINEERING

By

AISHWARYA PANDEY (12210102812)
LOKENDER SARNA(17010102812)
GAURAV KUMAR SHARMA(19010102812)

Under the supervision of

MR. M. GANGADHARAPPA

(Associate Professor)

Ambedkar Institute of Advanced Communication

Technologies & Research

(Guru Gobind Singh Indraprastha University)

Geeta colony Delhi-31

DECLARATION

We hereby declare that this submission is our own work and that to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature :

Name & Roll no.

AISHWARYA PANDEY (12210102812)
LOKENDER SARNA(17010102812)
GAURAV KUMAR SHARMA(19010102812)

CERTIFICATE

Certified that AISHWARYA PANDEY (12210102812) ,LOKENDER SARNA (17010102812) & GAURAV KUMAR SHARMA (19010102812) has carried out the project work "ANOMALY DETECTION IN CROWDED PLACES" for the award of Bachelor of Technology from AIACT&R GGSIPU DWARKA, DELHI under my supervision. The project embodies results of original work, and studies as are carried out by the students themselves and the project do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution

Signature:

Mr. M. Gangadharappa
(Associate Professor)
(Dept of Electronics & Communication & Engineering)

Date:

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend my sincere thanks to all of them. We are highly indebted to our project mentor Mr. M. Gangadharappa (Associate professor AIACT&R) and Project coordinator Ms. C.S Vinitha (Asst. professor AIACT&R) for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project. Our thanks and appreciations also go to our classmates in developing the project and people who have willingly helped us out with their abilities

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Abnormal behavior detection in crowd scenes is one of the most important applications in computer vision field. The reason is that abnormal behavior in crowd scenes translates invaluable informative clues for various promising applications, such as intelligent surveillance, safety evaluation, behavior analysis, etc. For crowd structure extraction, this paper proposes using machine learning techniques to intuitively exploit the context clues between individuals, which originally introduces the probability density function to model this relationship . Then a robust multi-object tracker is designed to associate the targets in different frames, which employs the excellent ability of incremental analysis of the newly proposed 3-D discrete cosine transform (DCT). With the obtained targets' association, the black and white images containing only anomaly movement are effectively computed, based on which the abnormality can be online detected.

1.1 TERMINOLOGIES INVOLVED

1.1.1 SYSTEM ANALYSIS

System analysis is a process of gathering and interpreting facts, diagnosing problems and the information to recommend improvements on the system. It is a problem solving activity that requires intensive communication between the system users and system developers. System analysis or study is an important phase of any system development process. The system is studied to the minutest detail and analyzed. The system analyst plays the role of the interrogator and dwells deep into the working of the present system. The system is viewed as a whole and the input to the system are identified. The outputs from the organizations are traced to the various processes. System analysis is concerned with becoming aware of the problem, identifying the relevant and decisional variables, analyzing and synthesizing the various factors and determining an optimal or at least a satisfactory solution or program of action.

1.1.2 SYSTEM REQUIREMENTS

- Windows/Linux operating system
- Intel Pentium Processor [P4] (Minimum)

1.1.3 SOFTWARE AND HARDWARE REQUIREMENTS

- MATLAB
- 512 MB RAM (Minimum), 1 GB RAM (RECOMMENDED)

1.2 TECHNIQUES INVOLVED

1.2.1 IMAGE PROCESSING

Digital Image Processing refers to processing of digital images by means of a digital computer. Digital image can be defined as two dimensional function f(x ,y) and the amplitude of fat any point gives the gray level or intensity of the image. Image Processing is done in three levels: Low level processing involves primitive operations like noise reduction, image sharpening etc. Middle level processing involves partitioning an image into regions or objects. Higher level processing involves "making sense" of an ensemble of recognized objects. Moving object detection comes under mid- level processing.

1.2.2 BACKGROUND ON MATLAB

MATLAB is a high performance language for technical computation whose basic data element is an array that does not require dimensioning. MATLAB stands for matrix laboratory which is a standard computational tool for advanced courses in mathematics, engineering and sciences. MATLAB is a choice for research and analysis as it is complemented by a family of application specific solutions called tool boxes. Image processing tool box helps in digital image processing.

1.2.3 OBJECTIVE

We aim to implement a robust moving object detection algorithm that can detect objects in variety of challenging real world scenarios.

1.2.4 INPUTS GATHERED

Live Camera feed is gathered from the pre-recorded video using videoreader tool of MATLAB. Snapshots of the video are gathered and processed accordingly to detect the red colored object. As the number of frames reaches a set value the camera feed is stopped.

1.2.5 OPERATIONS PERFORMED

- imread():This operation reads an image given as input and stores it in the form of a matrix. The pixels of the image can be read by using array editor.

- rgb2gray (): This function converts color image into a gray scale image. This operation is done because matlab functions are performed only on gray images.
- imsubtract(): This function is used to subtract one image from the other. Both the images should be of same type. Here we use it to isolate the object from the image.
- im2bw(): It is used to convert an image into binary format. It helps in removing the noise around the object. The threshold value of the object can be manipulated to clear the noise to a little large extent.
- Strel(): It is a user defined object or of a defined shape such as diamond, rectangle, etc. It is used in dilating and eroding an object.
- imdilate(): Dilation helps in thickening the object, it combines the disconnected components of the image.
- imdilate(): Dilation helps in thickening the object, it combines the disconnected components of the image.
- imerode(): Erosion helps in thinning the object, it helps in removing the noise on the edges of the object.
- imclose(): It is used to close the open parts of an object to form a single component.
- imfill(): It is used to fill the small holes in the object which have a closed outline.
- bwlabel(): It labels the different components present in the image. This helps us to know the number of objects present.
- graythresh(img): Returns the threshold of the input image.
- bwareaopen(): Helps in removing the small amount of noise in the background according to the given pixel intensity.
- plot(): This function is used plot lines on the frame. We use it here to draw a bounding box around the object.
- getframe(): This function is used to store the current frame generated.

CHAPTER 2: TECHNIQUES INVOLVED

2.1 GAUSSIAN MIXTURE MATRIX

2.1.1 INTRODUCTION

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMMs are commonly used as a parametric model of the probability distribution of continuous measurements or features in a biometric system, such as vocal-tract related spectral features in a speaker recognition system. GMM parameters are estimated from training data using the iterative Expectation-Maximization (EM) algorithm or Maximum A Posteriori (MAP) estimation from a well-trained prior model.

2.1.2 EQUATIONS

A Gaussian mixture model is a weighted sum of M component Gaussian densities as given by the equation,

$$p(\mathbf{x}|\lambda) = \sum_{i=1}^{M} w_i \, g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i),$$

…………………………………………… 2.1

where x is a D-dimensional continuous-valued data vector (i.e. measurement or features), w_i , i = 1, . . . , M, are the mixture weights, and $g(x|\mu_i , \Sigma_i)$, i = 1, . . . , M, are the component Gaussian densities. Each component density is a D-variate Gaussian function of the form

$$g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)' \, \Sigma_i^{-1} \, (\mathbf{x}-\boldsymbol{\mu}_i)\right\}$$

………… 2.2

with mean vector μi and covariance matrix Σi . The mixture weights satisfy the constraint that PM i=1 w_i = 1. The complete Gaussian mixture model is parameterized by the mean vectors, covariance matrices and mixture weights from all component densities. These parameters are collectively represented by the notation .

$$\lambda = \{w_i, \, \boldsymbol{\mu}_i, \, \Sigma_i\} \qquad i = 1, \ldots, M.$$

There are several variants on the GMM shown in above equation. The covariance matrices, Σi , can be full rank or constrained to be diagonal. Additionally, parameters can be shared, or tied, among the Gaussian components, such as having a common covariance matrix for all components, The choice of model configuration (number of components, full or diagonal covariance matrices, and parameter tying) is often determined by the amount of data available for estimating the GMM parameters and how the GMM is used in a particular biometric application. It is also important to note that because the component Gaussian are acting together to model the overall feature density, full covariance matrices are not necessary even if the features are not statistically independent. The linear combination of diagonal covariance basis Gaussians is capable of modeling the correlations between feature vector elements. The effect of using a set of M

full covariance matrix Gaussians can be equally obtained by using a larger set of diagonal covariance Gaussians.
GMMs are often used in biometric systems, most notably in speaker recognition systems, due to their capability of representing a large class of sample distributions. One of the powerful attributes of the GMM is its ability to form smooth approximations to arbitrarily shaped densities. The classical uni-modal Gaussian model represents feature distributions by a position (mean vector) and a elliptic shape (covariance matrix) and a vector quantizer (VQ) or nearest neighbor model represents a distribution by a discrete set of characteristic templates. A GMM acts as a hybrid between these two models by using a discrete set of Gaussian functions, each with their own mean and covariance matrix, to allow a better modeling capability. Figure 2.1 below compares the densities obtained using a unimodal Gaussian model, a GMM and a VQ model .

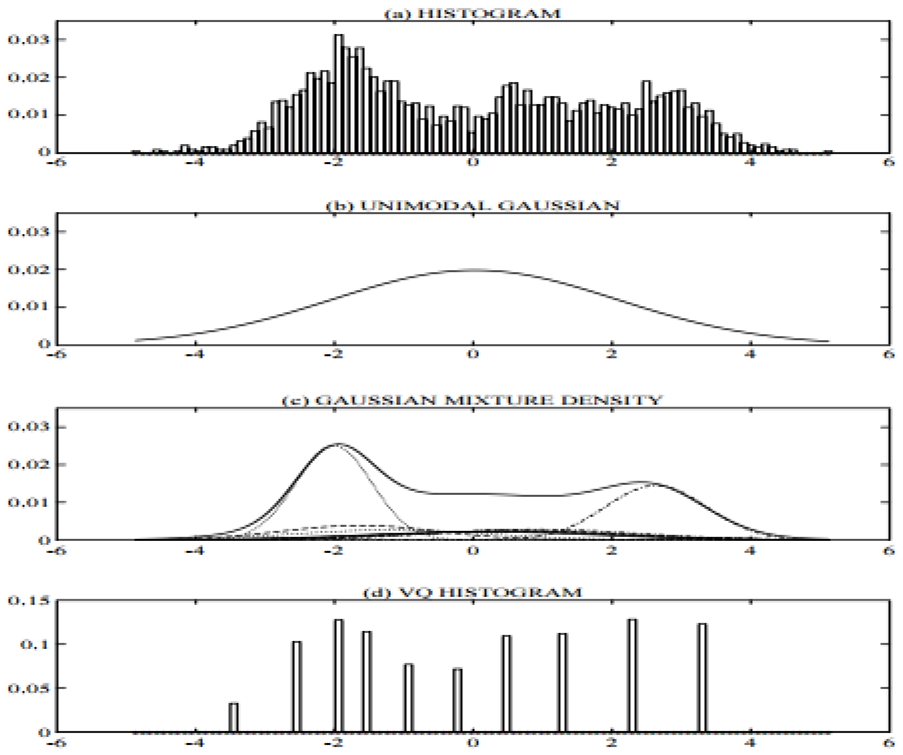


FIGURE 2.1

(a) shows the histogram of a single feature from a speaker recognition system (a single cepstral value from a 25 second utterance by a male speaker)
(b) shows a uni-modal Gaussian model of this feature distribution
(c) shows a GMM and its ten underlying component densities; and plot
(d) shows a histogram of the data assigned to the VQ centroid locations of a 10 element codebook. The GMM not only provides a smooth overall distribution fit, its components also clearly detail the multi-modal nature of the density .

2.2 FOURIER TRANSFORM

The tool which converts a spatial (real space) description of audio/image data into one in terms of its frequency components is called the Fourier transform. The new version is usually referred to as the Fourier space description of the data. We then essentially process the data: E.g. for filtering basically this means attenuating or setting certain frequencies to zero. We then need to convert data back to real audio/imagery to use in our applications.
The corresponding inverse transformation which turns a Fourier space description back into a real space one is called the inverse Fourier transform.

2.2.1 DISCRETE COSINE FOURIER

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio (e.g. MP3) and images (e.g. JPEG) (where small high-frequency components can be discarded), to spectral methods for the numerical solution of partial differential equations. The use of cosine rather than sine functions is critical for compression, since it turns out (as described below) that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions.

In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), where in some variants the input and/or output data are shifted by half a sample. There are eight standard DCT variants, of which four are common.

The most common variant of discrete cosine transform is the type-II DCT, which is often called simply "the DCT". Its inverse, the type-III DCT, is correspondingly often called simply "the inverse DCT" or "the IDCT". Two related transforms are the discrete sine transform (DST), which is equivalent to a DFT of real and odd functions, and the modified discrete cosine transform (MDCT), which is based on a DCT of overlapping data.

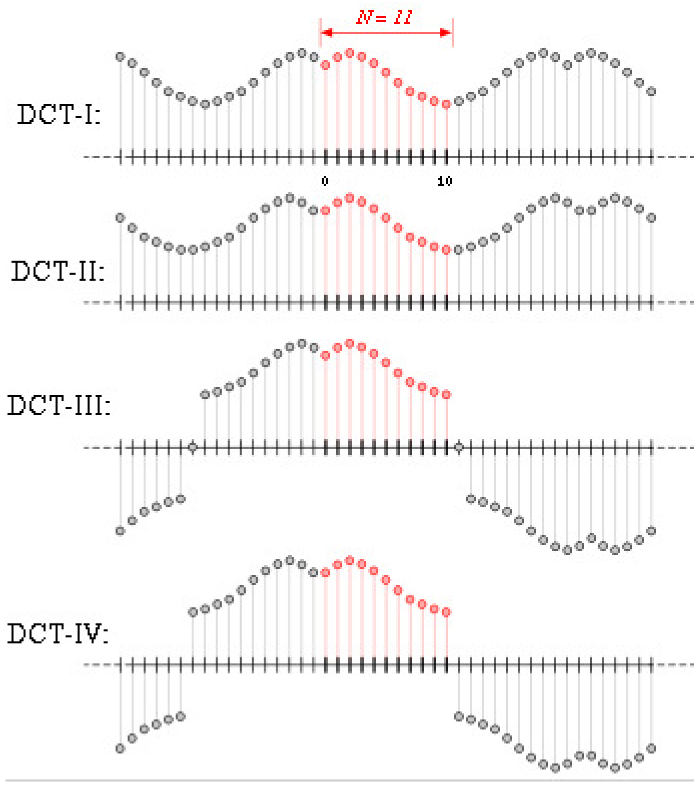


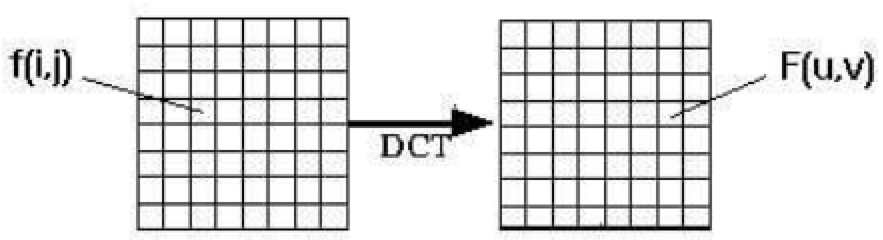
Figure 2.2 Various DCT transforms.

2.2.2 APPLYING THE DCT

Similar to the discrete Fourier transform:

– It transforms a signal or image from the spatial domain to the frequency domain .

– DCT can approximate lines well with fewer coefficients .



Helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality)

2.2.3 DCT EQUATIONS

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad k = 0,\ldots,N-1 \qquad \text{……………..} \quad 2.3$$

The DCT-II is probably the most commonly used form, and is often simply referred to as "the DCT" .
This transform is exactly equivalent (up to an overall scale factor of 2) to a DFT of $4N$ real inputs of even symmetry where the even-indexed elements are zero. That is, it is half of the DFT of the $4N$ inputs y_n, where $y_{2n} = 0$, $y_{2n+1} = x_n$ for $0 \le n < N$, $y_{2N} = 0$, and $y_{4N-n} = y_n$ for $0 < n < 2N$.
Some authors further multiply the X0 term by 1/√2 and multiply the resulting matrix by an overall scale factor of $\sqrt{2/N}$ (see below for the corresponding change in DCT-III). This makes the DCT-II matrix orthogonal, but breaks the direct correspondence with a real-even DFT of half-shifted input. This is the normalization used by Matlab, for example. In many applications, such as JPEG, the scaling is arbitrary because scale factors can be combined with a subsequent computational step (e.g. the quantization step in JPEG[3]), and a scaling that can be chosen that allows the DCT to be computed with fewer multiplications.[4][5]
The DCT-II implies the boundary conditions: xn is even around n=-1/2 and even around n=N-1/2; Xk is even around k=0 and odd around k=N.

2.2.4 INVERSE DCT EQUATIONS

$$X_k = \frac{1}{2}x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N}n\left(k+\frac{1}{2}\right)\right] \qquad k = 0,\ldots,N-1 \qquad \ldots 2.4$$

Because it is the inverse of DCT-II (up to a scale factor, see below), this form is sometimes simply referred to as "the inverse DCT" ("IDCT").[2]
Some authors divide the x0 term by √2 instead of by 2 (resulting in an overall x0/√2 term) and multiply the resulting matrix by an overall scale factor of $\sqrt{2/N}$ (see above for the corresponding change in DCT-II), so that the DCT-II and DCT-III are transposes of one another. This makes the DCT-III matrix orthogonal, but breaks the direct correspondence with a real-even DFT of half-shifted output.
The DCT-III implies the boundary conditions: xn is even around n=0 and odd around n=N; Xk is even around k=-1/2 and odd around k=N-1/2.

2.3 3-D DCT-BASED MULTIPLE OBJECT TRACKER

With the computed SCD for each individual in the crowd, the following work is to detect anomalies by analyzing the spatial-temporal SCD variation. However, the SCD variation between two frames needs to be built on the same target. For this purpose, the multiple targets in different frames should be associated. A straightforward strategy is to design a multi-object tracker. However, although many multi-object trackers are proposed, they are difficult to be implemented in the crowd anomaly detection because of the high density of the crowd, frequent occlusion and appearance/ illumination change. Besides, these multi-object trackers are computationally expensive. Fortunately, we find that the normal/abnormal can be judged alternatively by single individuals in the crowd. Because our purpose is to identify the abnormality, instead of tracking every target, we can fulfill this task by only employing the stable ones called observers. For these individuals, occlusion and appearance/illumination change hardly occur and they can be well tracked. By analyzing the observers' temporal SCD variations, the abnormality can be robustly detected. Therefore, different from the conventional multi-object trackers, this paper designs a new and efficient multi-object tracker to only seek the stable observers. In order to design the robust multi-object tracker, we employ a newly proposed 3-D DCT model [24] with an excellent ability of incremental analysis. The designed 3-D DCT multi-object tracker has three components: compact 3- D DCT template representation, multi-target association, and incremental template updating. Each component is described sequentially as follows.

2.3.1 Compact 3-D DCT-Based Object Representation

For the target association in a video sequence, we adopt to treat the frames as a 3-D volume by concatenating them. Then the self-correlation of the newly observed target sample with the previously collected target sample set is incrementally evaluated. To this end, the 3-D DCT is utilized as a tool to fulfill this task. Given a video sequence, assume the previously collected target sample set can be denoted as *(s***III***(x, y, z))N*1×*N*2×*N*3, where *N*1, *N*2 are the sample's width and height, and *N*3 is the number of samples in the target sample set. The new target sample in the next frame is denoted as *(n(x, y))N*1×*N*2 (abbreviated as *n* for short in the following description). The concatenated target sample set is specified as *(s_***III***(x, y, z))N*1×*N*2×*(N*3+1*)*, where the *(N*3 + 1*)th*frame is concatenated to the end of the previous target sample set. According to the 3-D DCT [24],*(s_***III***(x, y, z))N*1×*N*2×*(N*3+1*)* can be represented as

$$\mathcal{S}' = \mathbf{C}_{\mathbf{III}} \times_1 \mathbf{D}_1^T \times_2 \mathbf{D}_2^T \times_3 (\mathbf{D}')_3^T,$$
$$\mathbf{C}_{\mathbf{III}} = \mathcal{S}' \times_1 \mathbf{D}_1 \times_2 \mathbf{D}_2 \times_3 (\mathbf{D}')_3$$

represents the 3-D DCT coefficient matrix, and ×*m* is the mode-*m* product defined in tensor algebra .

D1 =*(a*1*(o, x))N*1×*N*1 is a cosine basis matrix whose entries are represented as

$$a_1(o, x) = a_1(o) \cos\left(\frac{\pi(2x+1)o}{2N_1}\right) \quad \ldots\ldots\ldots\ldots \quad 2.5$$

D2 = *(a2(p, y))N*2×*N*2 is a similar cosine basis matrix whose entries are specified as

$$a_2(p, y) = a_2(p) \cos\left(\frac{\pi(2y+1)p}{2N_2}\right) \quad \ldots\ldots\ldots \quad 2.6$$

and D'3= *(a_3(q, z))(N*3+1*)*×*(N*3+1*)* is a different cosine basis matrix whose entries are denoted as

$$a'_3(q, z) = \begin{cases} \sqrt{\frac{1}{N_3+1}}, & \text{if } q = 0; \\ \sqrt{\frac{2}{N_3+1}} \cos\left(\frac{\pi(2z+1)q}{2(N_3+1)}\right), & \text{otherwise} \end{cases}$$

where *o* ∈ {0, 1, . . . ,*N*1 − 1}, *p* ∈ {0, 1, . . . ,*N*2 − 1}, *q* ∈{0, 1, . . . ,*N*3 − 1}, and a_k *(o/p/q, x/y/z)* is defined as

$$a_k(o/p/q, x/y/z) = \begin{cases} \sqrt{\frac{1}{N_k}}, & \text{if } o/p/q = 0; \\ \sqrt{\frac{2}{N_k}}, & \text{otherwise.} \end{cases}$$

According to the properties of 3-D DCT, the larger the values (o, p, q) are, the higher frequency the corresponding component of CIII encodes. Depending on the values of (o, p, q), the work [24] compresses CIII by removing the high frequency coefficients usually sparse (e.g., texture clue) and maintaining the low-frequency ones that are relatively dense (e.g., mean value). Therefore, the compact 3-D DCT object representation CIII is modified as
C∗ III= S∗×1 D1 ×2 D2 ×3D_3 (12) where S∗= (s∗ III(x, y, z))N1×N2×(N3+1) is the approximation ofS_, representing the corresponding reconstructed image sequence of S_. Based on it, a reconstruction error representing the loss of low-frequency components is introduced, which is defined as e = ||n −s∗ III (: :,N3 + 1)||2. (13)
For a new target sample, its consistency likelihood with the target sample set can be measured by L = exp_−12λe_ (14)
Where λ is the scaling factor, and is set as 0.1 in this paper. This likelihood measurement is utilized both for target association in Section IV-B and the modeling of incremental template updating described in Section IV-C .

Chapter 3: IMPLEMENTATION AND RESULTS

3.1 BLOCK DIAGRAM

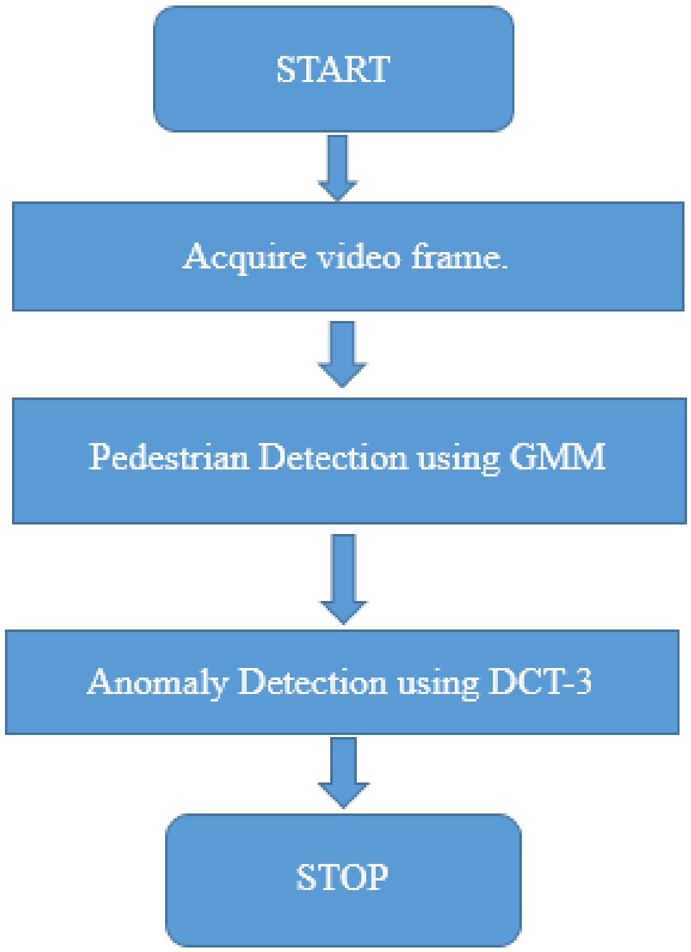


Figure 2.3 Block diagram of the implementation

3.2 METHOD USED

In this paper, an online anomaly crowd detection method is designed, and it is named as online anomaly detection for crowd (OADC). The main steps with a detailed description are as follows

1) **Training Set :** Given a video sequence, in order to generate the target set of each frame, this paper utilizes the state-of-the-art pedestrian detection algorithm . The set of 50 images is used to train the machine using machine learning based Gaussian Mixture Model .

2) **Gaussian Mixture Model :** A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities . GMM parameters are estimated from training data using the iterative Expectation-Maximization (EM) algorithm or Maximum A Posteriori (MAP) estimation from a well-trained prior model .GMM is used to implement machine learning training .

3) **Discrete Cosine Transform:** For the target association, the difficult task is to explicitly model the appearance change. One popular strategy is to learn a low-dimensional subspace, such as the incremental principle component analysis. However, it has high computational complexity. Considering this fact, an alternative object representation based 3-D DCT is utilized to accommodate to the appearance variation. With this representation, each target at different frames is associated by a context-aware multi-object tracker, which paves the way for filtering of low energy functions .

4) **Filtering and Blurring:**Various filtering operations are performed which remove the noise and distractions giving clean feed as output .

5) **Anomaly Detection:** In this step, the anomaly is detected as it is output from above operations . This can be anything other than normal walking which person is supposed to .

3.3 MATLAB CODE FOR IMPLEMENTATION

```
function varargout = ANOMALYGUI(varargin)

gui_Singleton = 1;

gui_State = struct('gui_Name',       mfilename, ...

                   'gui_Singleton',  gui_Singleton, ...

                   'gui_OpeningFcn', @ANOMALYGUI_OpeningFcn, ...

                   'gui_OutputFcn',  @ANOMALYGUI_OutputFcn, ...

                   'gui_LayoutFcn',  [] , ...

                   'gui_Callback',   []);

if nargin && ischar(varargin{1})

    gui_State.gui_Callback = str2func(varargin{1});

end


if nargout

    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});

else

    gui_mainfcn(gui_State, varargin{:});

end

function ANOMALYGUI_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);

function varargout = ANOMALYGUI_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

function pushbutton1_Callback(hObject, eventdata, handles)


myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test002';

 myFolder1=strcat(myFolder,'_gt');

 filePattern = fullfile(myFolder,'*.tif');

 filePattern1 = fullfile(myFolder1,'*.bmp');

jpegFiles = dir(filePattern);

jpegFiles1=dir(filePattern1);
```

```
for k = 1:length(jpegFiles)

  baseFileName = jpegFiles(k).name;

  baseFileName1 = jpegFiles1(k).name;

  fullFileName = fullfile(myFolder, baseFileName);

  fullFileName1 = fullfile(myFolder1, baseFileName1);

  imageArray = imread(fullFileName);

  imageArray1 = imread(fullFileName1);

  axes(handles.axes1);

  imshow(imageArray);

  drawnow;

  axes(handles.axes2);

  s=strel('disk',5);

  imageArray1=dct2(imageArray1);

  imageArray1(abs(imageArray1) < 5) = 0;

  K = idct2(imageArray1);

  K=imopen(K,s);

  imshow(K);

  drawnow;

end

function pushbutton2_Callback(hObject, eventdata, handles)

    videotraffic;



% --- Executes on selection change in popupmenu1.

function popupmenu1_Callback(hObject, eventdata, handles)

% hObject    handle to popupmenu1 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)


% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array

%        contents{get(hObject,'Value')} returns selected item from
popupmenu1
```

```
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test001';
 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);

  imageArray1 = imread(fullFileName1);

  axes(handles.axes1);

  imshow(imageArray);

  drawnow;

  axes(handles.axes2);

  imshow(imageArray1);

  drawnow;

end


% --- Executes on button press in pushbutton4.

function pushbutton4_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton4 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)


myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test003';

 myFolder1=strcat(myFolder,'_gt');

 filePattern = fullfile(myFolder,'*.tif');

 filePattern1 = fullfile(myFolder1,'*.bmp');

jpegFiles = dir(filePattern);

jpegFiles1=dir(filePattern1);

for k = 1:length(jpegFiles)

  baseFileName = jpegFiles(k).name;

  baseFileName1 = jpegFiles1(k).name;

  fullFileName = fullfile(myFolder, baseFileName);

  fullFileName1 = fullfile(myFolder1, baseFileName1);

  imageArray = imread(fullFileName);

  imageArray1 = imread(fullFileName1);

  axes(handles.axes1);

  imshow(imageArray);

  drawnow;
  axes(handles.axes2);

  imshow(imageArray1);

  drawnow;

end


% --- Executes on button press in pushbutton5.

function pushbutton5_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton5 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test004';

 myFolder1=strcat(myFolder,'_gt');

 filePattern = fullfile(myFolder,'*.tif');

 filePattern1 = fullfile(myFolder1,'*.bmp');

jpegFiles = dir(filePattern);

jpegFiles1=dir(filePattern1);

for k = 1:length(jpegFiles)

  baseFileName = jpegFiles(k).name;

  baseFileName1 = jpegFiles1(k).name;

  fullFileName = fullfile(myFolder, baseFileName);

  fullFileName1 = fullfile(myFolder1, baseFileName1);

  imageArray = imread(fullFileName);

  imageArray1 = imread(fullFileName1);

  axes(handles.axes1);

  imshow(imageArray);

  drawnow;

  axes(handles.axes2);

  imshow(imageArray1);

  drawnow;

end
```

```
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test005';
 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);
  imageArray1 = imread(fullFileName1);
  axes(handles.axes1);
  imshow(imageArray);
  drawnow;
  axes(handles.axes2);
  imshow(imageArray1);
  drawnow;
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test006';

 myFolder1=strcat(myFolder,'_gt');

 filePattern = fullfile(myFolder,'*.tif');

 filePattern1 = fullfile(myFolder1,'*.bmp');

jpegFiles = dir(filePattern);

jpegFiles1=dir(filePattern1);

for k = 1:length(jpegFiles)

  baseFileName = jpegFiles(k).name;

  baseFileName1 = jpegFiles1(k).name;

  fullFileName = fullfile(myFolder, baseFileName);

  fullFileName1 = fullfile(myFolder1, baseFileName1);

  imageArray = imread(fullFileName);

  imageArray1 = imread(fullFileName1);

  axes(handles.axes1);

  imshow(imageArray);

  drawnow;

  axes(handles.axes2);

  imshow(imageArray1);

  drawnow;

end


% --- Executes on button press in pushbutton8.

function pushbutton8_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton8 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)



myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test007';

 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);
  imageArray1 = imread(fullFileName1);
  axes(handles.axes1);
  imshow(imageArray);
  drawnow;
  axes(handles.axes2);
  imshow(imageArray1);
  drawnow;
end
% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test008';
 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);
  imageArray1 = imread(fullFileName1);
  axes(handles.axes1);
  imshow(imageArray);
  drawnow;
  axes(handles.axes2);
  imshow(imageArray1);
  drawnow;
end


% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test009';
 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);
  imageArray1 = imread(fullFileName1);
  axes(handles.axes1);
  imshow(imageArray);
  drawnow;
  axes(handles.axes2);
  imshow(imageArray1);
  drawnow;
end


% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test010';
 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);
  imageArray1 = imread(fullFileName1);
  axes(handles.axes1);
  imshow(imageArray);
  drawnow;
  axes(handles.axes2);
  imshow(imageArray1);
  drawnow;
end


% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test011';
 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);
  imageArray1 = imread(fullFileName1);
  axes(handles.axes1);
  imshow(imageArray);
  drawnow;
  axes(handles.axes2);
  imshow(imageArray1);
  drawnow;
end


% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
```

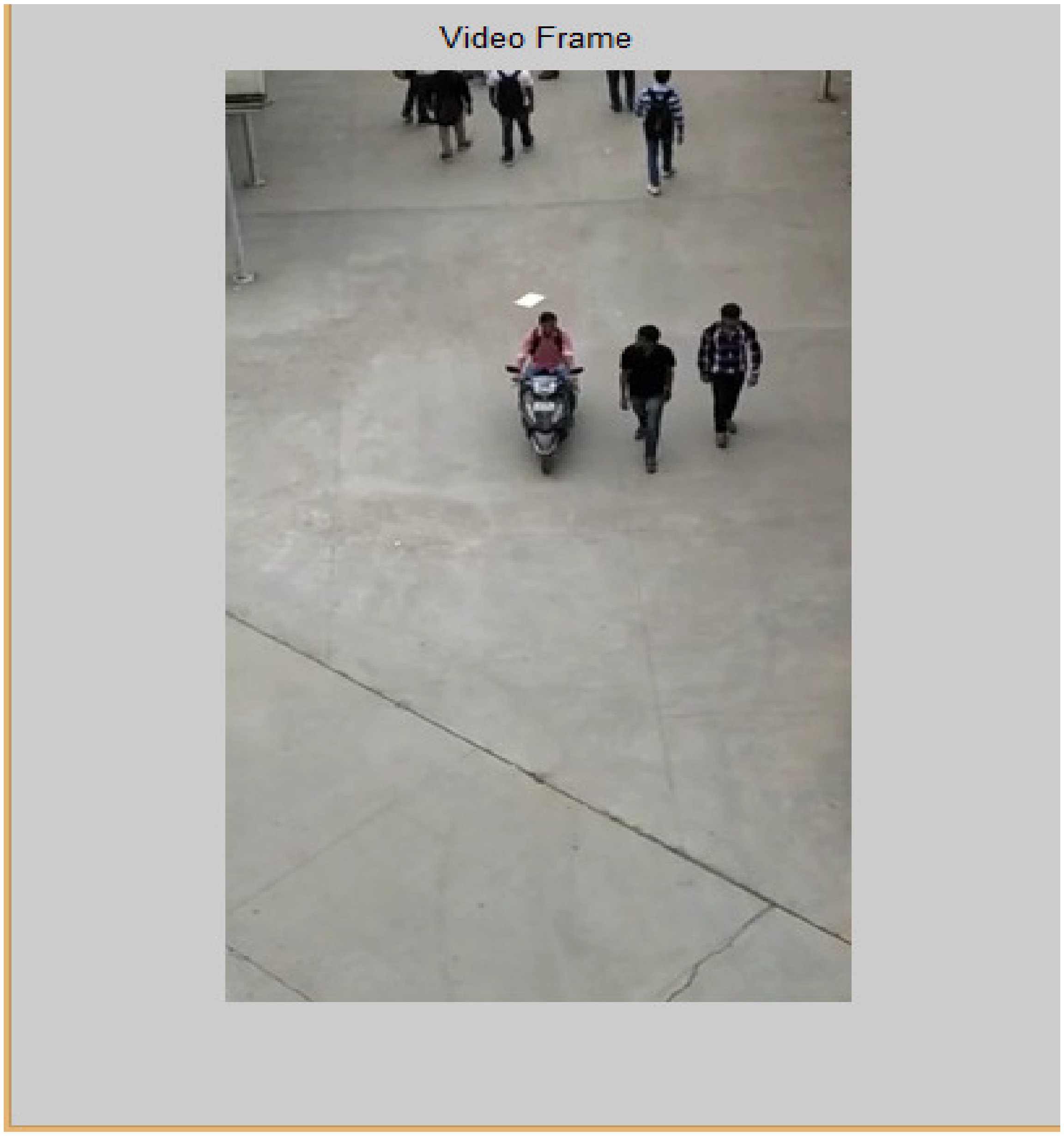
```
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

myFolder='C:\Users\lokender\Downloads\Compressed\UCSD_Anomaly_Dataset.v
1p2\UCSDped2\Test\Test012';
 myFolder1=strcat(myFolder,'_gt');
 filePattern = fullfile(myFolder,'*.tif');
 filePattern1 = fullfile(myFolder1,'*.bmp');
jpegFiles = dir(filePattern);
jpegFiles1=dir(filePattern1);
for k = 1:length(jpegFiles)
  baseFileName = jpegFiles(k).name;
  baseFileName1 = jpegFiles1(k).name;
  fullFileName = fullfile(myFolder, baseFileName);
  fullFileName1 = fullfile(myFolder1, baseFileName1);
  imageArray = imread(fullFileName);
  imageArray1 = imread(fullFileName1);
  axes(handles.axes1);
  imshow(imageArray);
  drawnow;
  axes(handles.axes2);
  imshow(imageArray1);
  drawnow;
end


% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
foregroundDetector = vision.ForegroundDetector('NumGaussians', 3, ...
    'NumTrainingFrames', 50);
```

```
videoReader =
vision.VideoFileReader('C:\Users\lokender\Downloads\Compressed\UCSD_Ano
maly_Dataset.v1p2\real.mp4');

s=strel('disk',5);

for i = 1:350

    frame = step(videoReader);

    frame=imresize(frame,0.2);

    frame=imrotate(frame,270);

    foreground = step(foregroundDetector, frame);

    subplot(1,3,1),imshow(frame); title('Video Frame');

    drawnow;

    imageArray1=imopen(foreground,s);

    p=imageArray1;

    imageArray1=dct2(imageArray1);

    imageArray1(abs(imageArray1)<2) = 0;

    K = idct2(imageArray1);

    subplot(1,3,2), imshow(K); title('Gaussian');

    drawnow;

     subplot(1,3,3), imshow(p); title('Gaussian + DCT');

    drawnow;

end
```

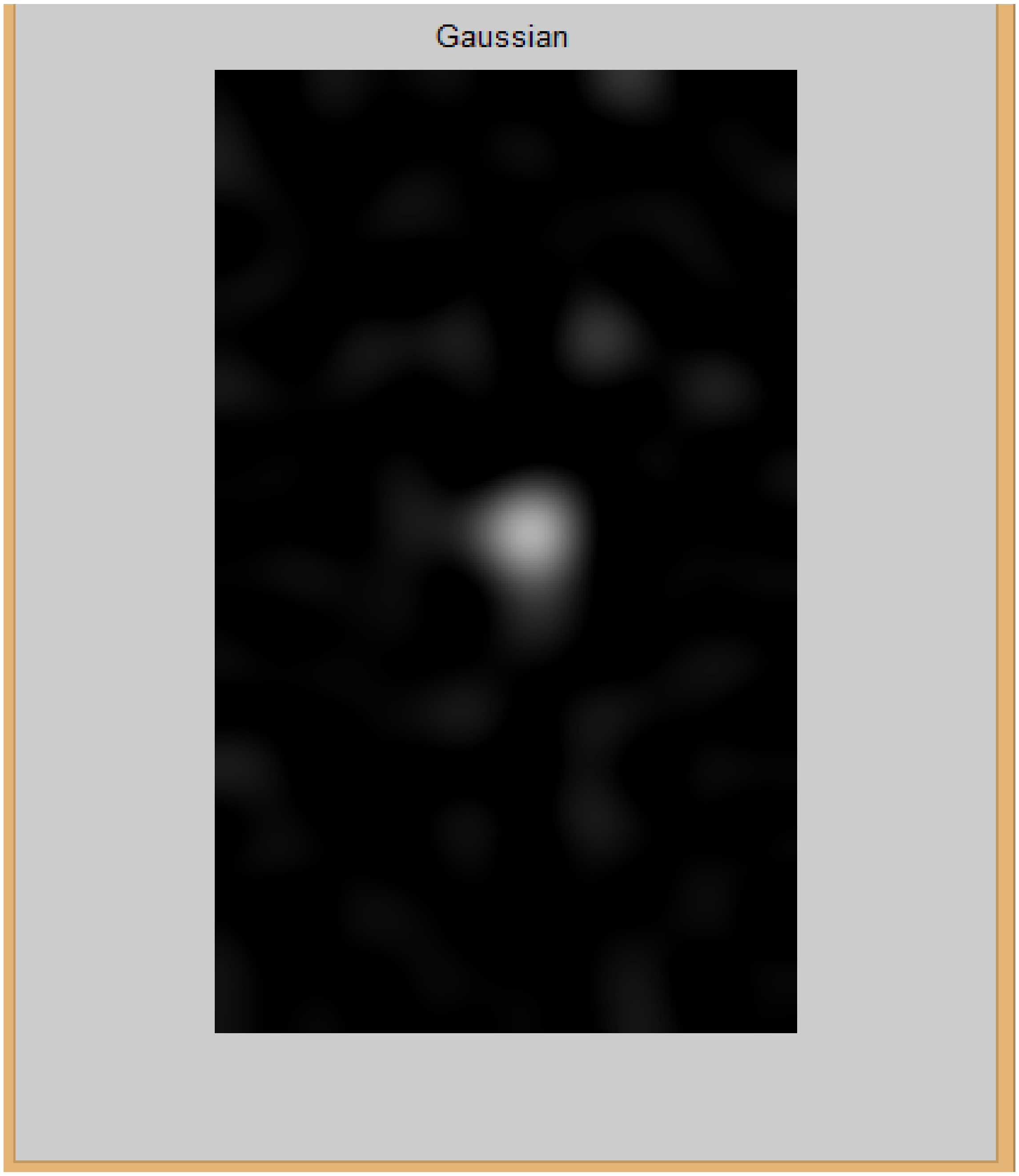
****

3.4 STEP BY STEP WORKING

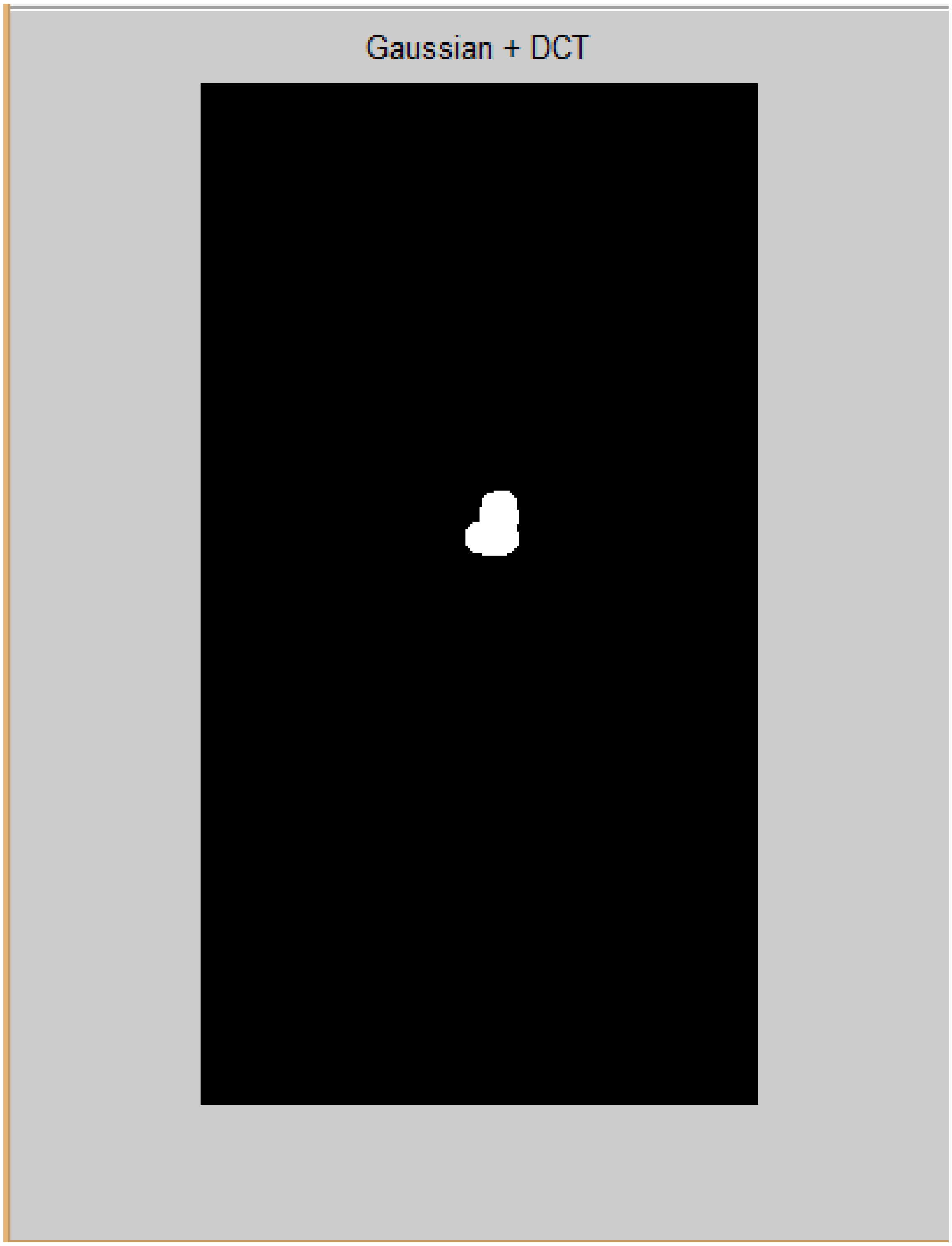
STEP 1: This is the original video frame which is passed to MATLAB software for performing operations .The MATLAB is trained with test sets using GMM which performs machine learning to train The system.



STEP2: After filtering , blurring and applying GMM , the probability density function of various objects with energy is traced .



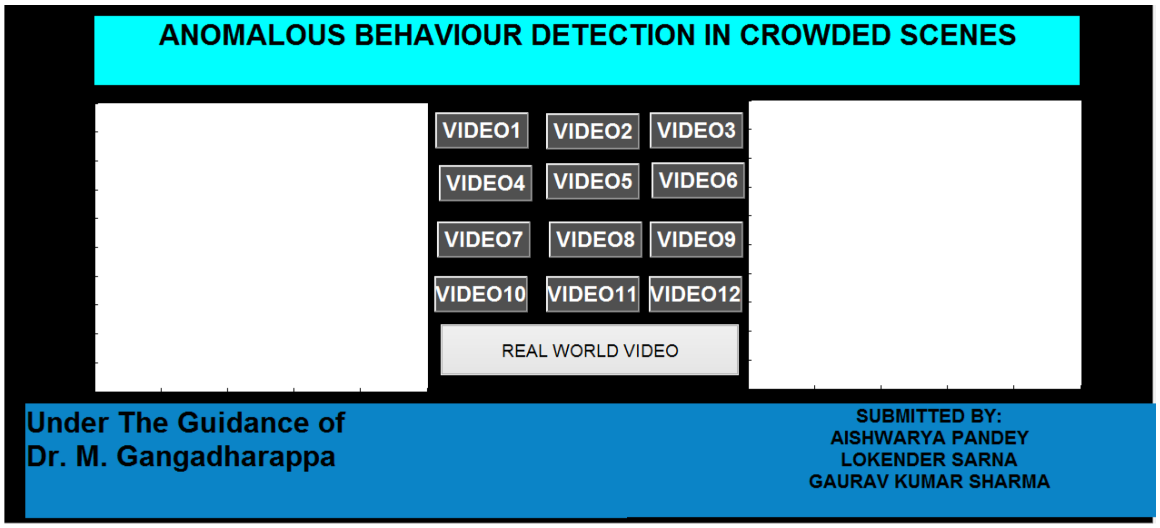
STEP3:Discrete Cosine Transform is used to get the position of objects with energy not matching the energy of objects used in training sets.



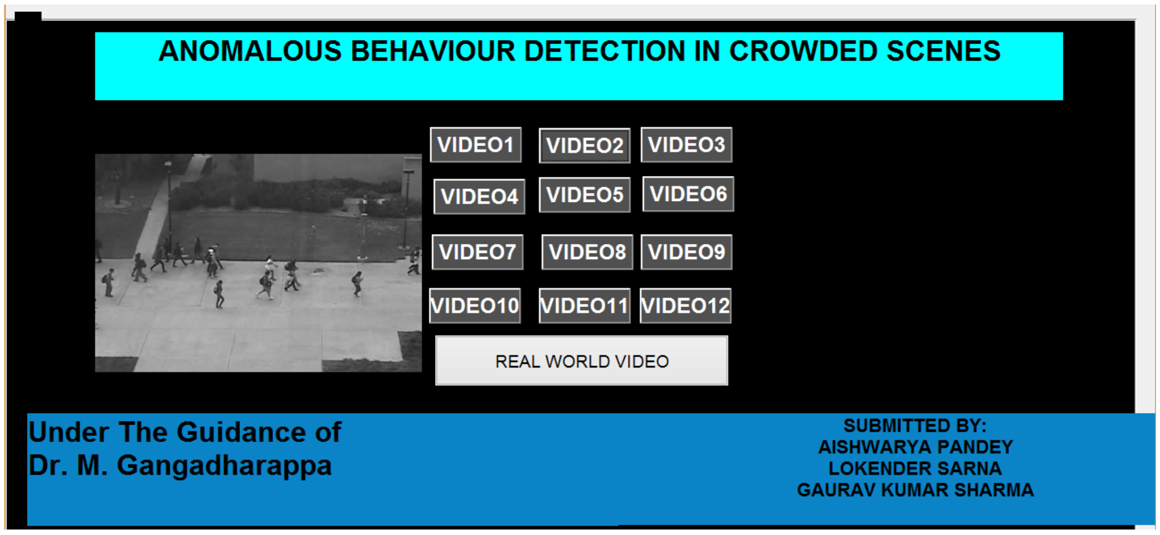
3.5 RESULTS

3.5.1 FOR STANDARD TEST CASES

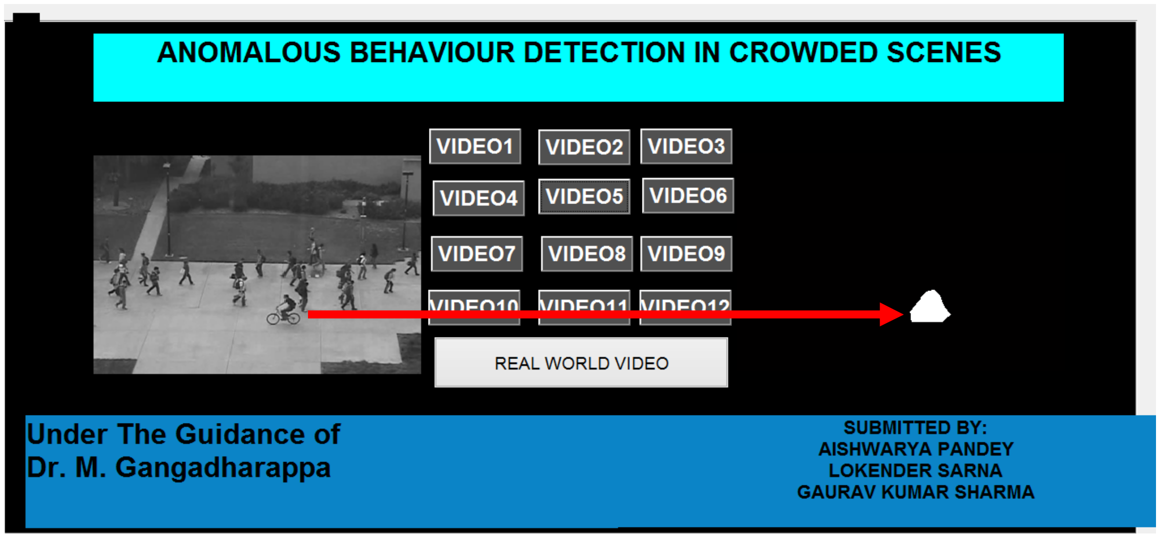
The Graphical User Interface contains the links to various test videos as well as Real World Videos which can be added by the user to perform anomaly detection on them.



GUI WHEN NO INPUT SELECTED



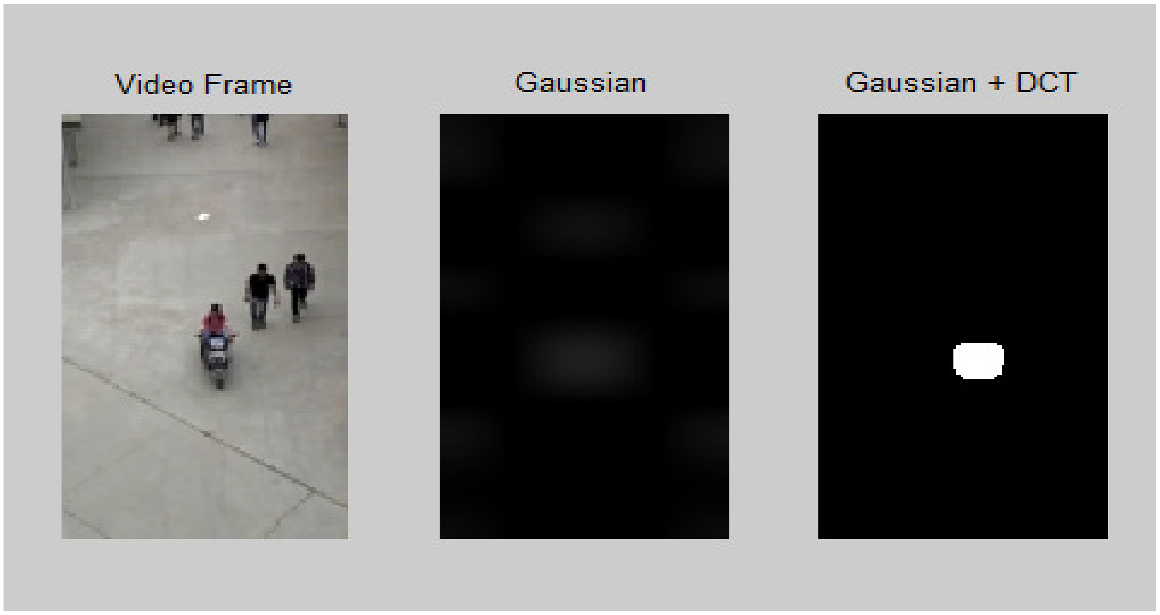
GUI WHEN NO ANOMALY DETECTED IN FRAME



GUI WHEN ANOMALY IS DETECTED

3.5.2 REAL WORLD RESULTS:

This is the result of video on real time objects as the image shown is of college parking lot and the motor vehicle is detected as anomalous on pedestrian walk-way which is treated like an anomaly.



CHAPTER 4:CONCLUSION

This report proposes an anomaly detection method from crowded environments. The visual structural context of the individuals is for the first time explored in this field. For this purpose, we originally introduce the Gaussian Probability Density to construct the GMM, which can effectively represent the relationship among individuals based on learning from data sets. In order to compute the variation efficiently, a robust multi-object tracker is then designed to associate the targets in different frames. The proposed tracker introduces the excellent incremental ability of 3-D DCT and only limited number of targets need to be stably tracked. This makes the tracking method feasible for anomaly detection in crowd scenes with high density. By this analysis of the variation, the crowd abnormality is detected in the end. From the testing results on several popular datasets, the proposed method is superior to others representing the state-of-the-arts. Our paper is tested only in the visible video sequences containing RGB channels. However, in severe weather conditions like foggy and rainy days, it takes difficulties to the proposed method, as well as the other competitive ones. But still the algorithm is feasible in Real Time systems generating results on real feeds and giving reliable results.

The results when tested on Real World scenario, gave the following results in terms of speed and accuracy:

Video no.	Frames detected with anomaly	Actual anomalous frames	Accuracy
1	88	120	74%
2	81	120	67%
3	83	119	70%

Table 4.1 Accuracy of our system

Video no.	Time duration	Processing time
1	4.9 s	9.5540s
2	5.0s	9.7004s
3	4.8s	9.8323s

Table 4.2 Real Time Performance of System

BIBLIOGRAPHY AND REFERENCES

[1] A. A. Sodemann, M. P. Ross, and B. J. Borghetti, “A review of anomaly detection in automated surveillance,” IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 42, no. 6, pp. 1257–1272, Nov. 2012.
[2] T. Gandhi and M. M. Trivedi, “Pedestrian protection systems: Issues, survey, and challenges,” IEEE Trans. Intell. Transp. Syst., vol. 8, no. 3, pp. 413–430, Sep. 2007.
[3] H. Zhou and H. Hu, “Human motion tracking for rehabilitation—A survey,” Biomed. Signal Process. Control, vol. 3, no. 1, pp. 1–18, 2008.
[4] L. Shao, L. Ji, Y. Liu, and J. Zhang, “Human action segmentation and recognition via motion and shape analysis,” Pattern Recognit. Lett., vol. 33, no. 4, pp. 438–445, 2012.