**CHAPTER 1**

**INTRODUCTION**

Real Time Intruder Motion Detection and Recording in the Region of Interest. Continuous-scene monitoring applications, such as ATM booths, parking lots or traffic monitoring systems, generate large volumes of data. Recording and archiving such volumes of data is a real problem, and one way to solve this is to reduce the size of the data stream right at the source. In addition to traditional methods for compressing individual video images, we could identify and record only “interesting” video images, such as those images with significant amounts of motion in the field of view. That could significantly help reduce the data rates for surveillance-specific applications. To extract the maximum benefit from this recorded digital data, detect any moving object from the scene is needed without engaging any human eye to monitor things all the time. Real-time segmentation of moving regions in image sequences is a fundamental step in many vision systems. The system will be able to identify and record only “interesting” video frames containing motion in the ROI. A typical method is background subtraction. Many background models have been introduced to deal with different problems. One of the successful solutions to these problems is to use a multi-color background model per pixel proposed by Grimson et al [1, 2].

**1.1 OBJECTIVES:**

Aim: The main goal of this project is to develop a system which is capable to detect and track moving object from the frames captured by a web-camera and can not only detect motion, but will

a) Warn the user of the intrusion and

b) Record the footage of the video from the moment the motion was detected.

**1.2 PROJECT SCOPES:**

In general, there are two main components in any motion detection system. One is the scene, more often called the world, and the other is the sensor, which, in most cases, is represented by the camera. For analyzing motion detection, it is imperative to understand the configuration between the camera and the world, as each one of them is treated in a slightly different manner. There are 3 possible configurations [3].

1. Stationary camera, moving objects (SCMO)

2. Moving camera, stationary objects (MCSO)

3. Moving camera, moving objects (MCMO)

In this project, the research is based on Stationary camera, moving objects (MCMO) system. The proposed detection and tracking algorithm is mainly concentrated on robustly detecting the movement of a single moving object from an image sequence obtained from the camera. Nevertheless, the algorithm has a capability to detect multiple moving objects but with some constraints. Each frame of the captured image sequence is fixed to a size of 256 x 256 pixels. The processing of the acquired image sequence is performed in the binary format, which has 2 values; ‘0’ and ‘1’. The entire moving object detection and tracking program will be implemented by using Mat lab environment and real time video processing .Camera motion is strictly translation and not rotation or pan-tilt.

* Fine tuning of movement and intruder detection by comparing two consecutive frames.
* Alarm raising system inclusion.
* Store the images and video only when there is a movement detected.
* The pixel values of each frame is compared with the pixel values of another frame and if it rises the threshold values(set by User) then only the alarm should rise. Alarm may be set to any desired tone and frequency (of user’s choice) then storage of the movement of object is carried out, storing the media in video format is normal AVI format of video.
* Camera to capture the real time video is to be done and the algorithm for optimum placement is to be developed by defining the fix coordinates.

**1.3 PROBLEM STATEMENTS:**

1. Implementation of SAD motion detection algorithm in MAT LAB for real time with video.
2. Inclusion of Alarm for safety.
3. Recording Image in the Hard disk in video AVI format.
4. Develop an algorithm to work ROI area only to be detected if motion is there.

**1.4 MOTION DETECTION:**

Motion detection in consequent images is nothing but the detection of the moving object in the scene. In video surveillance, motion detection refers to the capability of the surveillance system to detect motion and capture the events. Motion detection is usually a software-based monitoring algorithm which will signal the surveillance camera to begin capturing the event when it detects motions. This is also called activity detection. An advanced motion detection surveillance system can analyze the type of motion to see if it warrants an alarm. In this project, a camera fixed to its base has been placed and is set as an observer at the outdoor for surveillance. Any small movement with a level of tolerance it picks is detected as motion.

Aside from the intrinsic usefulness of being able to segment video streams into moving and background components, detecting moving blobs provides a focus of attention for recognition, classification, and activity analysis, making these later processes more efficient since only “moving” pixels need be considered. There are three conventional approaches to moving object detection [3]: temporal differencing, background subtraction and optical flow. Temporal differencing is very adaptive to dynamic environments, but generally does a poor job of extracting all relevant feature pixels. Background subtraction provides the most complete feature data, but is extremely sensitive to dynamic scene changes due to lighting and extraneous events. Optical flow can be used to detect independently moving objects in the presence of camera motion; however, most optical flow computation methods are computationally complex, and cannot be applied to full-frame video streams in real-time without specialized hardware [3].

**1.5 VIDEO SURVEILLANCE:**

An appliance that enables embedded image capture capabilities that allows video images or extracted information to be compressed, stored or transmitted over communication networks or digital data link. Digital video surveillance systems are used for any type of monitoring. Broadly, video surveillance is the image sequences which are recorded to monitor the live activities of a particular scene. The importance of this digital evidence is given the first priority for any kind of occurrence. This digital information is recently become the field of interest to the researchers on the field of AI, Robotics, Forensic Science and other major fields of science.

**1.6 APPLICATION OF MOTION DETECTION:**

Detecting and tracking a moving object in a dynamic video sequence has been a vital aspect of motion analysis. This detecting and tracking system has become increasingly important due to its application in various areas, including communication (videoconferencing), transportation (traffic monitoring and autonomous driving vehicle), security (premise surveillance) and industries (dynamic robot vision and navigation).

**CHAPTER 2:**

**THEORETICAL BACKGROUND OF MOTION DETECTION**

**2.1 INTRODUCTION**

A static camera observing a scene is a common case of a surveillance system. Detecting intruding objects is an essential step in analyzing the scene. A usually applicable assumption is that the images of the scene without the intruding objects exhibit some regular behaviour that can be well described by a statistical model [3]. If a statistical model of the scene has been revealed, an intruding object can be detected by spotting the parts of the image that don’t fit the model. This process is usually known as “background subtraction”. Usually a simple bottom-up approach is applied and the scene model has a probability density function for each pixel separately [4]. A pixel from a new image is considered to be a background pixel if its new value is well described by its density function. For example, for a static scene the simplest model could be just an image of the scene without the intruding objects. The next step would be, for example, to estimate appropriate values for the variances of the pixel intensity levels from the image since the variances can vary from pixel to pixel. However, pixel values often have complex distributions and more elaborate models are needed [4]. The scene could change from time to time (sudden or slow illumination changes, static objects removed etc.) [4].The model should be constantly updated to reflect the most current situation. The major problem for the background subtraction algorithms is how to automatically and efficiently update the model. Based on the extracted principles, analyzed and compared two efficient algorithms for the two models: Gaussian mixture and static background estimation. The Gaussian mixture density function is a popular flexible probabilistic model. A Gaussian mixture was proposed for background subtraction in various studies. One of the most commonly used approaches for updating the Gaussian mixture model. A Gaussian mixture having a fixed number of components is constantly updated using a set of heuristic equations. Based on some additional approximations a set of theoretically supported but still very simple equations for updating the parameters of the Gaussian mixture is seen [4]. The important improvement compared to the previous approaches is that at almost no additional cost also the number of components of the mixture is constantly adapted for each pixel. By choosing the number of components for each pixel in an on-line procedure, the algorithm can automatically fully adapt to the scene. Secondly, the simplest form of the reference image is a time-averaged background image. This method suffers from many problems and requires a training period absent of foreground objects. The motion of background objects after the training period and foreground objects motionless during the training period would be considered as permanent foreground objects. In addition, the approach cannot cope with gradual illumination changes in the scene. These problems lead to the requirement that any solution must constantly re-estimate the background model. Many adaptive background modelling methods have been proposed to deal with these slowly-changing stationary signals. Friedman and Russell modelled [5] each pixel in a camera scene by an adaptive parametric mixture model of three Gaussian distributions. The methods can cope well with the illumination changes; however, cannot handle the problem of objects being introduced or removed from the scene. One solution is to use a multiple-colour background model per pixel. Grimson et al [1, 2]. model can also lessen the effect of small repetitive motions; for example, moving vegetation like trees and bushes as well as small camera displacement. In addition to Grimson et al. [1, 2], many other authors have applied mixture models to model every pixel in camera scenes. Rowe and Blake applied the batch EM algorithm for off-line training in their virtual image plane. However, the model does not update with time and therefore leads to failure for external environments where the scene lighting changes with time. Although a number of speed-up routines is presented, the approach was still of high computational complexity.

**2.2 GOALS OF OBJECT DETECTION:**

The goals of the object tracking stage are to:

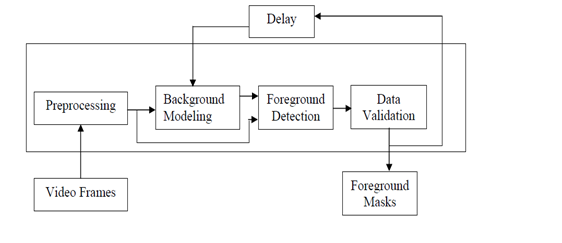
* Determine when a new object enters the system's field of view, and initialize motion models for tracking that object.
* Compute the correspondence between the foreground regions detected by the background subtraction and the objects currently being tracked [6].
* Employ tracking algorithms to estimate the position of each object, and update the motion model used for tracking. The target is to model the overall motion of an object.

**2.3 BACKGROUND SUBTRACTION:**

As computer vision begins to address the visual interpretation of action applications such as surveillance and monitoring are becoming more relevant. Similarly, recent work in intelligent environments and perceptual user interfaces involve vision systems which interpret the pose or gesture of users in a known, indoor environment. In all of these situations the first fundamental problem encountered is the extraction of the image region corresponding to the object or persons in the room. Previous attempts at segmenting object from a known background have taken one of the three approaches mentioned previously. Most common is some form of background subtraction. For example, Grimson et al[1,2] uses statistical texture properties of the background observed over extended period of time to construct a model of the background, and use this model to decide which pixels in an input image do not fall into the background class. The fundamental assumption of the algorithm is that the background is static in all respects: geometry, reflectance, and illumination. The second class of approach is based upon image motion only presuming that the background is stationary or at most slowly varying, but that the object is moving. In these methods no detailed model of the background is required. Of course, these methods are only appropriate for the direct interpretation of motion; if the object stops moving, no signal remains to be processed. This method also requires constant or slowly varying geometry, reflectance, and illumination. The final approach, and the one most related to the technique presented is based upon geometry. Kanade, et al [3]. employ special purpose multi-baseline stereo hardware to compute dense depth maps in real-time. Provided with a background disparity value, the algorithm can perform real-time depth segmentation or ``z-keying’’. The only assumption of the algorithm is that the geometry of the background does not vary. However, the computational burden of computing dense, robust, real-time stereo maps requires great computational power.

**2.4 METHOD ILLUSTRATION:**

Even though there exist a myriad of background subtraction algorithms in the literature [7], most of them follow a simple flow diagram shown in Figure 2.1.

****

**Fig. 2.1Fow diagram of generic background subtraction algorithm**

The four major steps in a background subtraction algorithm are pre-processing, background modelling, foreground detection, and data validation. Pre-processing consists of a collection of simple image processing tasks that change the raw input video into a format that can be processed by subsequent steps. Background modelling uses the new video frame to calculate and update a background model. This background model provides a statistical description of the entire background scene. Foreground detection then identifies pixels in the video frame that cannot be adequately explained by the background model and outputs them as a binary candidate foreground mask. Finally, data validation examines the candidate mask, eliminates those pixels that do not correspond to actual moving objects, and outputs the final foreground mask. Domain knowledge and computationally-intensive vision algorithms are often used in data validation. Real-time processing is still feasible as these sophisticated algorithms are applied only on the small number of candidate foreground pixels. Many different approaches have been proposed for each of the four processing steps. Some of the representative ones in the following subsections have been reviewed.

**2.4.1 PRE-PROCESSING:**

In most computer vision systems, simple temporal and/or spatial smoothing is used in the early stage of processing to reduce camera noise. Smoothing can also be used to remove transient environmental noise such as rain and snow captured in camera. For real-time systems, frame-size and frame-rate reduction are commonly used to reduce the data processing rate. If the multiple cameras are used at different locations, image registration between successive frames or among different cameras is needed before background modelling. Another key issue in pre-processing is the data format used by the particular background subtraction algorithm. Most of the algorithms handle luminance intensity, which is one scalar value per each pixel. However, colour image, in either RGB or HSV colour space, is becoming more popular in the background subtraction literature. Some algorithms argue that colour is better than luminance at identifying objects in low- contrast areas and suppressing shadow cast by moving objects. In addition to colour, pixel-based image features such as spatial and temporal derivatives are sometimes used to incorporate edges and motion information. For example, intensity values and spatial derivatives can be combined to form a single state space for background tracking with the Kalman filter. Pless et al [8]. combine both spatial and temporal derivatives to form a constant velocity background model for detecting speeding vehicles. The main drawback of adding colour or derived features in background modelling is the extra complexity for model parameter estimation. The increase in complexity is often significant as most background modelling techniques maintain an independent model for each pixel.

**2.4.2 RECURSIVE TECHNIQUES:**

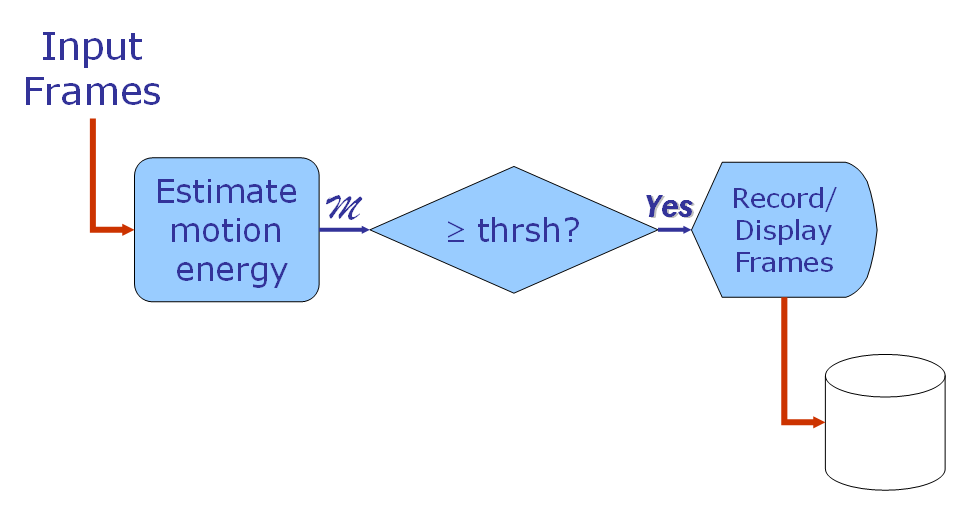
To reduce computation burden and speed up the process recursive techniques are introduced in the process. Recursive techniques do not maintain a buffer for background estimation. Instead, they recursively update a single background model based on each input frame [7]. As a result, input frames from distant past could have an effect on the current background model. Compared with no recursive techniques, recursive techniques require less storage, but any error in the background model can linger for a much longer period of time. Most schemes include exponential weighting to discount the past, and incorporate positive decision feedback to use only background pixels for updating. Approximated median filter Due to the success of non-recursive median filtering, McFarlane and Schofield propose a simple recursive filter to estimate the median. This technique has also been used in background modelling for urban traffic monitoring. In this scheme, the running estimate of the median is incremented by one if the input pixel is larger than the estimate, and decreased by one if smaller. This estimate eventually converges to a value for which half of the input pixels are larger than and half are smaller than this value, that is, the median.

**2.4.3 FOREGROUND DETECTION:**

Foreground detection compares the input video frame with the background model, and identifies candidate foreground pixels from the input frame. Except for the non-parametric model and the MoG (Mixture of Gaussians) model, almost all the techniques use a single image as their background models [7]. Another approach to introduce spatial variability is to use two thresholds with hysteresis. The basic idea is to first identify "strong" foreground pixels whose absolute differences with the background estimates exceeded a large threshold. Then, foreground regions are grown from strong foreground pixels by including neighbouring pixels with absolute differences larger than a smaller threshold. The region growing can be performed by using a two-pass, connected-component grouping algorithm.

**2.4.4 DATA VALIDATION:**

Data validation is defined as the process of improving the candidate foreground mask based on information obtained from outside the background model. All the background models discussed so far have three main limitations: first, they ignore any correlation between neighbouring pixels; second, the rate of adaption may not match the moving speed of the foreground objects; and third, non-stationary pixels from moving leaves or shadow cast by moving objects are easily mistaken as true foreground objects. The first problem typically results in small false-positive or false-negative regions distributed randomly across the candidate mask. The most common approach is to combine morphological filtering and connected component grouping to eliminate these regions. Applying morphological filtering on foreground masks eliminates isolated foreground pixels and merges nearby disconnected foreground regions [7]. Many applications assume that all moving objects of interest must be larger than a certain size which is partly wrong. Connected-component grouping can then be used to identify all connected foreground regions, and eliminates those that are too small to correspond to real moving objects. When the background model adapts at a slower rate than the foreground scene, large areas of false foreground, commonly known as "ghosts", often occur [7]. If the background model adapts too fast, it will fail to identify the portion of a foreground object that has corrupted the background model. A simple approach to alleviate these problems is to use multiple background models running at different adaptation rates, and periodically cross-validate between different models to improve performance [7]. Sophisticated vision techniques can also be used to validate foreground detection. If multiple cameras are available to capture the same scene at different angles, disparity information between Cameras can be used to estimate depth. Depth information is useful as foreground objects are closer to the camera than background. But this is not the case for this work. The moving-leaves problem can be addressed by using sophisticated background modelling techniques like MoG (Mixture of Gaussians) and applying morphological filtering for cleanup. On the other hand, suppressing moving shadow is much more problematic, especially for luminance-only video.



**Figure2.3: The Motion Detection Process**

**CHAPTER 3:**

**ALGORITHMS**

This chapter presents the main software design and implementation issues. It starts by describing the algorithm used in code and general flow chart of the main program that was implemented in MATLAB. It then explains each component of the flow chart with some details.

**3.1 ALGORITHM USED IN CODE**

We have used 2 inputs to detect the motion in the particular environment those 2 inputs are :

* **Video input from Camera**
* **Audio input from microphone**

These 2 inputs are processed by Matlab Environment according to our algorithm and the Intruder motion in a particular region is recorded.

The user can set threshold value according to his/her environmental conditions and the whole setup is done, if there is sound or intruder/motion is detected in the ROI of video image then alarm is raised and Motion and sound is recorded for that instance.

The video file that is being recorded is in AVI format and the naming convention tells that when it was being recorded “IntruderLog-yyyymmddTHHMMS.avi” i.e. Year Month Date and time in Hours Minute and Seconds. AVI format runs on any simple video player.

**3.1.1MOTION AND NOISE DETECTION ALGORITHM:**

Set Motion threshold and sound threshold first

%% Look for motion.

if imageMax > motion Threshold

motion = true;

else

motion = false;

end

%% Look for noise.

if soundMax > soundThreshold

noise = true;

% make noise = true if you want sound intrusion detection activation too.

else

noise = false;

end

**3.2 MAIN PROGRAM:**

The main task of the software was to read the still images recorded from the camera and then process these images to detect motions and take necessary actions accordingly. Figure 3.1 below shows the general flow chart of the main program. It starts with general initialization of software parameters and objects setup. Then, once the program started it start reading the images then process those using SAD algorithms. If a motion is detected it starts a series of actions and then it go back to read the next images, otherwise it goes directly to read the next images. Whenever the flag value will be set to zero and the program is stopped, memory is cleared and necessary results are recorded. This terminates the program and returns the control for the operator to collect the results.

****

**Figure3.1 Main Program Flow Diagram**

**3.3 IMAGE ACQUISITION:**

****

**Figure 3.2 Image acquisitions Process**

After setup stage the image acquisition starts as shown in figure3.2 above. This process reads images from the PC camera and save them in a format suitable for the motion detection algorithm.

There were three possible options from which one is implemented. The first option was by using auto snapshots software that takes images automatically and save them on a hard disk as JPEG format, and then another program reads these images in the same sequence as they were saved. It was found that the maximum speed that can be attained by this software is one frame per second and this limits the speed of detection. Also, synchronization was required between both image processing and the auto snapshot software’s where next images need to be available on the hard disk before processing them.

The second option was to display live video on the screen and then start capturing the images from the screen. This is a faster option from the previous approach but again it faced the problem of synchronization, when the computer monitor goes into a power saving mode where black images are produced all the time during the period of the black screen.

The third option was by using the image acquisition toolbox provided in MATLAB 6.5.1 or higher versions [10] [11].

The image acquisition toolbox is a collection of functions that extend the capability of MATLAB. The toolbox supports a wide range of image acquisition operations, including acquiring images through many types of image acquisition devices, such as frame grabbers and USB PC cameras, also viewing a preview of the live video displayed on monitor and reading the image data into the MATLAB workspace directly.

For this project *video input* function was used to initialize a video object that connects to the PC camera directly. Then *preview* function was used to display live video on the monitor. *Get snapshot* function was used to read images from the camera and place them in MATLAB workspace.

The later approach was implemented because it has many advantages over the others. It achieved the fastest capturing speed at a rate of five frames per seconds depending on algorithm complexity and PC processor speed. Furthermore, the problem of synchronization was solved because both capturing and processing of images were done using the same software.

All read images were converted it into a two dimensional monochrome images. This is because equations in other algorithms in the system were designed with such image format.

**3.4 MOTION DETECTION ALGORITHM**

A motion detection algorithm was applied on the previously read images. There were two approaches to implement motion detection algorithm. The first one was by using the two dimensional cross correlation while the second one was by using the sum of absolute difference algorithm. These are explained in details in the next two sub sections.

**3.4.1 MOTION DETECTION USING TWO DIMENSIONAL CROSS CORRELATION**

First the two images were sub divided into four equal parts each. This was done to increase the sensitivity of calculation where it is easier to notice the difference between part of image rather than a whole one. A two dimensional cross correlation was calculated between each sub image with its corresponding part in the other image. This process produces four values ranging from -1 to 1 depending on the difference of the two correlated images. Because the goal of this division was to achieve more sensitivity the minimum value of correlation will be used as reference to the threshold.

In normal cases, motion can easily be detected when the measured minimum cross correlation value is used to set the threshold. However, detection fails when images contain global variations such as illuminations changes or when camera moves.

****

**Figure 3.3 Direct Thresholds for correlation value**

Figure 3.3 above shows a test case that contains consecutive illumination level changes by switching the light on and off. During the time where the lights are on (frames 1-50 and frames 100-145) the correlation value is around 0.998 and when the lights are switched off (frames 51-99 and frames 146-190) the correlation value is around 0.47. If the threshold for detection was fixed around the value of 0.95 it will continuously detect motion during the light off period.

To overcome this problem continuous re-estimation of threshold value was required. This can be done by using an adaptive filter but it is not easy to design. Another solution is to look at the variance of the set of data produced from the cross correlation process, and detect motion from it. This method solved the problem of changing illumination and camera movements.



**Figure 3.4 Variance Values as Reference for Threshold**

Figure 3.4 shows the variance signal calculated from the same set of images of figure 3.3. It can be seen that the need for continuously re-estimate the threshold value is eliminated. Choosing a threshold of 1\*10-2 will detect the times when only the lights are switched on and off. This results into a robust motion detection algorithm with high sensitivity of detection.

**3.4.2 MOTION DETECTION USING SUM OF ABSOLUTE DIFFERENCE (SAD):** This is both the most obvious and most simple algorithm of all: The two consecutive frames are compared pixel by pixel, summing up the absolute values of the differences of each two corresponding pixels. The result is a positive number that is used as the score. SAD reacts very sensitively to even minor changes within a scene: fast movements of the camera, explosions or the simple switching on of a light in a previously dark scene result in false hits. On the other hand, SAD hardly reacts to soft cuts at all. Yet, SAD is used often to produce a basic set of "possible hits" as it detects all visible hard cuts with utmost probability. Motion vectors on pixel

data reduces the input data rate by about two orders of magnitude, and allows real-time operation on limited computational platforms.

This algorithm is based on image differencing techniques. It is mathematically represented using the following equation:

 3.1

Where  is the number of pixels in the image used as scaling factor,  is the image  at time ,  is the image  at time and  is the normalized sum of absolute difference for that time*.* In an ideal case when there is no motion.

 3.2

and. However noise is always presented in images and a better model of the images in the absence of motion will be

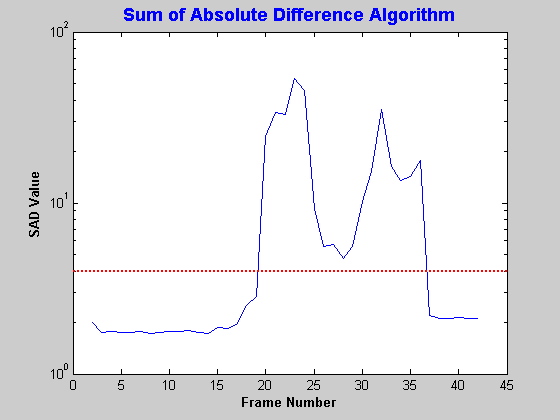
 3.3

Where  is a noise signal.

The value 

that represents

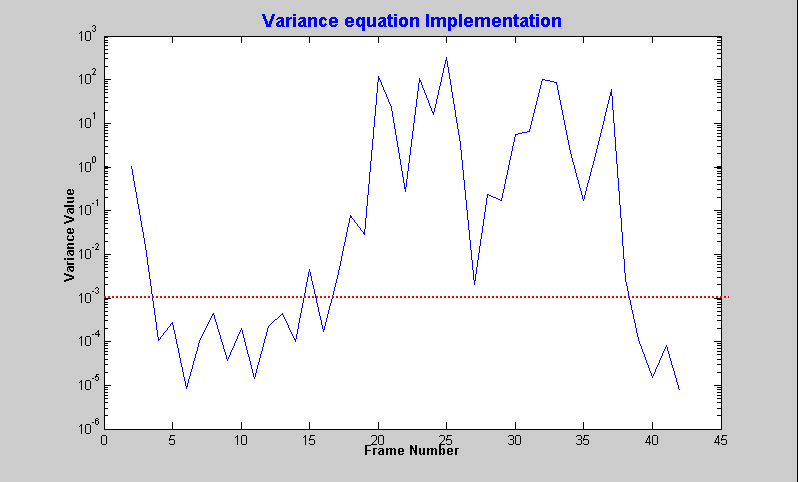
the normalized sum of absolute difference can be used as a reference to be compared with a threshold value as shown in figure 3.5.



**Figure 3.5 Direct Thresholds for SAD Values**

The above figure shows a test case that contains a large change in the scene being monitored by the camera this was done by moving the camera. During the time before the camera was moved the SAD value was around 1.87 and when the camera was moved the SAD value was around 2.2. If the threshold for detection was fixed around the value less than 2.2 it will continuously detect motion after the camera stop moving.

To overcome this problem the same solution that was applied to the correlation algorithm will be used. The variance value was computed after collecting two SAD values and the result is shown for the same test case in figure 3.6.



**Figure 3.6 Variance Values as Reference for Threshold**

This approach solve the need for continuously re-estimate the threshold value. Choosing a threshold of 1\*10-3 will detect the times when only the camera is moved. This results into a robust motion detection algorithm that cannot be affected by illumination change and camera movements.

**3.4.3 ACTIONS ON MOTION DETECTION:**

Before explaining series of actions happen when motion is detected it is worth to mention that the values of variance that was calculated whether it was above or below the threshold will be stored in an array, where it will be used later to produce a plot of frame number Vs. the variance value. This plot helps in comparing the variance values against the threshold to be able to choose the optimum threshold value. Whenever the variance value is less than threshold the image will be dropped and only the variance value will be recorded. However when the variance value is greater than threshold sequence of actions is being started as shown in figure 3.7 below.

****

**Figure 3.7 Actions on Motion Detection**

The above flow chart show a number of activities happen when motion is detected. First the serial port is being triggered by a pulse from the PC; this pulse is used to activate external circuits connected to the PC. Also a log file is being created and then appended with information about the time and date of motion also the frame number in which motion occur is being recorded in the log file. Another process is to display the image that was detected on the monitor. Finally the image that was detected in motion will be converted to a movie frame and will be added to the film structure.

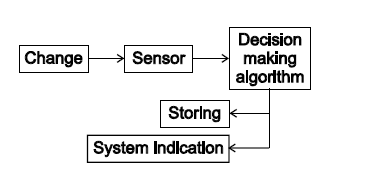
**3.4.4 DATA RECORD:**

Finally when the program is terminated a data collection process starts where variable and arrays that contain result of data on the memory will be stored on the hard disk. This approach was used to separate the real time image processing from results processing. This has the advantage of calling back these data whenever it is required. The variables that are being stored from memory to the hard disk are variance values and the movie structure that contain the entire frames with motion.

**CHAPTER 4**

**EXPERIMENTATION RESULTS & CONCLUSION**

This chapter gives an overview on the equipment/development tools used in developing the entire motion detection system. In addition, this chapter also focuses on demonstrating the performance of the developed detection system with some real world image sequences. The objective of this work was to develop a surveillance system which would detect motion in a live video feed and if Motion is detected, then to activate a warning system and store the video feed for future reference and processing purposes. The activation of an alarm would help in nullifying a threat of security and storing of video provides a proof of such malicious activity. Keeping the work objective in mind, we firstly developed basic system architecture as shown in the Fig.4.1



**Fig. 4.1 Basic system architecture the purposed system**

The system architecture, which we developed, describes how the system component interacts and work together to achieve the overall system goals. It describes the system operation, what each component of the system does and what information is exchange. The architecture was designed for basically getting an idea of how the actual system works and Operates [11],[12].

**4.1 SYSTEM ARCHITECTURE FUNCTIONING:**

The system architecture is going to function in following way:

• Capturing the live video feed through a web cam: To detect motion we first have to capture live video frames of the area to be monitored and kept under surveillance this is done by using a web cam which continuously provides a sequence of video frames in a particular speed of FPS (frames per second).

• Comparing the current frames captured with previous frames to detect motion: For checking whether any motion is present in the live video feed, we compare the live video frames being provided by the web cam with each other so that we can detect changes in these frames and hence predict the occurrence of some motion.

• Storing the frames on the memory if motion is detected: If motion is being detected, we would require storing such motion so that the user can view it in the near future. This also helps the user in providing a legal proof of some inappropriate activity since a video coverage can be used as a proof in the court of law.

• Indicating through an alarm when the motion is detected: The user may want to be notified immediately that there has been some intrusion detected by the software, hence an alarm system is included in the software. This alarm system immediately activates a WAV file format audio alarm signal if any kind of motion is detected hence. This helps in preventing any kind of breach of security at that moment of time.

• Develop an algorithm to fix the coordinates of restricted area i.e. ROI in starting of the motion detection.

**4.2** **ROJECT SETUP AND EXPERIMENTAL RESULTS:**

**4.2.1 EQUIPMENT USED FOR PROJECT DEVELOPMENT**

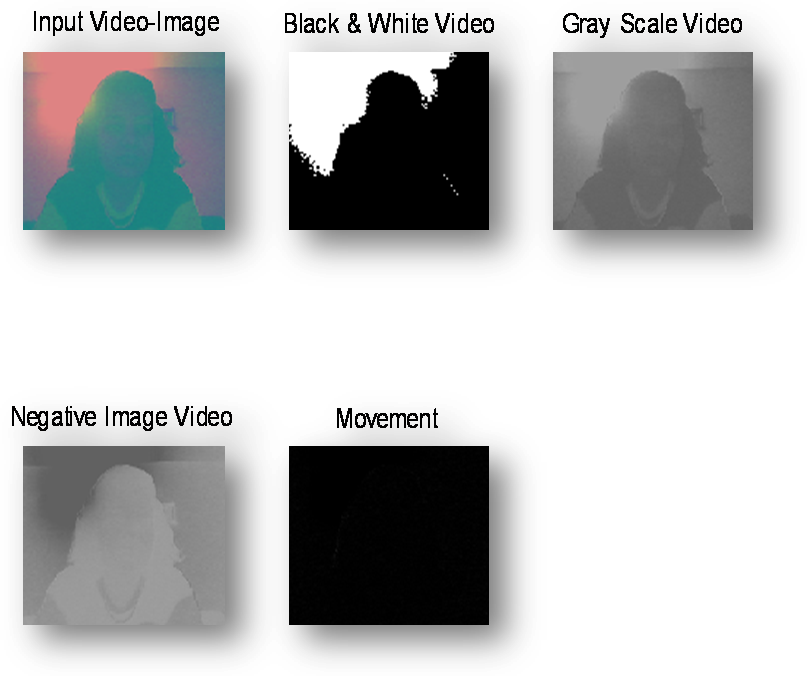
The proposed moving object detection system has been implemented on a PC.The video images were acquired through a web camera. Each frame of the acquired images is converted to the 8 bit greyscale format and finally 1 bit binary format before undergoing further processing. To fulfil our aim, we have used strong computing software called Mat lab 7 [11], [12].

**ADVANTAGE OF MAT LAB:** Basically the advantage of using Mat lab is that Mat lab is an interpreted language for numerical computation. It allows one to perform numerical calculations, and visualize the results without the need for complicated and time consuming programming. Mat lab allows its users to accurately solve problems, produce graphics easily and produce code efficiently. **Entire motion detection and tracking program has been developed using mat lab**. Mat lab is especially useful for evaluating and analyzing the performance of the developed detection and tracking system, as well as for performance comparison [11, 12].

**4.2.2 EXPERIMENTAL RESULTS:**

This section demonstrates some of the tested image sequences that are able to highlight the effectiveness of the proposed detection system. These experimental results are obtained using the proposed detection algorithm that has been discussed in Chapter 3. It has been tested with live video image. The captured images then were broken up into 50 frames per second (50 fps). The results of the proposed detection and tracking algorithm are depicted in Figure 4.1 and Figure 4.2

* Fig 4.1 illustrating that there is no image in the ‘Movement’ frame.
* Fig.4.2 illustrate that there is an image with motion.



**Fig.4.1 When there is no motion/movement**



Input Video-Image



Black & White Video



Gray Scale Video



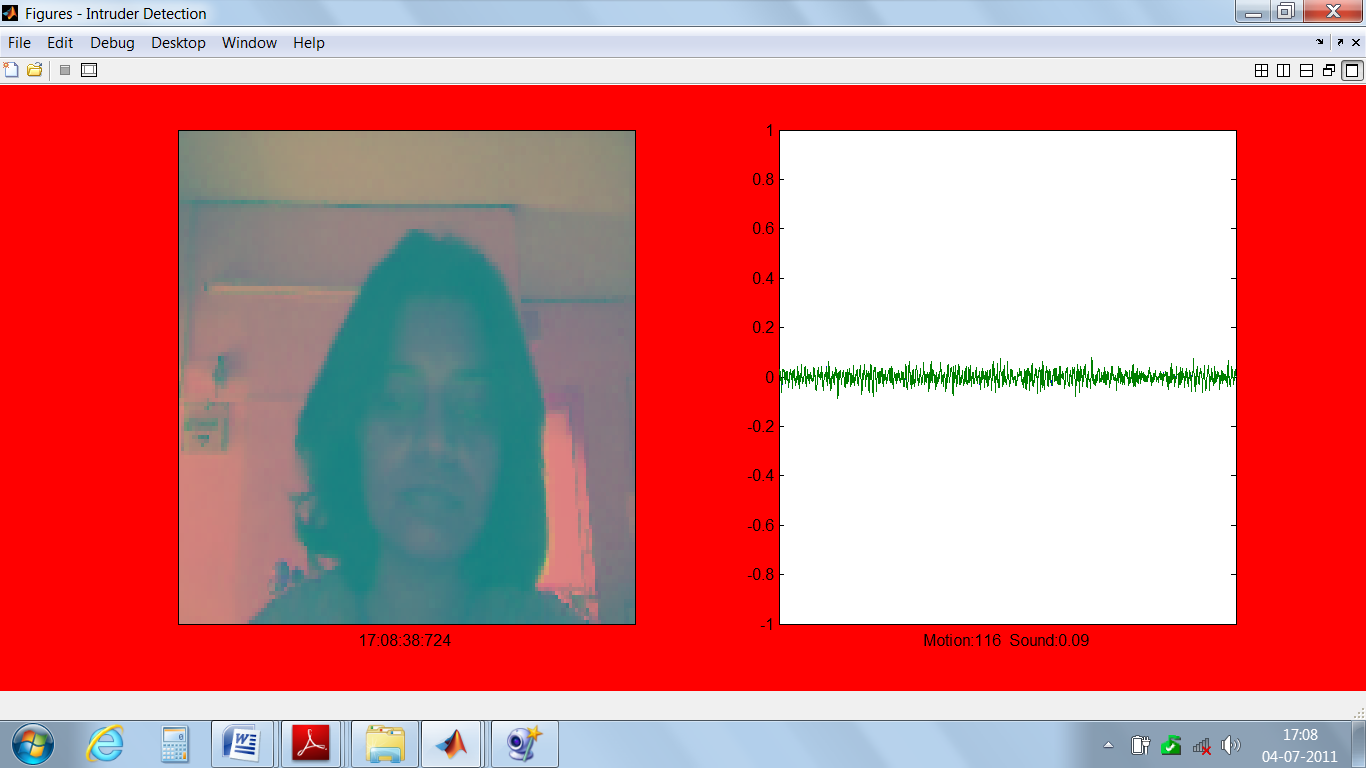
Negative Image



Video Movement

**Fig.4.2 When there is motion/movement**

**RESULTS OF FINAL CODE FOR INTRUDER MOTION DETECTION AND RECORDING:**

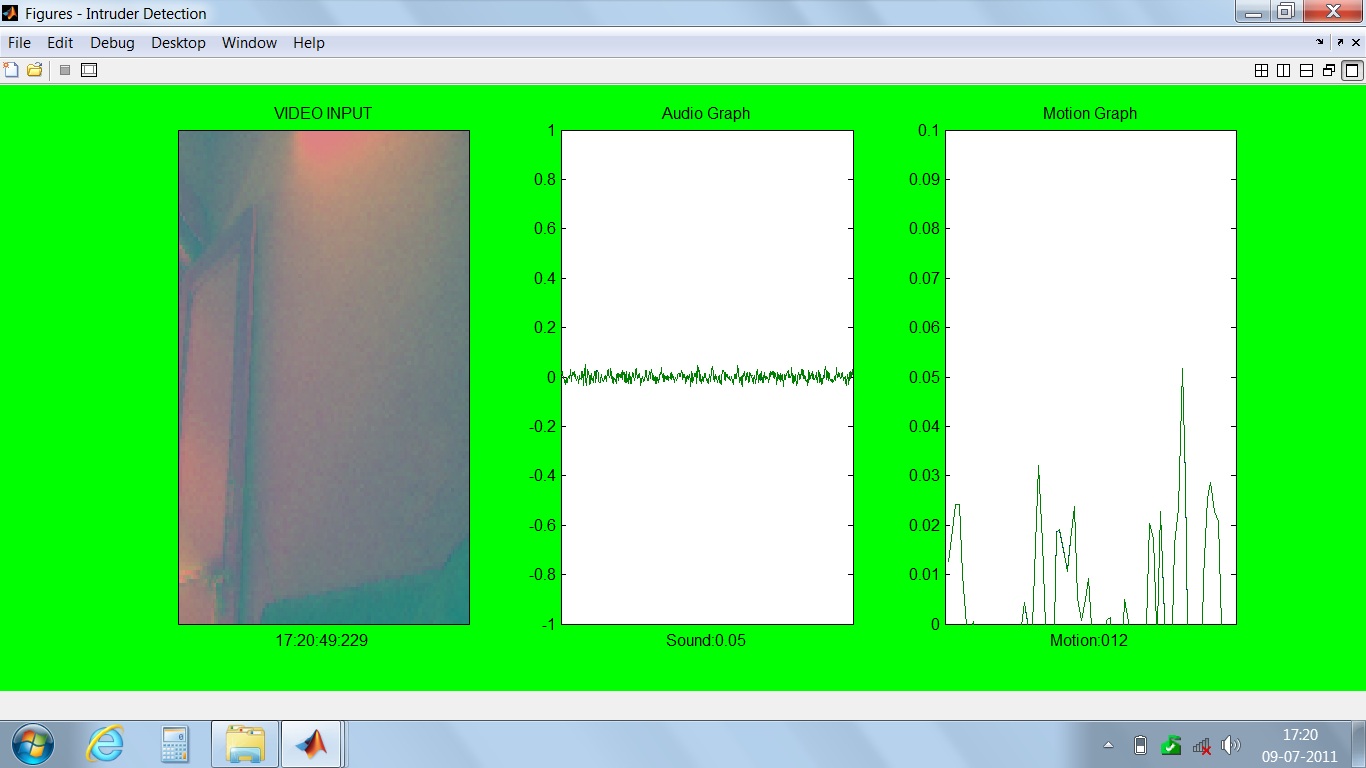
****

**Figure4.3: Showing Image Motion and Audio amplitude wave.**

**RESULT CASE 1:**

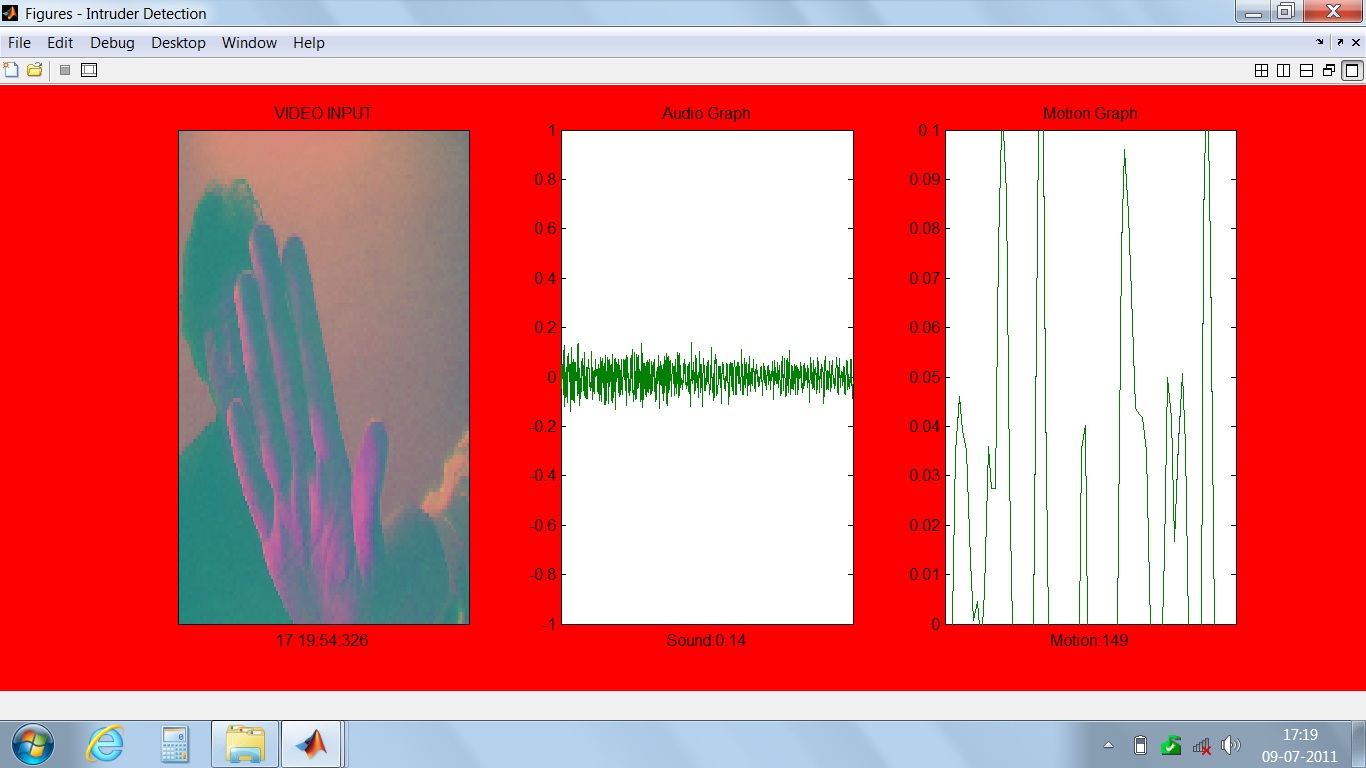
**SIMPLE ENVIRONMENT (Showing hand Movement):**

Code edited to show the histogram for the movement above threshold value set by user according to the environment.

****

**Figure 4.4: No hand movement**

Above figure shows that there is no movement in the area of motion detection (i.e. in the region of interest, ROI), if there is some movement in the video or audio level above threshold value (which is set by user according to the environment) the Matlab setup environment changes showing the motion detection and the background will become red in colour and the recording process starts.

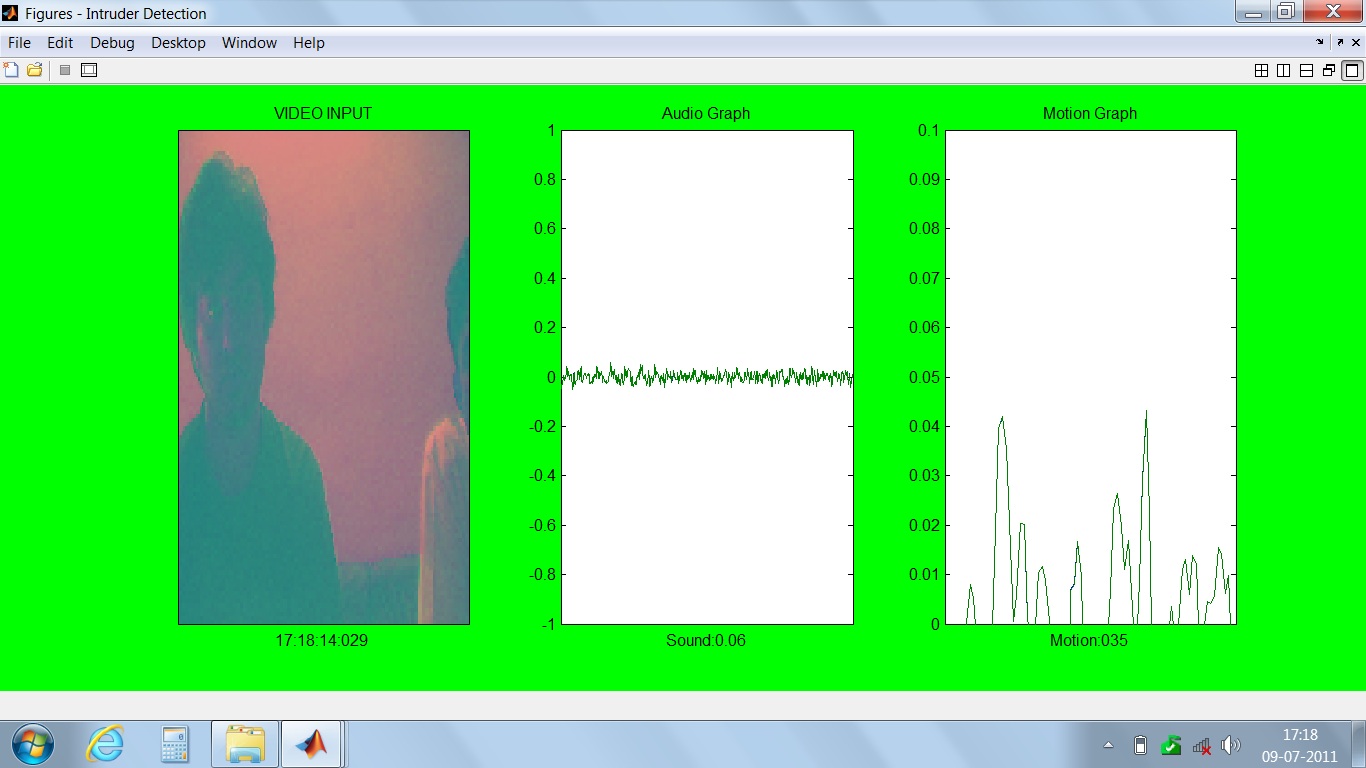


**Figure 4.5: Hand Movement Detection and raise of alarm and Motion histogram**

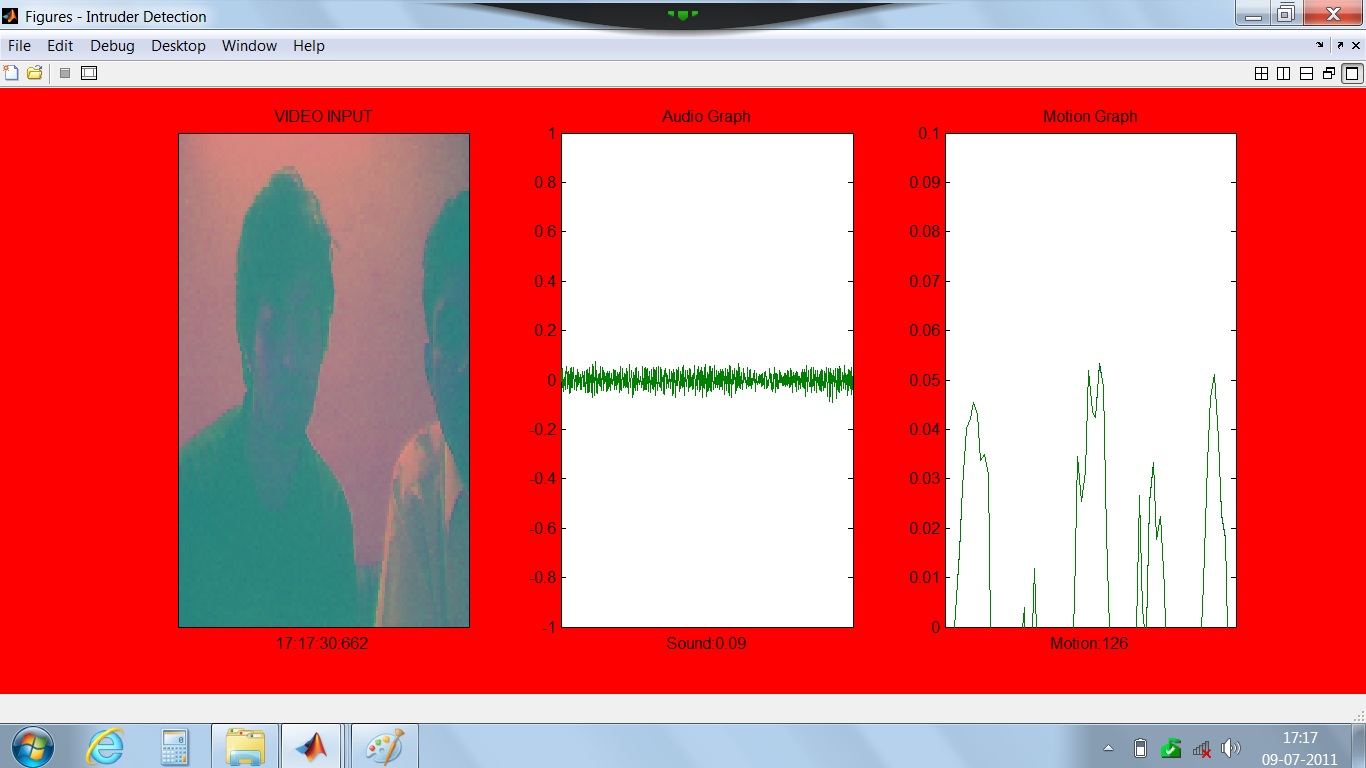
On change in the audio or video input levels, the histogram value (last subplot in the above plot) changes and shows the major motion or audio level change in the surrounding environment.

**RESULT CASE 2:**

**REAL TIME ENVIRONMENT (Showing Object/Intruder motion)**



**Figure4.5: No Intruder Motion**



**Figure4.6: Intruder Motion Detected**

**4.2.3 FEATURES INCLUDED:** Following important features are included in the system. This software can be integrated and used with any company manufactured web cam.

* Our code not only detects object/Intruder video motion but also change in surrounding audio levels so that it can surveillance in both field for whole environment change.
* It has a facility to use any audio (.wav) file as alarm signal. If the user wants he can use the software without the alarm audio signal.
* Video can be recorded after the motion has been detected in the hard disk (AVI format).
* The stored video can be viewed in any media player which allows all the features of a media player and hence we can forward, rewind, pause etc on a stored video for future analysis.

**4.2.4 LIMITATIONS:**

•More features could have been added but due to time Constraint they could not be added. But this is not an issue since he code can be directly extracted and manipulated to include new features.

•The code is dependent on Mat lab compiler without which the code would not run but this is not a problem since acquiring Mat lab is not difficult.

**4.3 CONCLUSION:**

***“*INTRUDER MOTION DETECTION AND RECORDIONG IN THE REGION OF INTEREST*”*** system was thus developed successfully. This system mainly provides an efficient method for surveillance purposes and is aimed to be highly beneficial for any person or organization. Thus, motion based change detection in avi video format was completed and successfully implemented. The future scope of the work done could be as follows: Following changes or additions can be done to include some new features.

• With the existing alarm system, advancement can be included and SMS can be sent to the user when motion is detected.

• The stored video can be automatically transferred to some email account so that an extra backup data can be used.

• A user\_id and password can be given to a user so that unauthorized people don’t have access to the software.

• In the future, the user can be provided a remote access to this software from some remote PC through internet.

**REFERENCES:**

[1] E. Grimson, C. Stauffer, R. Romano, and L. Lee, ªUsing Adaptive Tracking to Classify and Monitoring Activities in a Site, Proc. Computer Vision and Pattern Recognition Conf., pp. 22-29, 1998.

[2] E. Grimson and C. Stauffer, Adaptive Background Mixture Models for Real Time Tracking, Proc. Computer Vision and Pattern Recognition Conf., 1999.

[3] A. Calvango, G. Fantozzi, F. Rinaldo, and R. Viareggio. Model {based global and local motion estimation for video conference sequences. IEEE Transactions on Circuits and Systems for Video Technology, 14(9):1156}1161, September 2004.

[4] Zoran Zivkovic, Improved Adaptive Gaussian Mixture Model for Background Subtraction, Intelligent and Autonomous Systems Group University of Amsterdam, the Netherlands.

[5] N. Friedman and S. Russell, Image Segmentation in Video Sequences: A Probabilistic Approach, Uncertainty in Artificial Intelligence, 1997.

[6] I. Haritaoglu, D. Harwood, and L. Davis, W4: Who, When, Where, What: A Real Time System for Detecting and Tracking People, Proc. Third Face and Gesture Recognition Conf., pp. 222-227, 1998.

[7] Sen-Ching S. Cheung and Chandrika Kamath, Robust techniques for background subtraction in urban traffic video, Center for Applied Scientific Computing Lawrence Livermore National Laboratory 7000 East Avenue, Livermore, CA 94550.

[8] Zoran Zivkovic’, Motion Detection and Object Tracking in Image Sequences, Ph. D Thesis

[9]Optimal Placement of Light Sources for a Visual Sensor Network , S.Indu, Sakar Arora, Shyam Shankaran , Santanu Chaudhury and Asok Bhattacharyya.

[10] RAFAEL C. GONZALEZ, RICHARD E. WOODS. (2002): Digital Image processing, Prentice Hall International

[11] Image Processing Toolbox™ 6 User’s Guide

[12] Motion detection with image acquisition toolbox, *Math works*, and Mat lab.

APPENDIX

Original Project Code done in MATLAB

function intruderdetection(logName, secondsToRecord, silentAlarm)

%

% Project Made by Yogeswari tolia

%

if ~ispc

error('INTRUDERDETECTION is only supported on Microsoft Windows platforms.');

end

%% Set defaults, if none provided.

if nargin < 3

silentAlarm = false;

end

if nargin < 2

secondsToRecord = 10;

end

if nargin < 1

logName = ['IntruderLog-' datestr(now,30)];

end

%% Other settings...

% Maxmimum rate at which to record video to disk.

maxDiskFrameRate = 5;

% Video codec to use.

videoCompression = 'Cinepak';

% Maximum rate at which to display video and audio data.

maxDisplayRate = 10;

% Threshold at which to consider the motion something of interest.

motionThreshold = 110; % Between 0 and 255

% Threshold at which to consider the sound something of interest.

soundThreshold = 0.55; % Between 0.0 and 1.0 volts.

%% Create a video input object and determine the average frame rate.

vid = videoinput('winvideo');

disp(sprintf('Measuring actual frame rate...'));

triggerconfig(vid,'manual');

set( vid, 'FramesPerTrigger', 50 );

start(vid);

pause(1.0);

trigger(vid);

% Acquire 50 frames or 20 seconds of data, whichever comes first.

try

wait(vid,20,'running');

catch

% Supress any error and just use the frames we were able to get.

stop(vid);

end

[frames,relTimes] = getdata(vid, vid.FramesAvailable);

actualFrameRate = 1/mean(diff(relTimes));

disp(sprintf('Average frame rate:%f measured over %d frames',...

actualFrameRate, size(frames,ndims(frames))));

%% Configure the video input object to detect motion and record video when

%% triggered.

% Keep the recording frame rate under maxDiskFrameRate.

set(vid, 'FrameGrabInterval', ceil(actualFrameRate / maxDiskFrameRate) );

framesPerSecond = actualFrameRate / get(vid, 'FrameGrabInterval');

frameLogFile = avifile(logName,...

'Compression', videoCompression,...

'Fps', framesPerSecond );

set(vid, 'TriggerRepeat', Inf);

framesToRecord = ceil(framesPerSecond \* secondsToRecord);

set(vid, 'FramesPerTrigger', framesToRecord);

set(vid, 'LoggingMode', 'disk');

set(vid, 'DiskLogger', frameLogFile);

set(vid, 'StartFcn', @videoStartFcn);

% Display at close to the actual frame rate but no more than maxUpdateRate.

displayPeriod = max(1.1\*(1/actualFrameRate), 1/maxDisplayRate);

set(vid, 'TimerPeriod', displayPeriod);

set(vid, 'TimerFcn', @videoTimerFcn);

% Adjust the number of seconds that we are recording to match the

% number of frames.

secondsToRecord = framesToRecord / framesPerSecond;

%% Configure the analog input object to detect noise and record sound when

%% triggered.

ai = analoginput('winsound');

addchannel(ai,[1 2]);

set(ai, 'LoggingMode', 'disk');

set(ai, 'LogFileName', logName);

set(ai, 'LogToDiskMode', 'overwrite');

set(ai, 'TriggerType', 'manual');

set(ai, 'TriggerRepeat', Inf);

samplesToRecord = ceil(ai.SampleRate \* secondsToRecord);

set(ai, 'SamplesPerTrigger', samplesToRecord );

%% Configure the analog output object to make alarm.

% Shorten alarm so the sound doesn't trigger a new intruder alert.

secondsToAlarm = secondsToRecord - 1;

% If alarm is too short, make it silent.

if secondsToAlarm < 1

secondsToAlarm = 1;

silentAlarm = true;

end

ao = analogoutput('winsound');

[alarmData, frequency, nbits] = customAlarmData();

addchannel(ao,1:size(alarmData,2)); % Add a channel for each alarm channel.

set(ao, 'SampleRate', frequency);

set(ao, 'BitsPerSample', nbits);

putdata(ao, alarmData);

set(ao, 'TriggerType', 'immediate');

set(ao, 'RepeatOutput', Inf);

set(ao, 'TimerPeriod', secondsToAlarm);

set(ao, 'TimerFcn', @alarmTimerFcn);

set(ao, 'StopFcn', @alarmStopFcn);

%% Create the figure

fig = figure('DoubleBuffer','on', ...

'Name', 'Intruder Detection', ...

'NumberTitle', 'off', ...

'WindowStyle', 'docked', ...

'Toolbar', 'none', ...

'MenuBar', 'none', ...

'Color',[.5 .5 .5], ...

'CloseRequestFcn', @figureCloseFcn, ...

'DeleteFcn', @figureDeleteFcn);

%% Initialize the previous image.

imagePrevious = [];

timePrevious = [];

%% Start the sound and video input objects

start(ai);

start(vid);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generate custom alarm data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [data, frequency, nbits] = customAlarmData()

[data, frequency, nbits] = wavread('alarm.wav','double');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Do any user-specific action when intruder is detected.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function customIntruderAction()

% e.g. Send notification.

%sendmail <your email address> 'Intruder Alert';

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Video Timer Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function videoStartFcn(vid, event) %#ok<INUSD>

% Get initialize image and time.

imagePrevious = getsnapshot(vid);

timePrevious = now;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Video Timer Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function videoTimerFcn(vid, event) %#ok<INUSD>

%% Get the most recent image and time.

try

imageCurrent = getsnapshot(vid);

timeCurrent = now;

catch

% getsnapshot can fail if object is deleted while we are waiting.

return;

end

%% Get the sound that occured since the last image.

samplesRequested = ceil((timeCurrent - timePrevious) \* (60\*60\*24) \*...

ai.SampleRate);

warning('off','daq:peekdata:requestedSamplesNotAvailable');

try

sound = peekdata(ai, samplesRequested);

catch

% Occasionally, peekdata fails.

sound = zeros(samplesRequested, length(ai.Channel));

end

warning('on','daq:peekdata:requestedSamplesNotAvailable');

%% Compute the difference between the current and previous images

[xs ys zs]=size(imagePrevious);

% [xs1 ys1 zs1]=size(imageCurrent);

imageDifference = abs(imagePrevious((floor(xs/2)-40):(floor(xs/2)+40),(floor(ys/2)-40):(floor(ys/2)+40),:) - imageCurrent((floor(xs/2)-40):(floor(xs/2)+40),(floor(ys/2)-40):(floor(ys/2)+40),:));

imageMax = max(imageDifference(:));

% save('img','imageCurrent');

%% Compute the loudest sound detected during the interval.

sound = sound - mean(sound(:,1)); % Center about the mean.

soundMax = max(max(abs(sound))); % Calculate max deviation from mean.

% Save the current image and time for the next iteration.

imagePrevious = imageCurrent;

timePrevious = timeCurrent;

%% Make our figure current.

figOld = get(0,'CurrentFigure');

if fig ~= figOld

set(0, 'CurrentFigure', fig);

end

%% Show the current image and time.

subplot(1,3,1);

image(imageCurrent);

label = datestr(timeCurrent, 'HH:MM:SS:FFF');

title('VIDEO INPUT');

xlabel(label);

set(gca,'XTick',[], 'YTick',[]);

%% Show the sound and maximum values.

subplot(1,3,2);

plot(sound);

axis([0 size(sound,1) -1 1]);

label = sprintf('Sound:%04.2f', soundMax);

title('Audio Graph');

xlabel(label);

set(gca,'XTick',[]);

%% Show the sound and maximum values.

subplot(1,3,3);

plot(sound);

title('Motion Graph');

axis([0 size(imageDifference,1) 0 0.1]);

label = sprintf('Motion:%03d', imageMax);

xlabel(label);

set(gca,'XTick',[]);

%% Look for motion.

if imageMax > motionThreshold

motion = true;

else

motion = false;

end

%% Look for noise.

if soundMax > soundThreshold

noise = true;

% make noise = true if you want sound itrusion detection activation too.

else

noise = false;

end

% If we are not yet in a recording/alarm state.

if ~islogging(vid) && ~islogging(ai) && ~isrunning(ao)

% If either motion or noise occur, begin recording.

if ( motion || noise )

% Give visual indication of recording.

set(gcf, 'Color', [1 0 0]);

% Start alarm if desired.

if ~silentAlarm

start(ao);

end

% Begin recording sound.

trigger(ai);

% Begin recording video.

trigger(vid);

% Do any custom action.

customIntruderAction();

else

% Give visual indication of not recording.

set(gcf, 'Color', [0 1 0]);

end;

end

% Restore previous figure.

if fig ~= figOld

set(0, 'CurrentFigure', figOld);

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Alarm Timer Callback

% Stop the alarm.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function alarmTimerFcn(obj, event) %#ok<INUSD>

% Stop the alarm.

stop(obj);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Alarm Stop Callback

% Reload the alarm data for the next time.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function alarmStopFcn(obj, event) %#ok<INUSD>

% Reload data for the next time alarm is started.

putdata(obj, alarmData);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Figure Close Function

% Stop objects and close figure window.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figureCloseFcn(obj, event) %#ok<INUSD,INUSD>

try

% Stop objects so they don't write to figure any more.

stop(vid);

stop([ai ao]);

catch

end

closereq;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Figure Delete Function

% Cleanup objects and files.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figureDeleteFcn(obj, event) %#ok<INUSD,INUSD>

% Save video and audio events to a MAT file.

videoEvents = vid.EventLog; %#ok<NASGU>

audioEvents = ai.EventLog; %#ok<NASGU>

save(logName,'videoEvents','audioEvents');

% Close video logging file.

videoFile = close(vid.DiskLogger);

% Delete objects.

delete(vid);

delete([ai ao]);

% Convert audio data into wav file.

disp(sprintf('Writing wave file...'));

soundData = daqread(logName);

soundInfo = daqread(logName, 'info');

wavwrite(soundData, soundInfo.ObjInfo.SampleRate, ...

soundInfo.HwInfo.Bits, logName);

disp(sprintf('Done.'));

end

end