CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION TO IMAGE PROCESSING

Image Processing is a technique to enhance raw images received from cameras/sensors placed on space probes, aircrafts and satellites or pictures taken in normal day-today life for various applications. An Image is rectangular graphical object. Image processing involves issues related to image representation, compression techniques and various complex operations, which can be carried out on the image data. The operations that come under image processing are image enhancement operations such as sharpening, blurring, brightening, edge enhancement etc. Image processing is any form of signal processing for which the input is an image, such as photographs or frames of video; the output of image processing can be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it.

Many techniques have been developed in Image Processing during the last four to five decades. Most of the methods are developed for enhancing images obtained from

unmanned space probes, space crafts and military reconnaissance flights. Image Processing systems are becoming widely popular due to easy availability of powerful personnel computers, large memory devices, graphics softwares and many more. Image processing involves issues related to image representation, compression techniques and various complex operations, which can be carried out on the image data. The operations that come under image processing are image enhancement operations such as sharpening, blurring, brightening, edge enhancement.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information extraction. An image may be defined as a two-dimensional function, f(x, y), where x and y are spatial (plane) coordinates, and the amplitude off at any pair of coordinates(x, y) is called the intensity or gray level of the image at that point. When x, y, and the amplitude values of f are all finite, discrete quantities, we call the image a digital image. The field of digital image processing refers to processing digital images by means of a digital computer. Note that a digital image is composed
of a finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements, pels and pixels. Pixel is the term most widely used to denote the elements of a digital image. Vision is the most advanced of our senses, so it is not surprising that images play the single most important role in human perception. However, unlike humans, who are limited to the visual band of the electromagnetic (EM) spectrum, imaging machines cover almost the entire EM spectrum, ranging from gamma to radio waves. They can operate on images generated by sources that humans are not accustomed to associating with images. These include ultrasound, electron microscopy, and computer-generated images. Thus, digital image processing encompasses a wide and varied field of applications.

Steps involved in the Digital Image Processing

- Image acquisition
- Image Enhancement
- Image restoration
- Color Image Processing
- Wavelets and multi resolution Processing
- Compression
- Morphological Processing
- Segmentation
- Representation and description
- Object Recognition
- Display

Examples of some fields that use Digital Image Processing are

- Medicine
- Astronomy
- Law Enforcement
- Industries
- Biological Imaging
- Geology

1.2 LITERATURE SURVEY

The ongoing research on object tracking in video sequences has attracted many researchers. Detecting the objects in the video and tracking its motion to identify its characteristics has been emerging as a demanding research area in the domain of image processing and computer vision. Here is a literature review on the state of the art tracking methods, categorize them into different categories, and then identify useful tracking methods. Most of the methods include object segmentation using background subtraction. The tracking strategies use different methodologies like Mean-shift, Kalman filter, Particle

filter etc. The performance of the tracking methods vary with respect to background information.

Tracking objects in video sequences of surveillance camera is nowadays a demanding application. Tracking objects is much more challenging in video sequences to improve recognition and tracking performances. There are many existing methods of object tracking but all has some drawbacks.

1.2.1 METHODS OF OBJECT TRACKING

Contour-based object tracking model

Active contour model is used for finding object outline from an image [7]. In the contour-based tracking algorithm, the objects are tracked by considering their outlines as boundary contours. Thereafter these contours are updated dynamically in successive frames. The discrete version of this approach is represented in active contour model. The discrete version of this approach takes the advantage of the point distribution model to limit the shape. However, this algorithm is highly sensitive to the initialization of tracking, making it difficult to start tracking automatically.

Region-based object tracking model

The region based object model bases it's tracking of objects on the color distribution of the tracked object [8][9]. It represents the object based on the color. Hence, it is computationally efficient. However, its efficiency is degraded when several objects move together in the image sequences. It is not possible to achieve accurate tracking when multiple objects move due to occlusion. Also, in the absence of any object shape information, the object tracking is largely dependent on the background model used in the extraction of the object outlines.

Feature point based tracking algorithm

In Feature point based model feature points is used to describe the objects [10][11]. There are three basic steps in feature point based tracking algorithm. The first step is to recognize and track the object by extracting elements. The second step is to cluster them into higher level features. The last step is to match these extracted features between images in successive frames. Feature extraction and feature correspondence are the important steps of feature based object tracking. The challenging problem in feature point based tracking is feature correspondence because a feature point in one image may have many similar points in another image, and hence results in feature correspondence ambiguity.

1.2.2 METHODA OF OBJECT DETECTION

Segmentation based

Segmentation based algorithms are used to segment the image frame into segments to find out the objects of interest. Criteria for good partition and efficient partitioning method plays important role in segmentation algorithms. Later on the segmented objects are considered for tracking.

A. Graph cut

In graph cut method the input image is considered as a graph. The segmentation of the objects in the image is considered as the graph partitioning problem. For a graph G (image), the vertices (i.e. pixels), **V** = {u, v…}, are partitioned into N disjoint sub-graphs (regions), Ai, Ai =**V**, Ai _ Aj =null, i _ j, by pruning the weighted edges of the graph. Based on the similarity of color, brightness and texture, weight between the nodes is computed. The minimum cut criterion for partitioning an image proposed by Wu and Leahy uses color similarity for weight calculation but their method suffers from over segmentation. Yi and Moon [12] considered graph cut image segmentation as pixel labeling problems. The label of the foreground object (s-node) is set to be 1 and the background (t-node) is set to be 0. By minimizing the energy-function with the help of minimum graph cut the process of pixel labeling can be done. Shi and Malik [13] propose the normalized cut to overcome the over segmentation problem. The ‘cut ’of their method depends on the sum of weights of

the edges in the cut and on the ratio of the total connection weights of nodes in each partition to all nodes of the graph. For image based segmentation, the product of the spatial proximity and color similarity defines the weights between the nodes.

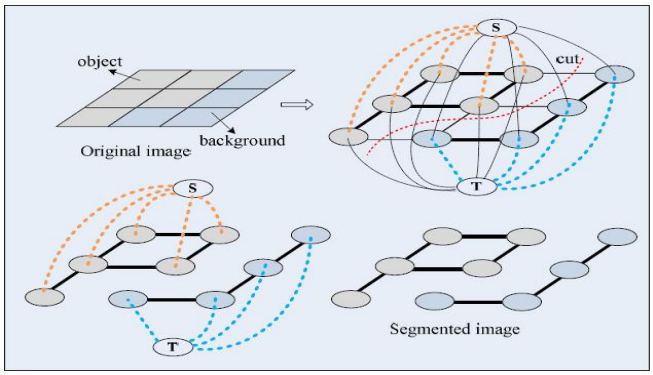

Fig 1 Illustration of Graph cut for image Segmentation

Supervised Learning based Background Subtraction

Supervised learning based background subtraction method can also be used for object detection. Supervised learning mechanism helps in learning of different objects view from a set of examples automatically. Supervised learning methods generate a function that maps inputs to desired outputs for a given set of learning examples. Classification problem is the standard formulation of supervised learning, where the learner approximates the behavior of a function. This approximation is done by generating an output in the form of either a continuous value. This process is called regression, or a class label, which is called classification. Some of the learning approaches are boosting. Viola et al [14], support vector machines Papa GeorgiouEt al. [15] etc.

A. Adaptive Boosting

Boosting is done by combining many base classifiers to find accurate results. In the first step of training phase of the Adboost algorithm is an initial distribution of weights over the training set is constructed. The first step of Adaptive boosting is that the boosting mechanism selects the base classifier with least error. The error of the classifier is

proportional to the misclassified data weights. Next, the misclassified data weights are increased which are selected by the base classifier. In the next iteration the algorithm selects another classifier that performs better on the misclassified data.

B. Support Vector Machines

For a linear system, the available data can be clustered into two classes or groups by finding the maximum marginal hyper plane that separates one class from the other with the help of Support Vector Machines. The distance of hyper plane and the closest data points helps in defining the margin of the maximized hyper plane. The data points that lie on the hyper plane margin boundary are called the support vectors. For object detection purpose the objects can be included in two classes, object class (positive samples) and the non-object class (negative samples). For applying SVM classifier to a nonlinear system, a kernel trick has to be applied to the input feature vector which is extracted from the input.

1.2.3 STATISTICAL METHODS

The importance of an object tracker is that it finds out the motion trajectory of an object as video Frames progresses along with time by identifying the object position in every frame of the video. The complete region that is occupied by the object in the image at every time instant can also be found out by the object tracker. The detected objects in frames are being tracked in the subsequent frames. The object detection task and object correspondence establishment task between the instances of the object across frames can be done separately or jointly. In the first scenario, with the help of object detection algorithm possible object regions in every frame are obtained, and objects correspondence across frames is performed by object tracker. In the latter scenario, information obtained from previous frames helps in finding the object region and correct estimation of correspondence is done jointly by iterative updating of object region and its location.

Kalman Filter

It is a single object state estimation procedure**.** Kalman filter is used as an estimator to predict and correct system state. It helps in studying system dynamics, estimation, analysis, control and processing. It is not only powerful practically but also very well precise theoretically. Kalman filter predicts the states of past, present, and future of an object or variable efficiently. For a linear system Kalman filter finds the correct estimation, with white Gaussian noise.

The two most important steps of Kalman filter are prediction (time update) step and correction (measurement update) step. A state model is used by the prediction step to predict the new state of the variables.

The Extended Kalman filter (EKF) is a nonlinear version of Kalman Filter. Extended Kalman filter uses Kalman filters to linearize about the current mean and covariance. The result of Extended Kalman Filtering shows faster convergence in the terms of iterations in comparison to traditional methods, though each iteration cost is higher. There might also be some cases where EKF finds better or more robust solutions. In recent days Extended Kalman Filtering (EKF) along with ANN is being used in training.

We divide the tracking approaches into three categories, contour based, region based and feature based approach. In our survey we have seen that moving tracking is a kind of motion tracking. Tracking object motion is done by object detection and then using tracking strategy. In this paper, we survey the various approaches of object tracking, including feature descriptors and object segmentation technique in video frames and various tracking methodologies.

We preferred kalman filter for tracking new locations because of the following reasons

- Good results in practice due to optimality and structure.
- Convenient form for online real time processing.
- Easy to formulate and implement given a basic understanding.

1.3 MOTIVATION

Accidents are one of the man-made hazards that we observe very frequently on roads. The major cause of the road accidents are the mistakes of the drivers. The drivers are often prone to poor visibility of the road, this may be due to many reasons. Firstly, the weather conditions like snow, fog and rain cause the diminishing of the human visible range. Thus driver cannot view and observe the people on road clearly, thus leading to accidents. Secondly, the old people and the people having poor eye sight do not have the clear view of the road and people on the road. So it is difficult for these kind of people to drive on the busy roads. Also, whenever there is crowd gathering or public function people move in masses causing traffic problems. So the drivers passing through that route are often struck in the traffic jams without knowing the actual problem. So there is a critical need to ensure the road safety by providing prior informing to the people travelling on road that there is some problem ahead. Human object detection for road safety is very important in modern as day life as the accidents in the recent years has increased very heavily. Due to poor eye of the drivers, due to poor visibility due to the climatic conditions, pedestrians are often not clearly visible to the people in the vehicle. So the accidents are at high probability in these kind of circumstances. Also the people having very poor eye sight could not drive properly as they could not identify the pedestrians suddenly. Also there is no proper analysis about the pedestrian traffic on the roads. In India as pedestrian traffic cannot be perfectly restricted, pedestrians on road need to be properly analyzed so that the drivers can be alerted much before they reach the area of heavily crowded regions. But there is no proper system to perform this task efficiently. So there is need for pedestrian and human detection on roads for road safety.

As the need for road safety is rapidly increasing, a solution is needed to clear this problem. By depending upon some normal statistics, we do not do not a find a long term solution for this problem. So there is definitely a need to use of technology to solve this. By use of image processing we can easily detect the people on road including the pedestrians and the people on cycles and bikes. This can be processed to a video that will be displayed on the screen of the drivers in a vehicle.

1.4 OBJECTIVE

With the implementation of this project, we are aiming at the below mentioned objectives

- To improve the road safety by providing more information about the human objects (pedestrians) to the people travelling on the street.
- To help the drivers with poor eye sight for recognizing the pedestrians.
- To analyze the traffic data by discriminating two wheelers and four wheelers on the road.

In the GUI we developed, we can load a recorded video into the interface and then it detects the human objects and highlights them using the tracker.

The major steps can be summarized as follows

- Auxiliary input and global parameters of the tracking system.
- Create System Objects for the Tracking System Initialization.
- The initialize Tracks function creates an array of tracks, where each track is a structure representing a moving object in the video.
- Read a Video Frame.
- The detect People function returns the centroids, the bounding boxes, and the classification scores of the detected people.
- Predict New Locations of Existing Tracks using the Kalman Filter.
- Assigning object detections in the current frame to the existing tracks.
- The updateAssignedTracks function updates each assigned track with the corresponding detection.
- The deleteLostTracks function deletes tracks that have been invisible for too many consecutive frames.
- Create new tracks from unassigned detections.

- The displayTrackingResults function draws a colored bounding box for each track on the video frame. The level of transparency of the box together with the displayed score indicate the confidence of the detections and tracks.

1.5 PROJECT LAYOUT

We developed a GUI (GRAPHICAL USER INTERFACE) in which we can load a video into the interface. After Pushing the Process button, the loaded video is processed and the pedestrians are detected and they ate tracked continuously. There is also an exit button in the GUI from which we can exit from the GUI. We have used Matlab GUIDE tools for building the GUI. The GUI we built is shown below.

Fig 2 GUI of our project

The pedestrians are detected and they are highlighted using the bounding boxes. Those detected people are tracked in continuous frames until they are disappeared from the frame. The Motion-Based Multiple Object Tracking, contains several algorithmic steps. These steps include people detection, customized non-maximum suppression, and heuristics to identify and eliminate false alarm tracks.

After the video is loaded into the interface, if we press the process button, the video is processed and the people are detected and tracked continuously. If new people come into the frame, then new tracks are created for those people and they are also being tracked. Some new tracks are created due to noise and turbulences in the video. They are eliminated after some time. Each track has some confidence score based on the visibility of the track in the past TimeSizeWindow (T).

This tracking system requires a data file that contains information that relates the pixel location in the image to the size of the bounding box marking the pedestrian's location. This prior knowledge is stored in a vector **pedScaleTable**. The n-th entry in **pedScaleTable** represents the estimated height of an adult person in pixels. The index n references the approximate Y-coordinate of the pedestrian's feet.

Important techniques used in our project

- The **detectPeopleACF** technique is used to detect the people in a frame using Aggregate Channel Features.
- **GUIDE** tools are used for developing a MATLAB GUI.
- **Kalman filter** is used to predict the centroid of each track in the current frame, and update its bounding box accordingly. We take the width and height of the bounding box in previous frame as our current prediction of the size.
- **AssignDetectionsToTracks** function uses the Munkres' version of the Hungarian algorithm to compute an assignment which minimizes the total cost. It returns an M x 2 matrix containing the corresponding indices of assigned tracks and detections in its two columns. It also returns the indices of tracks and detections that remained unassigned.
- **SelectStrongestBbox** - Apply non maximum suppression to select the strongest bounding boxes.
- **Vision.KalmanFilter** class is used to correct the location estimate. Next, it stores the new bounding box by taking the average of the size of recent (up to) 4 boxes.

Noisy detections tend to result in creation of false tracks. For this example, we remove a track under following conditions. The displayTrackingResults function draws a colored bounding box for each track on the video frame. The level of transparency of the box together with the displayed score indicate the confidence of the detections and tracks.

CHAPTER 2: DETAILED DESCRIPTION OF OUR PROJECT

2.1 KALMAN FILTER

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone, by using Bayesian inference and estimating a joint probability distribution over the variables for each timeframe. The filter is named after Rudolf E. Kalman, one of the primary developers of its theory.

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization. The Kalman filter has also found use in modeling the central nervous system's control of movement. Due to the time delay between issuing motor

commands and receiving sensory feedback, use of the Kalman filter provides the needed model for making estimates of the current state of the motor system and issuing updated commands.

The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

The Kalman filter does not require any assumption that the errors are Gaussian. However, the filter yields the exact conditional probability estimate in the special case that all errors are Gaussian-distributed.

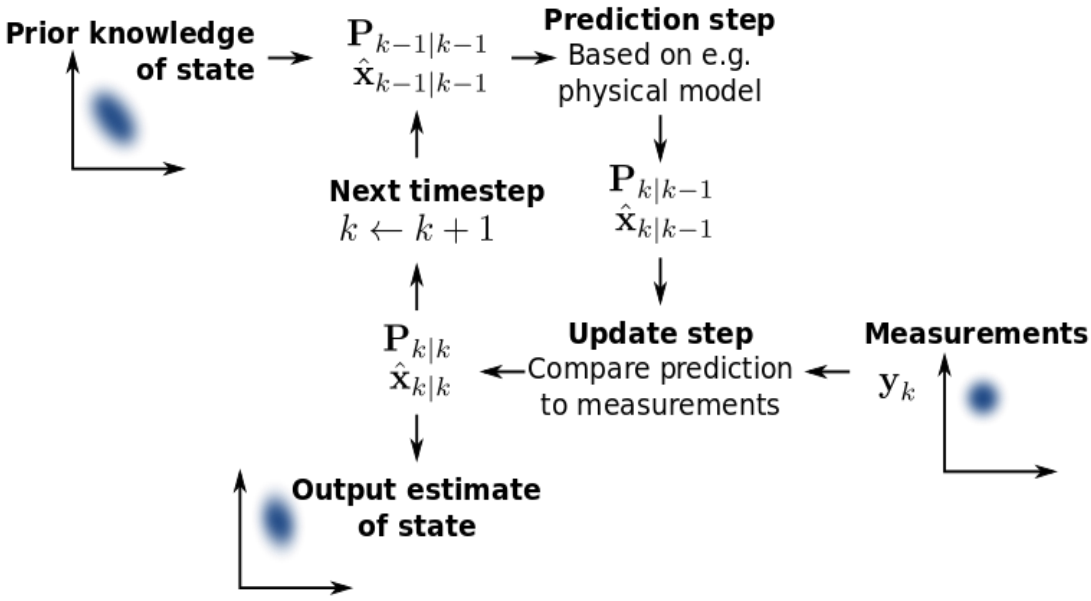

Fig 3 Basic Concept of Kalman Filtering

The Kalman filter uses a system's dynamics model (e.g., physical laws of motion), known control inputs to that system, and multiple sequential measurements (such as from sensors) to form an estimate of the system's varying quantities (its state) that is better than the estimate obtained by using any one measurement alone. As such, it is a common sensor fusion and data fusion algorithm.

All measurements and calculations based on models are estimated to some degree. Noisy sensor data, approximations in the equations that describe how a system changes, and external factors that are not accounted for introduce some uncertainty about the inferred values for a system's state. The Kalman filter averages a prediction of a system's state with a new measurement using a weighted average. The purpose of the weights is that values with better (i.e., smaller) estimated uncertainty are "trusted" more. The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state. The result of the weighted average is a new state estimate that lies between the predicted and measured state, and has a better estimated uncertainty than either alone. This process is repeated every time step, with the new estimate and its covariance informing the prediction used in the following iteration. This means that the Kalman filter works recursively and requires only the last "best guess", rather than the entire history, of a system's state to calculate a new state.

Because the certainty of the measurements is often difficult to measure precisely, it is common to discuss the filter's behavior in terms of gain. The Kalman gain is a function of the relative certainty of the measurements and current state estimate, and can be "tuned" to achieve particular performance. With a high gain, the filter places more weight on the measurements, and thus follows them more closely. With a low gain, the filter follows the model predictions more closely, smoothing out noise but decreasing the responsiveness. At the extremes, a gain of one causes the filter to ignore the state estimate entirely, while a gain of zero causes the measurements to be ignored.

When performing the actual calculations for the filter (as discussed below), the state estimate and covariances are coded into matrices to handle the multiple dimensions involved in a single set of calculations. This allows for a representation of linear

relationships between different state variables (such as position, velocity, and acceleration) in any of the transition models or covariances.

As an example application, consider the problem of determining the precise location of a truck. The truck can be equipped with a GPS unit that provides an estimate of the position within a few meters. The GPS estimate is likely to be noisy; readings 'jump around' rapidly, though always remaining within a few meters of the real position. In addition, since the truck is expected to follow the laws of physics, its position can also be estimated by integrating its velocity over time, determined by keeping track of wheel revolutions and the angle of the steering wheel. This is a technique known as dead reckoning. Typically, the dead reckoning will provide a very smooth estimate of the truck's position, but it will drift over time as small errors accumulate.

In this example, the Kalman filter can be thought of as operating in two distinct phases: predict and update. In the prediction phase, the truck's old position will be modified according to the physical laws of motion (the dynamic or "state transition" model) plus any changes produced by the accelerator pedal and steering wheel. Not only will a new position estimate be calculated, but a new covariance will be calculated as well. Perhaps the covariance is proportional to the speed of the truck because we are more uncertain about the accuracy of the dead reckoning position estimate at high speeds but very certain about the position estimate when moving slowly. Next, in the update phase, a measurement of the truck's position is taken from the GPS unit. Along with this measurement comes some amount of uncertainty, and its covariance relative to that of the prediction from the previous phase determines how much the new measurement will affect the updated prediction. Ideally, if the dead reckoning estimates tend to drift away from the real position, the GPS measurement should pull the position estimate back towards the real position but not disturb it to the point of becoming rapidly changing and noisy.

2.2 DETECTING THE PEOPLE USING AGGREGATE CHANNEL FEATURES TECHNIQUE

Matlab has an inbuilt function **detectPeopleACF ()** that detects the people using the Aggregate Channel Features technique.

Syntax

Bboxes = detectPeopleACF (I)

[Bboxes, scores] = detectPeopleACF (I)

[___] = detectPeopleACF (I, roi)

[___] = detectPeopleACF (Name, Value)

Description

Bboxes = detectPeopleACF (I) returns a matrix, bboxes, that contains the locations of detected upright people in the input image, I. The locations are represented as bounding boxes. The function uses the aggregate channel features (ACF) algorithm.

[Bboxes, Scores] = detectPeopleACF (I) also returns the detection scores for each bounding box.

[___] = detectPeopleACF (I, roi) also detects people within the rectangular search region specified by roi, using either of the previous syntaxes.

[___] = detectPeopleACF (Name, Value) uses additional options specified by one or more Name, Value pair arguments. Unspecified properties have default values.

Input Arguments

I – input Image

Input image, specified as a true color image. The image must be real and non-sparse.

Roi - Rectangular search region

Rectangular search region, specified as a four-element vector, [x, y, width, height]. The roi must be fully contained in I.

Output Arguments

Bboxes - Locations of detected People.

Locations of people detected using the aggregate channel features (ACF) algorithm, returned as an M-by-4 matrix. The locations are represented as bounding boxes. Each row in bboxes contains a four-element vector, [x, y, width, height]. This vector specifies the upper-left corner and size of a bounding box, in pixels, for a detected person.

Scores – Confidence value.

Confidence value for the detections, returned as an M-by-1 vector. The vector contains a positive value for each bounding box in Bboxes. The score for each detection is the output of a soft-cascade classifier. The range of score values is [-INF -INF]. Greater scores indicate a higher confidence in the detection.

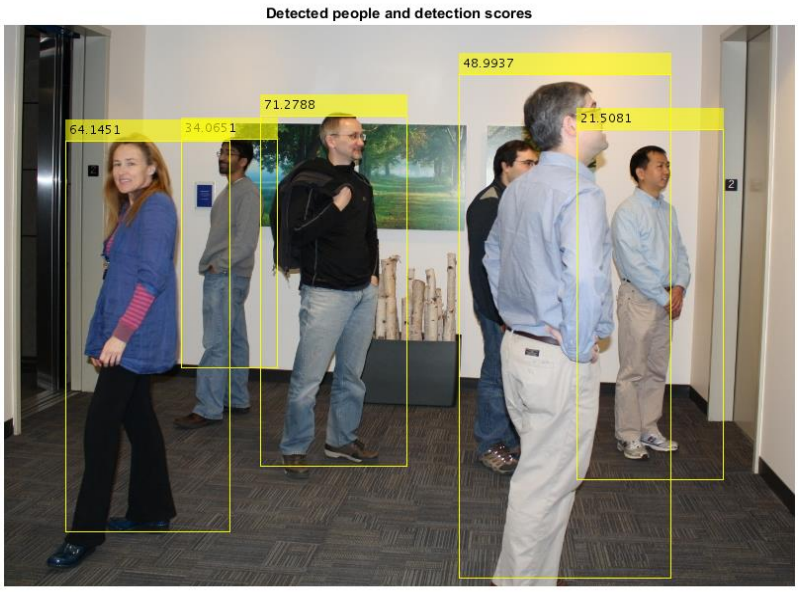

Fig 4 Detected People by aggregate channel Features Technique and the confidence scores

2.3 MUNKRES VERSION OF THE HUNGARIAN ALGORITHM

The **Hungarian method** is a combinatorial optimization algorithm that solves the assignment problem in polynomial time and which anticipated later primal-dual methods. It was developed and published in 1955 by Harold Kuhn, who gave the name "Hungarian method" because the algorithm was largely based on the earlier works of two Hungarian mathematicians: Denes Konig and Jeno Egervary.

James Munkres reviewed the algorithm in 1957 and observed that it is (strongly) polynomial. Since then the algorithm has been known also as the **Kuhn–Munkres algorithm** or **Munkres assignment algorithm**. The time complexity of the original algorithm was O ($n^{4)}$, however Edmonds and Karp, and independently Tomizawa noticed that it can be modified to achieve an O (n^3) running time. Ford and Fulkerson extended the method to general transportation problems. In 2006, it was discovered that Carl Gustav Jacobi had solved the assignment problem in the 19th century, and the solution had been published posthumously in 1890 in Latin.

AssignDetectionsToTracks function uses the Munkre's version of the Hungarian algorithm to compute an assignment which minimizes the total cost. It returns an M x 2 matrix containing the corresponding indices of assigned tracks and detections in its two columns. It also returns the indices of tracks and detections that remained unassigned.

Let C be an nxn matrix representing the costs of each of n workers to perform any of n jobs. The assignment problem is to assign jobs to workers in a way that minimizes the total cost. Since each worker can perform only one job and each job can be assigned to only one worker the assignments represent an independent set of the matrix C. One way to generate the optimal set is to create all permutations of the indexes necessary to traverse the matrix so that no row and column are used more than once.

While this approach works fine for small matrices, it does not scale. It executes in O (*n*!) time: Calculating the permutations for an *n*x*n* matrix requires *n*! Operations. For a 12x12 matrix, that's 479,001,600 traversals. Even if you could manage to perform each traversal in just one millisecond, it would still take more than 133 hours to perform the entire traversal. A 20x20 matrix would take 2,432,902,008,176,640,000 operations. At an optimistic millisecond per operation, that's more than 77 million years.

The Munkres algorithm runs in O (*n*^3) time, rather than O (*n*!). The instantiated Munkres object can be used multiple times on different matrices. The Munkres algorithm assumes that the cost matrix is square. However, it's possible to use a rectangular matrix if you first pad it with 0 values to make it square. This module automatically pads rectangular cost matrices to make them square. The cost matrix is just that: A cost matrix. The Munkres algorithm finds the combination of elements (one from each row and column) that results in the smallest cost. It's also possible to use the algorithm to maximize profit. To do that, however, you have to convert your profit matrix to a cost matrix. The simplest way to do that is to subtract all elements from a large value.

2.4 SELECTING STRONGEST BOUNDING BOXES FROM OVERLAPPING CLUSTERS

By using the Matlab function **selectStrongestBbox ()** returns selected bounding boxes that
have a high confidence score. The function uses non-maximal suppression to eliminate overlapping bounding boxes from the bbox input.

Syntax

[SelectedBbox, selectedScore] = selectStrongestBbox (bbox, score)
[SelectedBbox, selectedScore, index] = selectStrongestBbox (bbox, score)
[SelectedBbox, selectedScore, index] = selectStrongestBbox (___, Name, Value)

Description

[SelectedBbox, selectedScore] = selectStrongestBbox (bbox, score) returns selected bounding boxes that have a high confidence score. The function uses nonmaximal suppression to eliminate overlapping bounding boxes from the bbox input.

[SelectedBbox, selectedScore, index] = selectStrongestBbox (Bbox, score) additionally returns the index vector associated with selectedBbox. This vector contains the indices of the selected boxes in the bbox input.

[SelectedBbox, selectedScore, index] = selectStrongestBbox (___, Name, Value) uses additional options specified by one or more Name, Value pair arguments.

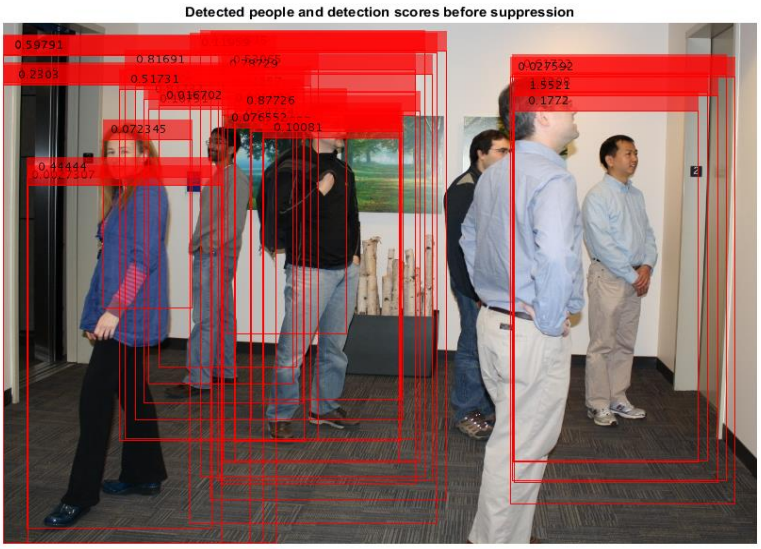

Fig 5

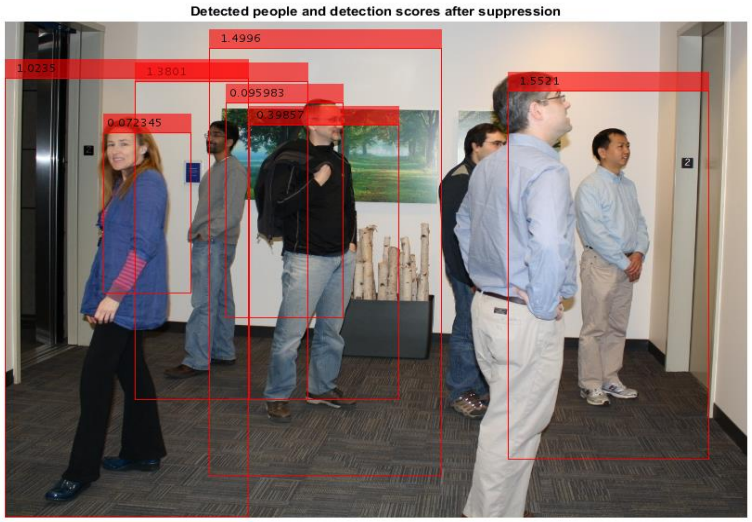

Fig 6

Input arguments

Bbox – Bounding boxes

Bounding boxes, specified as an M-by-4 matrix with M bounding boxes. Each row of the bbox input represents a bounding box, specified in the format [x y width height], where x and y correspond to the upper left corner of the bounding box. The bbox input must be real, finite, and non-sparse.

Score - Confidence Score.

Confidence score, specified as an M-by-1 vector. The Mth score in the score input corresponds to the Mth bounding box in the Bbox input. The score input must be real, finite, and non-sparse. The function uses nonmaximal suppression to eliminate overlapping bounding boxes and associate the confidence score with the boxes. A higher score represents a higher confidence in keeping the bounding box.

Output Arguments

SelectedBbox - Selected Bounding boxes

Selected bounding boxes, returned as an *M*-by-4 matrix. The selectedBbox output returns the selected bounding boxes from the Bbox input that have the highest confidence score. The function uses nonmaximal suppression to eliminate overlapping bounding boxes.

SelectedScore – Scores of selected Bounding boxes.

Scores of selected bounding boxes, returned as an *M*-by-1 vector. The *M*th score in the selectedScore output corresponds to the *M*th bounding box in the selectedBbox output.

Index – Index of selected Bounding Boxes.

Finally this function selected bounding boxes that have a high confidence score. The function uses non-maximal suppression to eliminate overlapping bounding boxes from the bbox input.

2.5 GUI (GRAPHICAL USER INTERFACE)

A graphical user interface (GUI) is a graphical display in one or more windows containing controls, called components, which enable a user to perform interactive tasks. The user of the GUI does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user of a GUI need not understand the details of how the tasks are performed. GUI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders—just to name a few. GUIs created using MATLAB tools can also perform any type of computation, read and write data files, communicate with other GUIs, and display data as tables or as plots. The following figure illustrates a simple GUI that you can easily build.

Typically, GUIs wait for an end user to manipulate a control, and then respond to each user action in turn. Each control, and the GUI itself, has one or more call-backs, named for the fact that they "call back" to MATLAB to ask it to do things. A particular user action, such as pressing a screen button, or passing the cursor over a component, triggers the execution of each call back. The GUI then responds to these events. You, as the GUI creator, write call-backs that define what the components do to handle events. This kind of programming is often referred to as event-driven programming. In event-driven programming, call back execution is asynchronous, that is, events external to the software trigger call back execution. In the case of MATLAB GUIs, most events are user interactions with the GUI, but the GUI can respond to other kinds of events as well, for example, the creation of a file or connecting a device to the computer.

You can code call-backs in two distinct ways:

•As MATLAB language functions stored in files

•As strings containing MATLAB expressions or commands (such as 'c = sqrt(a*a + b*b); 'or' print')Using functions stored in code files as call-backs is preferable to using strings, because functions have access to arguments and are more powerful and flexible. You cannot use MATLAB scripts (sequences of statements stored in code files that do not define functions) as call-backs. Although you can provide a call back with certain data and make it do anything you want, you cannot control when call-backs execute. That is,

when your GUI is being used, you have no control over the sequence of events that trigger particular call-backs or what other call-backs might still be running at those times. This distinguishes event-driven programming from other types of control flow, for example, processing sequential data files.

A MATLAB GUI is a figure window to which you add user-operated components. You can select, size, and position these components as you like using call-backs you can make the components do what you want when the user clicks or manipulates the components with keystrokes.

You can build MATLAB GUIs in two ways:

- Use GUIDE (GUI Development Environment), an interactive GUI construction kit. This approach starts with a figure that you populate with components from within a graphic layout editor. GUIDE creates an associated code file containing call-backs for the GUI and its components. GUIDE saves both the figure (as a FIG-file) and the code file. Opening either one also opens the other to run the GUI.
- Create code files that generate GUIs as functions or scripts (programmatic GUI construction). Using this approach, you create a code file that defines all component properties and behaviors. When a user executes the file, it creates a figure, populates it with components, and handles user interactions. Typically, the figure is not saved between sessions because the code in the file creates a new one each time it runs.

The code files of the two approaches look different. Programmatic GUI files are generally longer, because they explicitly define every property of the figure and its controls, as well as the call-backs. GUIDE GUIs define most of the properties within the figure itself. They store the definitions in its FIG-file rather than in its code file. The code file contains call-backs and other functions that initialize GUI when it opens.

You can create a GUI with GUIDE and then modify it programmatically. However, you cannot create a GUI programmatically and then modify it with GUIDE. The GUI-

building technique you choose depends on your experience, your preferences, and the kind of application you need the GUI to operate.

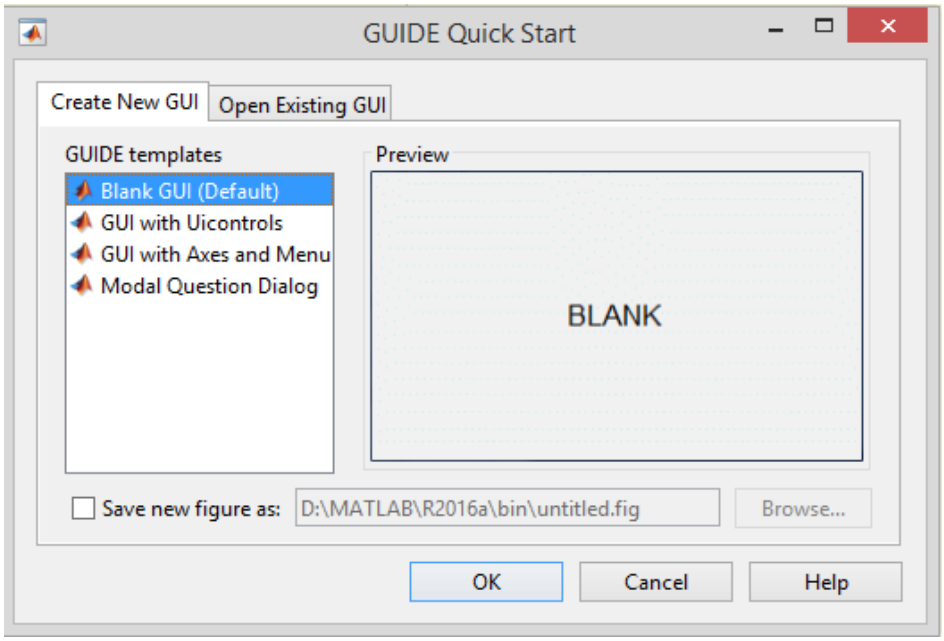

Fig 7 Dialogue box to select the Blank GUI

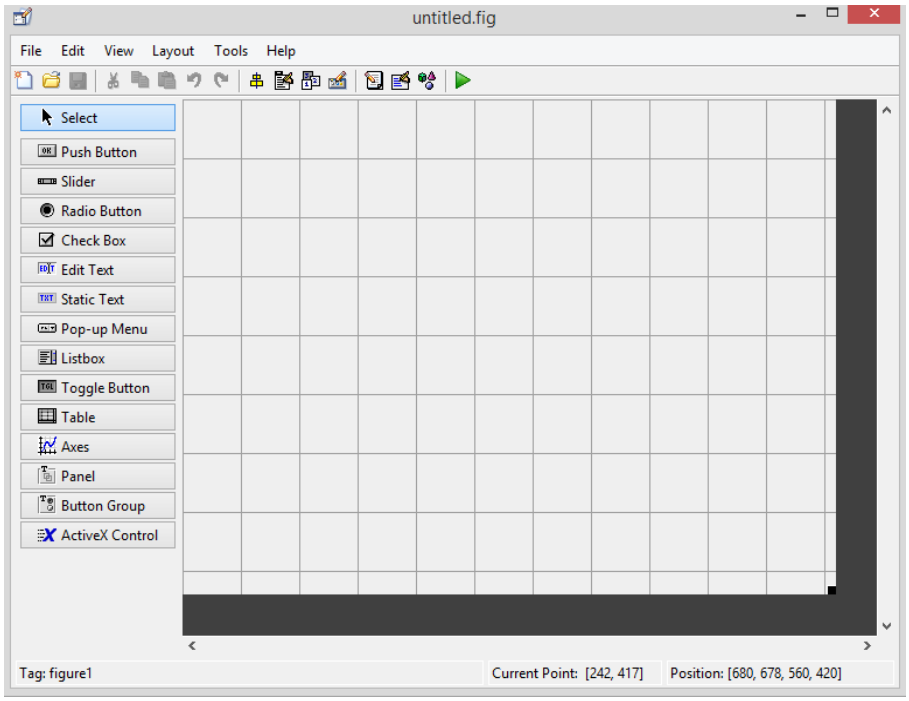

Fig 8 GUI Developing Environment (GUIDE)

CHAPTER 3: IMPLEMENTATION AND RESULTS

3.1 INTRODUCTION TO MATLAB

The name MATLAB stands for MATRIX LABORATORY. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research. MATLAB has many advantages compared to conventional computer languages (e.g., C, FORTRAN) for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide. It has powerful built-in routines that enable a very wide variety of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

There are various tools in Matlab that can be utilized for image processing, such as Simulink, GUI etc. Simulink contains various toolboxes and image processing toolbox is one such example. Simulink is used for simulation of various projects. GUI is another important tool in Matlab. It can be designed either by manual programming which is tedious task or by using guide. GUI is explained in next section.

3.2 IMPLEMENTATION IN MATLAB

This example shows how to perform automatic detection and tracking of people in a video from a moving camera. It demonstrates the flexibility of a tracking system adapted to a moving camera, which is ideal for automotive safety applications. Unlike the stationary camera example, The Motion-Based Multiple Object Tracking, this example contains several additional algorithmic steps. These steps include people detection, customized non-maximum suppression, and heuristics to identify and eliminate false alarm tracks.

First we have to prepare the GUI (Graphical User Interface).
GUI is prepared using GUIDE tools in Matlab**.**

3.2.1 Developing GUI

1. Open a new UI in the GUIDE layout Editor
2. In the GUIDE Quick Start dialog box, select the Blank (Default) template, and then click OK.
3. Display the names of the components in the component palette.
4. Set the window size in GUIDE.
5. Layout the UI: - Add, align, and label the components in the UI.
6. Save the Layout.

After saving the layout, GUIDE creates two files, a FIG-file and a code file. The FIG-file, with extension .fig, is a binary file that contains a description of the layout. The code file, with extension .m, contains MATLAB functions that control the app's behavior.

To run an app created with GUIDE without opening GUIDE, execute its code file by typing its name.

Guidata

Guidata (object_handle, data)
Data = guidata (object_handle)
Guidata (object_handle, data) stores the variable data with the object specified by object_handle. If object_handle is not a figure, then the object's parent figure is used. Data can be any MATLAB variable, but is typically a structure, which enables you to add new fields as required.
Guidata can manage only one variable at any time. Subsequent calls to guidata (object_handle, data) overwrite the previously stored data.
To change the data managed by guidata:
1. Get a copy of the data with the command data = guidata (object_handle).
2. Make the desired changes to data.
3. Save the changed version of data with the command guidata (object_handle, data).
Guidata provides application developers with a convenient interface to a figure's application data.

Using Guidata in GUIDE

Notice that you use the input argument hObject in place of gcbo to refer to the object whose callback is executing.
If you use GUIDE, you do not need to call guihandles to create a structure. You can add your own data to the handles structure. For example, this code adds the field, numberOfErrors, to the structure:
Suppose you needed to access the numberOfErrors field in a push button callback. Your callback code now looks something like this:

```
% --- Executes on button press in pushbutton1.
Function my_GUIDE_GUI_pushbutton1_Callback (hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% ...
% No need to call guidata to obtain a structure;
% it is provided by GUIDE via the handles argument
handles.numberOfErrors = handles.numberOfErrors + 1;
% save the changes to the structure
Guidata (hObject, handles)
```

Like these for each button in the GUI the Call back functions get called and get executed and update the hObject.

Hence by using the GUIDE, the algorithm works and the count will be updated real-time.

The developed GUI is shown below:

Fig 9 Output GUI

3.2.2 Steps involved in tracking the human objects on the streets

Auxiliary Input and Global Parameters of the Tracking System

This tracking system requires a data file that contains information that relates the pixel location in the image to the size of the bounding box marking the pedestrian's location. This prior knowledge is stored in a vector pedScaleTable. The n-th entry in pedScaleTable represents the estimated height of an adult person in pixels. The index n references the approximate Y-coordinate of the pedestrian's feet.

To obtain such a vector, a collection of training images were taken from the same viewpoint and in a similar scene to the testing environment. The training images contained images of pedestrians at varying distances from the camera. Using the trainingImageLabeler app, bounding boxes of the pedestrians in the images were manually annotated. The height of the bounding boxes together with the location of the pedestrians in the image were used to generate the scale data file through regression. Here is a helper function to show the algorithmic steps to fit the linear regression model: helperTableOfScales.m

There is also a set of global parameters that can be tuned to optimize the tracking performance. You can use the descriptions below to learn out how these parameters affect the tracking performance.

- **ROI:** Region-Of-Interest in the form of [x, y, w, h]. It limits the processing area to ground locations.
- **cThresh**: Tolerance threshold for scale estimation. When the difference between the detected scale and the expected scale exceeds the tolerance, the candidate detection is considered to be unrealistic and is removed from the output.
- **GatingThresh:** Gating parameter for the distance measure. When the cost of matching the detected bounding box and the predicted bounding box exceeds the threshold, the system removes the association of the two bounding boxes from tracking consideration.
- **GatingCost:** Value for the assignment cost matrix to discourage the possible tracking to detection assignment.
- **CostOfNonAssignment:** Value for the assignment cost matrix for not assigning a detection or a track. Setting it too low increases the likelihood of creating a new track,

and may result in track fragmentation. Setting it too high may result in a single track corresponding to a series of separate moving objects.

- **TimeWindowSize:** Number of frames required to estimate the confidence of the track.
- **ConfidenceThresh:** Confidence threshold to determine if the track is a true positive.
- **AgeThresh:** Minimum length of a track being a true positive.
- **VisThresh:** Minimum visibility threshold to determine if the track is a true positive.

Create System Objects for the Tracking System Initialization

The **setupSystemObjects** function creates system objects used for reading and displaying the video frames and loads the scale data file.

The **pedScaleTable** vector, which is stored in the scale data file, encodes our prior knowledge of the target and the scene. Once you have the regressor trained from your samples, you can compute the expected height at every possible Y-position in the image. These values are stored in the vector. The n-th entry in pedScaleTable represents our estimated height of an adult person in pixels. The index n references the approximate Y-coordinate of the pedestrian's feet.

Initialize Tracks

The initializeTracks function creates an array of tracks, where each track is a structure representing a moving object in the video. The purpose of the structure is to maintain the state of a tracked object. The state consists of information used for detection-to-track assignment, track termination, and display.

The structure contains the following fields:

- **Id:** An integer ID of the track.
- **Color:** The color of the track for display purpose.
- **Bboxes:** AN N-by-4 matrix to represent the bounding boxes of the object with the current box at the last row. Each row has a form of [x, y, width, height].
- **Scores:** An N-by-1 vector to record the classification score from the person detector with the current detection score at the last row.
- **KalmanFilter:** A Kalman filter object used for motion-based tracking. We track the center point of the object in image;

- **Age:** The number of frames since the track was initialized.
- **TotalVisibleCount:** The total number of frames in which the object was detected (visible).
- **Confidence:** A pair of two numbers to represent how confident we trust the track. It stores the maximum and the average detection scores in the past within a predefined time window.
- **PredPosition:** The predicted bounding box in the next frame.

Read a Video Frame

Read the next video frame from the video file.

Detect People

The detectPeople function returns the centroids, the bounding boxes, and the classification scores of the detected people. It performs filtering and non-maximum suppression on the raw output of detectPeopleACF.

Centroids: An N-by-2 matrix with each row in the form of [x, y].

Bboxes: An N-by-4 matrix with each row in the form of [x, y, width, height].

Scores: An N-by-1 vector with each element is the classification score at the corresponding frame.

Predict New Locations of Existing Tracks

Use the Kalman filter to predict the centroid of each track in the current frame, and update its bounding box accordingly. We take the width and height of the bounding box in previous frame as our current prediction of the size.

Assign Detections to Tracks

Assigning object detections in the current frame to existing tracks is done by minimizing cost. The cost is computed using the bboxOverlapRatio function, and is the overlap ratio between the predicted bounding box and the detected bounding box. In this example, we assume the person will move gradually in consecutive frames due to the high frame rate of the video and the low motion speed of a person.

The algorithm involves two steps:

Step 1: Compute the cost of assigning every detection to each track using the bboxOverlapRatio measure. As people move towards or away from the camera, their motion will not be accurately described by the centroid point alone. The cost takes into account the distance on the image plane as well as the scale of the bounding boxes. This prevents assigning detections far away from the camera to tracks closer to the camera, even if their centroids coincide. The choice of this cost function will ease the computation without resorting to a more sophisticated dynamic model. The results are stored in an MxN matrix, where M is the number of tracks, and N is the number of detections.

Step 2: Solve the assignment problem represented by the cost matrix using the assignDetectionsToTracks function. The function takes the cost matrix and the cost of not assigning any detections to a track.

The value for the cost of not assigning a detection to a track depends on the range of values returned by the cost function. This value must be tuned experimentally. Setting it too low increases the likelihood of creating a new track, and may result in track fragmentation. Setting it too high may result in a single track corresponding to a series of separate moving objects.

The assignDetectionsToTracks function uses the Munkres' version of the Hungarian algorithm to compute an assignment which minimizes the total cost. It returns an M x 2 matrix containing the corresponding indices of assigned tracks and detections in its two columns. It also returns the indices of tracks and detections that remained unassigned.

Update Assigned Tracks

The updateAssignedTracks function updates each assigned track with the corresponding detection. It calls the correct method of vision.KalmanFilter to correct the location estimate. Next, it stores the new bounding box by taking the average of the size of recent (up to) 4 boxes, and increases the age of the track and the total visible count by 1. Finally, the function adjusts our confidence score for the track based on the previous detection scores.

Update Unassigned Tracks

The updateUnassignedTracks function marks each unassigned track as invisible, increases its age by 1, and appends the predicted bounding box to the track. The confidence is set to zero since we are not sure why it was not assigned to a track.

Delete Lost Tracks

The deleteLostTracks function deletes tracks that have been invisible for too many consecutive frames. It also deletes recently created tracks that have been invisible for many frames overall.

Noisy detections tend to result in creation of false tracks. For this example, we remove a track under following conditions:

- The object was tracked for a short time. This typically happens when a false detection shows up for a few frames and a track was initiated for it.
- The track was marked invisible for most of the frames.
- It failed to receive a strong detection within the past few frames, which is expressed as the maximum detection confidence score.

Create New Tracks

Create new tracks from unassigned detections. Assume that any unassigned detection is a start of a new track. In practice, you can use other cues to eliminate noisy detections, such as size, location, or appearance.

Display Tracking Results

The displayTrackingResults function draws a colored bounding box for each track on the video frame. The level of transparency of the box together with the displayed score indicate the confidence of the detections and tracks.

3.3 SIMULATION AND RESULTS

To test this program, follow the steps given below:

1. Run the program. A GUI will appear, as shown in Figure.

2. Click on Load button to load the video into the interface.

3. Next, click on Process button and then a new window will open and tracks the people in the frame and display the results.

4. To exit from the interface, click the exit button.

Simulation Results

The output of GUI given below clearly indicates the implemented results of the algorithm designed. The tracked people are annotated with the bounding boxes. Each Bounding box has confidence score based on the visibility of the track in the past TimeSizeWindow. If the new tracks are detected in the frame, then new tracks are created and the tracker tracks them. Due to noise and the turbulences in the video, some false alarm tracks are visible in the video for short durations. These are eliminated in few frames based in the confidence score of those tracks.

The below figures are the output window screenshots of the processed video. The video loaded here is taken from a camera attached to a motor cycle in the morning time.

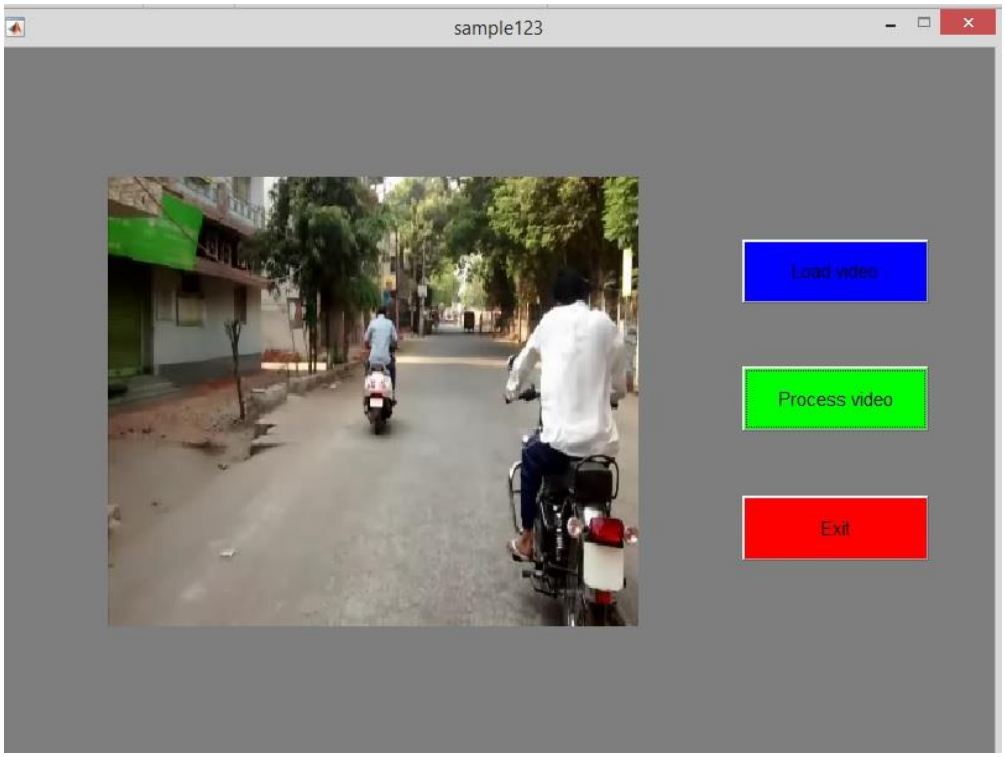

Fig 10 Video loaded into the interface

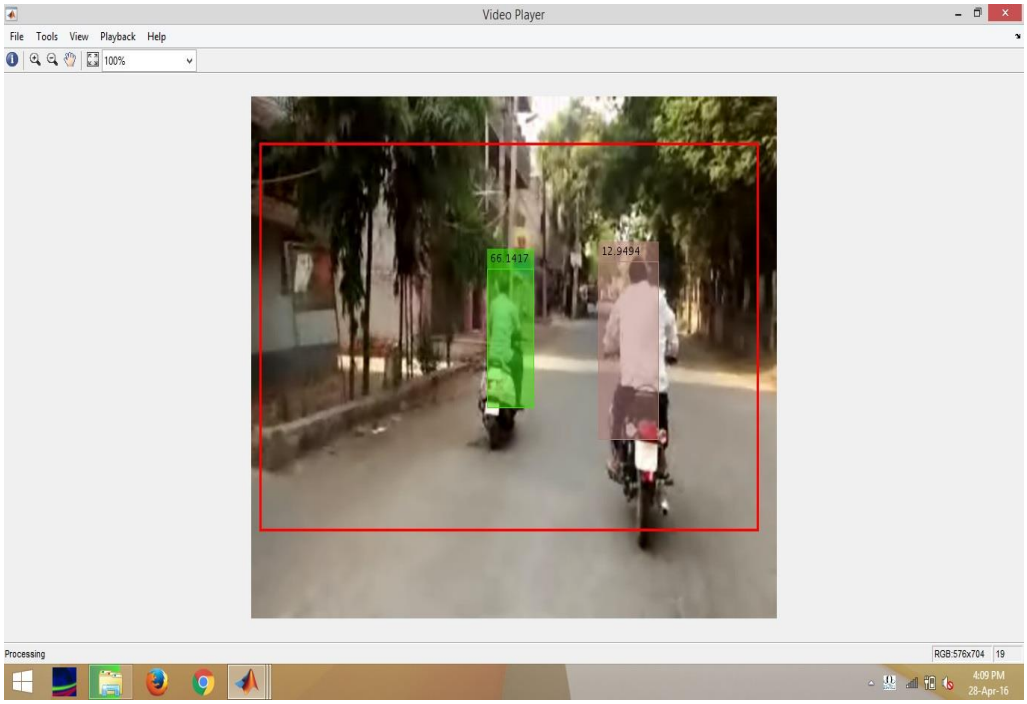

Fig 11 Output Window screenshot 1

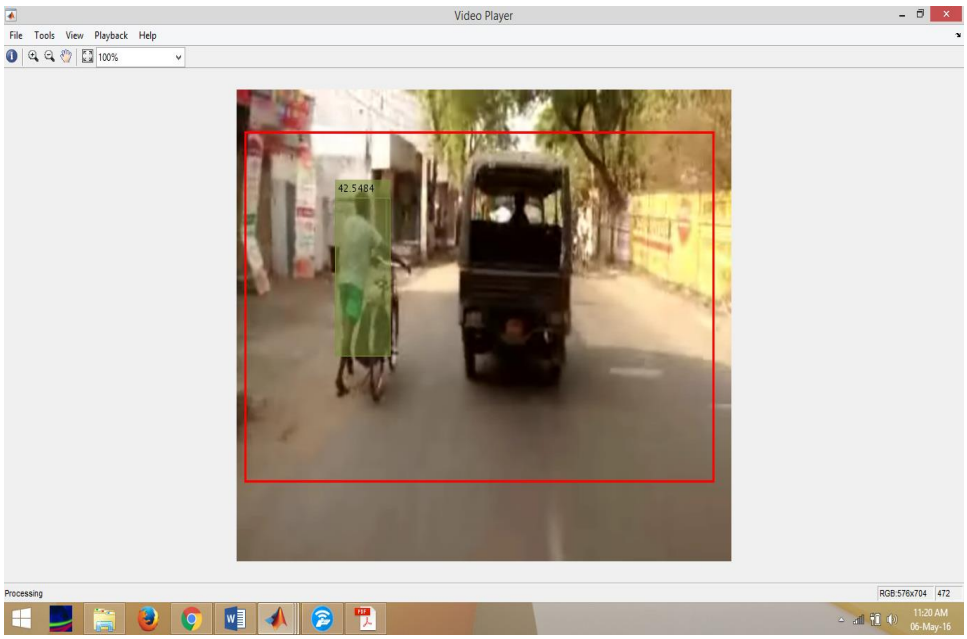

Fig 12 Output window screen shot 2

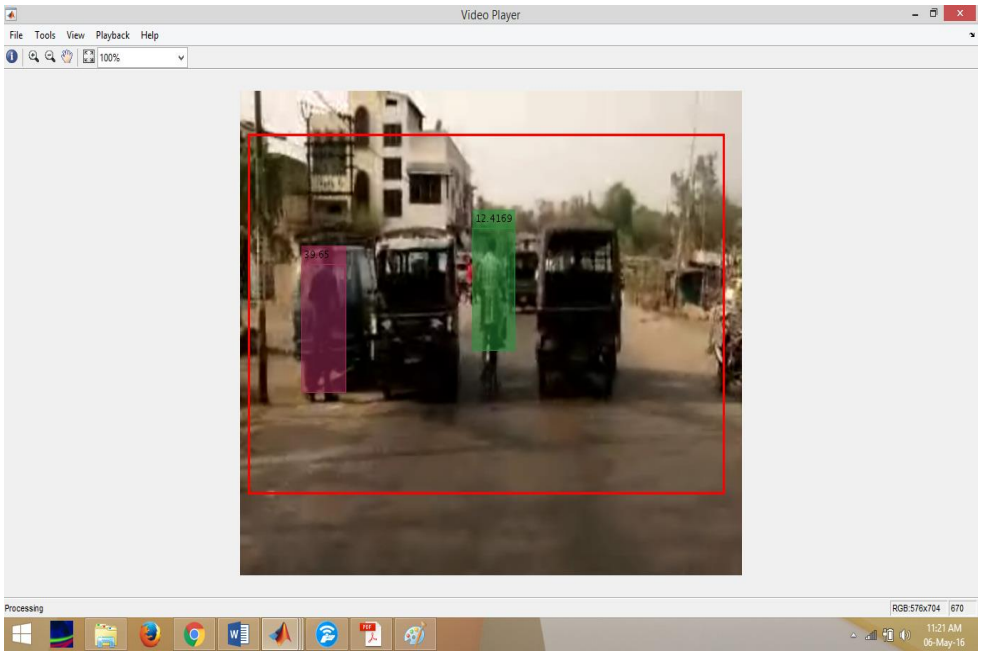

Fig 13 Output window screen shot 3

CHAPTER 4: APPLICATIONS, CONCLUSION AND FUTURE WORK

4.1 APPLICATIONS

There are many applications of "Human object detection and tracking on streets". Some of them are mentioned below.

- It will improve the Road safety by providing more information about the human objects (pedestrians) to the people travelling on the road. If the Human objects are highlighted and tracked, the people who are travelling on the roads can easily recognize them. So the accidents are reduced by implementing this in vehicles.
- To analyze the traffic data by discriminating two wheelers and four wheelers on the road. We can also count the number of two wheelers and four wheelers on the road by extending this algorithm to track four wheelers and two wheelers instead of tracking humans. So, by taking the footage from the CC Camera present at the traffic signal, we can analyze the traffic data and count the percentage of traffic shared between two wheelers and four wheelers.

- In poor climatic conditions like heavy rainfall, fog, snow the people travelling on the roads cannot clearly recognize the human objects. Hence by implementing this in vehicles, they can recognize the people travelling on the roads.
- Due to huge crowd gatherings in the road like during the rallies, there is a lot of scope for traffic Jam. Hence by recognizing the crowd gatherings in an area we can inform the same to the people travelling in those roads in advance about the huge crowd gatherings so that they can choose an alternate way for travelling. By this traffic Jams can be controlled by sending the traffic analysis data to the people travelling on the roads.
- To help the drivers with poor eye sight for recognizing the pedestrians. In the morning time and the evening time the light is poor. So it is difficult to recognize the pedestrians crossing on the streets. So, this helps in recognizing the pedestrians and help the people travelling on the streets. Some drivers have poor vision so that even in the day times they find difficult to recognize the pedestrians crossing the roads. So, by implementing this, they can be able to recognize the pedestrians and avoid accidents.

4.2 CONCLUSION

Detecting the objects in the video and tracking its motion to identify its characteristics has been emerging as a demanding research area in the domain of image processing and computer vision. Most of the methods include object segmentation using background subtraction. The tracking strategies use different methodologies like Mean-shift, Kalman filter, Particle filter etc. The performance of the tracking methods vary with respect to background information.

The objective of this project is to improve the road safety by providing more information about the human objects (pedestrians) to the people travelling on the streets, to help the drivers with poor eye sight for recognizing the pedestrians and to analyze the traffic data by discriminating two wheelers and four wheelers on the road.

"Human object detection using MATLAB" technique that we propose overcomes all the limitations of the earlier and in use techniques. We have also discussed all the techniques used in our project implementation. This project is very useful for implementation on large scale because the equipment and tools required are very minimal. This project when implemented with good quality equipment like cameras of good resolution would give very good results as considered in terms image processing.

We recorded a video in the traffic of Jamshedpur and processed in Matlab. We got good results. Every person coming into the frame is detected and highlighted with a bounding box while tracking them. The tracks which are having Confidence Score less than the confidence Threshold are eliminated. Examples of these tracks are noise and turbulences in the surroundings. We can also define the Region of Interest which is a rectangular shape from which the pixels are processed.

The video we used here is of resolution 320*240 pixels. If we process the video captured in high Resolution in good lighting conditions, we can get better results. We also captured a video in the Morning time where the light is poor and processed that video. We achieved very good results and all the human objects moving on the streets are tracked.

Instead of using recorded video, if we use live video by attaching the camera to the car, the pedestrians are highlighted and it helps the drivers for recognizing them in dim lights and for having an enhanced view of the street.

4.3 FUTURE WORK

The project we implemented will take a recorded video as input and do the tracking of human objects. We can also use this recorded video to analyze the traffic in that area. The video we used here is of resolution 320*240 pixels. If we analyze the video captured in high Resolution in good lighting conditions, we can get better results. We will also get the accurate crowd analysis by using high resolution cameras for capturing videos. Here we are giving a recorded video input to the GUI we developed. If we use a live video as input by attaching the camera to the moving car, the pedestrians are highlighted and it helps the

drivers for recognizing them in dim lights and for having an enhanced view of the street. This also helps the people travelling on the streets who are having poor eye sight by detecting pedestrians even in climatic conditions like heavy rainfall, fog, snow. We are working on installing this application in a moving car by giving the car camera input to it. Then it can be able to process the live video and giving the tracking results to the driver. We can also give the output results of this application to the goggles. If the driver wears these goggles while driving, they can be able to see the tracked results. So, it helps the drivers. Till now this application will only detect the human objects. By extending the algorithm we used in this project, we can detect four wheelers or two wheelers instead of detecting human objects. By this we can analyze the traffic in an area and count the percentage traffic shared by four wheelers or two wheelers. If we send this traffic data in advance to the people travelling on the same road, this helps in controlling the traffic jams which occur frequently in cities.

REFERENCES

[1] Rafael C. Gonzalez and Richard E. Woods., “Digital Image processing”,3rd edition,pearson,2009.

[2] Dollar, C. Wojeck, B. Shiele, and P. Perona. "Pedestrian detection: An evaluation of the state of the art." Pattern Analysis and Machine Intelligence, IEEE Transactions.Vol. 34, Issue 4, 2012, pp. 743–761.

[3] Dollar, C., Wojeck, B. Shiele, and P. Perona. "Pedestrian detection: A benchmark." IEEE Conference on Computer Vision and Pattern Recognition. 2009.

[4] Pedro F. Felzenszwalb, “Object Detection with Discriminatively Trained Part-Based Models”, IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume:32 , Issue: 9) ,2009.

[5] Chenglong Yu, Xuan Wang “Pedestrian detection based on combinational holistic and partial features “, Machine Learning and Cybernetics (ICMLC), 2011 International Conference on (Volume:4),2011.

[6] Wanjun Jin , Lide Wu “Patch-based natural object detection using CF*IRF “,Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on (Volume:3),2004

[7] D. Serby, E. K. Meier, and L. V. Gool, "Probabilistic Object Tracking Using Multiple Features", IEEE Proc. of International Conf on Pattern Recognition Intelligent Transportation Systems, Vol. 6, pp. 43- 53, 2004.

[8] L. Li, S. Ranganath, H. Weimin, and K. Sengupta, "Framework for Real-Time Behavior Interpretation From Traffic Video", IEEE Tran. On Intelligen Transportation Systems, , Vol. 6, No. 1, pp. 43-53, 2005.

[9] P. Kumar, H. Weimin, I. U. Gu, and Q. Tian, "Statistical Modeling of Complex Backgrounds for Foreground Object Detection", IEEE Trans. On Image Processing, Vol. 13, No. 11, pp. 43-53, November 2004.

[10] Z Zivkovi, "Improving the selection of feature points for tracking", In Pattern Analysis and Applications, vol.7, no. 2, Copyright Springer-Verlag London Limited, 2004.

[11] J. Lou, T. Tan, W. Hu, H. Yang, and S. J. Maybank, "3D Model-Based Vehicle Tracking", IEEE Trans. on Image Processing, Vol. 14, pp. 1561-1569, October 2005.

[12] Faliu Yi, Inkyu Moon," Image Segmentation: A Survey of Graph-cut Methods",International Conference on Systems and Informatics (ICSAI 2012), 2012.

[13] Shi, J. And Malik, J. 2000. Normalized cuts and image segmentation. IEEE Trans. Patt. Analy. Mach. Intell. 22, 8, pp.888–905.

[14] Viola, P., Jones, M., And Snow, D. 2003. Detecting pedestrians using patterns of motion and appearance. In IEEE International Conference on Computer Vision (ICCV). pp.734–741.

[15] Papageorgiou, C., Oren, M., And Poggio, T. “A general framework for object detection”. In IEEE International Conference on Computer Vision (ICCV). pp.555–562, 1998.

[16] T. Chen, ‘Object Tracking Based on Active Contour Model by Neural Fuzzy Network’, IITA
International Conference on Control Automation and Systems Engineering, pp. 570-574, 2009.

APPENDIX

Matlab Program

```
function varargout = sample123(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @sample123_OpeningFcn, ...
                   'gui_OutputFcn',  @sample123_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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



% --- Executes just before sample123 is made visible.

function sample123_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to sample123 (see VARARGIN)

% Choose default command line output for sample123
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes sample123 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
end
```

```
% --- Outputs from this function are returned to the command line.
function varargout = sample123_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

end



% --- Executes on button press in pushbutton2.

function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global filename;
[filename, ~] = ...
   uigetfile({'*.mp4';'*.mpg';'*.avi';'*.*'},'File Selector');
v = VideoReader(filename);
ima = read(v,1);
axes(handles.axes3);
imshow(ima);
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)
global filename
videoFile      = filename;
scaleDataFile   = 'pedScaleTable.mat'; % An auxiliary file that helps to determine the size
of a pedestrian at different pixel locations.

obj = setupSystemObjects(videoFile, scaleDataFile);
```

```
% Create an empty array of tracks.
tracks = initializeTracks();

% ID of the next track.
nextId = 1;

% Set the global parameters.
option.ROI                 = [10 40 500 320 ];
% A rectangle [x, y, w, h] that limits the processing area to ground locations.
option.scThresh            = 0.3;
% A threshold to control the tolerance of error in estimating the scale of a detected
pedestrian.
option.gatingThresh        = 0.9;
% A threshold to reject a candidate match between a detection and a track.
option.gatingCost          = 100;
% A large value for the assignment cost matrix that enforces the rejection of a candidate
match.
option.costOfNonAssignment  = 10;
% A tuning parameter to control the likelihood of creation of a new track.
option.timeWindowSize      = 16;
% A tuning parameter to specify the number of frames required to stabilize the
confidence score of a track.
option.confidenceThresh    = 2;
% A threshold to determine if a track is true positive or false alarm.
option.ageThresh           = 8;
% A threshold to determine the minimum length required for a track being true positive.
option.visThresh           = 0.6;
% A threshold to determine the minimum visibility value for a track being true positive.

% Detect people and track them across video frames.

cont = ~isDone(obj.reader);
while cont
   frame   = readFrame();

   [centroids, bboxes, scores] = detectPeople();

   predictNewLocationsOfTracks();

   [assignments, unassignedTracks, unassignedDetections] = ...
      detectionToTrackAssignment();

   updateAssignedTracks();
   updateUnassignedTracks();
   deleteLostTracks();
   createNewTracks();
```

```
    displayTrackingResults();

    % Exit the loop if the video player figure is closed by user.
    cont = ~isDone(obj.reader) && isOpen(obj.videoPlayer);
end



%% Auxiliary Input and Global Parameters of the Tracking System

    function obj = setupSystemObjects(videoFile,scaleDataFile)
        % Initialize Video I/O
        % Create objects for reading a video from a file, drawing the
        % detected and tracked people in each frame, and playing the video.

        % Create a video file reader.
        obj.reader = vision.VideoFileReader(videoFile, 'VideoOutputDataType', 'uint8');

        % Create a video player.
        obj.videoPlayer = vision.VideoPlayer('Position', [29, 597, 643, 386]);

        % Load the scale data file
        ld = load(scaleDataFile, 'pedScaleTable');
        obj.pedScaleTable = ld.pedScaleTable;
    end



%% Initialize Tracks

    function tracks = initializeTracks()
        % Create an empty array of tracks
        tracks = struct(...
            'id', {}, ...
            'color', {}, ...
            'bboxes', {}, ...
            'scores', {}, ...
            'kalmanFilter', {}, ...
            'age', {}, ...
            'totalVisibleCount', {}, ...
            'confidence', {}, ...
            'predPosition', {});
    end

%% Read a Video Frame
```

```
% Read the next video frame from the video file.
   function frame = readFrame()
      frame = step(obj.reader);
   end

%% Detect People

   function [centroids, bboxes, scores] = detectPeople()
      % Resize the image to increase the resolution of the pedestrian.
      % This helps detect people further away from the camera.
      resizeRatio = 1.5;
      frame = imresize(frame, resizeRatio, 'Antialiasing',false);

      % Run ACF people detector within a region of interest to produce
      % detection candidates.
      [bboxes, scores] = detectPeopleACF(frame, option.ROI, ...
         'Model','caltech',...
         'WindowStride', 2,...
         'NumScaleLevels', 4, ...
         'SelectStrongest', false);

      % Look up the estimated height of a pedestrian based on location of their feet.
      height = bboxes(:, 4) / resizeRatio;
      y = (bboxes(:,2)-1) / resizeRatio + 1;
      yfoot = min(length(obj.pedScaleTable), round(y + height));
      estHeight = obj.pedScaleTable(yfoot);

      % Remove detections whose size deviates from the expected size,
      % provided by the calibrated scale estimation.
      invalid = abs(estHeight-height)>estHeight*option.scThresh;
      bboxes(invalid, :) = [];
      scores(invalid, :) = [];

      % Apply non-maximum suppression to select the strongest bounding boxes.
      [bboxes, scores] = selectStrongestBbox(bboxes, scores, ...
                  'RatioType', 'Min', 'OverlapThreshold', 0.6);

      % Compute the centroids
      if isempty(bboxes)
         centroids = [];
      else
         centroids = [(bboxes(:, 1) + bboxes(:, 3) / 2), ...
            (bboxes(:, 2) + bboxes(:, 4) / 2)];
      end
```

```
    end

%% Predict New Locations of Existing Tracks

    function predictNewLocationsOfTracks()
        for i = 1:length(tracks)
            % Get the last bounding box on this track.
            bbox = tracks(i).bboxes(end, :);

            % Predict the current location of the track.
            predictedCentroid = predict(tracks(i).kalmanFilter);

            % Shift the bounding box so that its center is at the predicted location.
            tracks(i).predPosition = [predictedCentroid - bbox(3:4)/2, bbox(3:4)];
        end
    end


%% Assign Detections to Tracks

    function [assignments, unassignedTracks, unassignedDetections] = ...
            detectionToTrackAssignment()

        % Compute the overlap ratio between the predicted boxes and the
        % detected boxes, and compute the cost of assigning each detection
        % to each track. The cost is minimum when the predicted bbox is
        % perfectly aligned with the detected bbox (overlap ratio is one)
        predBboxes = reshape([tracks(:).predPosition], 4, [])';
        cost = 1 - bboxOverlapRatio(predBboxes, bboxes);

        % Force the optimization step to ignore some matches by
        % setting the associated cost to be a large number. Note that this
        % number is different from the 'costOfNonAssignment' below.
        % This is useful when gating (removing unrealistic matches)
        % technique is applied.
        cost(cost > option.gatingThresh) = 1 + option.gatingCost;

        % Solve the assignment problem.
        [assignments, unassignedTracks, unassignedDetections] = ...
            assignDetectionsToTracks(cost, option.costOfNonAssignment);
    end


%% Update Assigned Tracks

    function updateAssignedTracks()
```

```
        numAssignedTracks = size(assignments, 1);
        for i = 1:numAssignedTracks
            trackIdx = assignments(i, 1);
            detectionIdx = assignments(i, 2);

            centroid = centroids(detectionIdx, :);
            bbox = bboxes(detectionIdx, :);

            % Correct the estimate of the object's location
            % using the new detection.
            correct(tracks(trackIdx).kalmanFilter, centroid);

            % Stabilize the bounding box by taking the average of the size
            % of recent (up to) 4 boxes on the track.
            T = min(size(tracks(trackIdx).bboxes,1), 4);
            w = mean([tracks(trackIdx).bboxes(end-T+1:end, 3); bbox(3)]);
            h = mean([tracks(trackIdx).bboxes(end-T+1:end, 4); bbox(4)]);
            tracks(trackIdx).bboxes(end+1, :) = [centroid - [w, h]/2, w, h];

            % Update track's age.
            tracks(trackIdx).age = tracks(trackIdx).age + 1;

            % Update track's score history
            tracks(trackIdx).scores = [tracks(trackIdx).scores; scores(detectionIdx)];

            % Update visibility.
            tracks(trackIdx).totalVisibleCount = ...
                tracks(trackIdx).totalVisibleCount + 1;

            % Adjust track confidence score based on the maximum detection
            % score in the past 'timeWindowSize' frames.
            T = min(option.timeWindowSize, length(tracks(trackIdx).scores));
            score = tracks(trackIdx).scores(end-T+1:end);
            tracks(trackIdx).confidence = [max(score), mean(score)];
        end
    end


%% Update Unassigned Tracks

    function updateUnassignedTracks()
        for i = 1:length(unassignedTracks)
            idx = unassignedTracks(i);
            tracks(idx).age = tracks(idx).age + 1;
            tracks(idx).bboxes = [tracks(idx).bboxes; tracks(idx).predPosition];
            tracks(idx).scores = [tracks(idx).scores; 0];
```

```
            % Adjust track confidence score based on the maximum detection
            % score in the past 'timeWindowSize' frames
            T = min(option.timeWindowSize, length(tracks(idx).scores));
            score = tracks(idx).scores(end-T+1:end);
            tracks(idx).confidence = [max(score), mean(score)];
        end
    end


%% Delete Lost Tracks

    function deleteLostTracks()
        if isempty(tracks)
            return;
        end

        % Compute the fraction of the track's age for which it was visible.
        ages = [tracks(:).age]';
        totalVisibleCounts = [tracks(:).totalVisibleCount]';
        visibility = totalVisibleCounts ./ ages;

        % Check the maximum detection confidence score.
        confidence = reshape([tracks(:).confidence], 2, [])';
        maxConfidence = confidence(:, 1);

        % Find the indices of 'lost' tracks.
        lostInds = (ages <= option.ageThresh & visibility <= option.visThresh) | ...
            (maxConfidence <= option.confidenceThresh);

        % Delete lost tracks.
        tracks = tracks(~lostInds);
    end


%% Create New Tracks

    function createNewTracks()
        unassignedCentroids = centroids(unassignedDetections, :);
        unassignedBboxes = bboxes(unassignedDetections, :);
        unassignedScores = scores(unassignedDetections);

        for i = 1:size(unassignedBboxes, 1)
            centroid = unassignedCentroids(i,:);
            bbox = unassignedBboxes(i, :);
            score = unassignedScores(i);
```

```
            % Create a Kalman filter object.
            kalmanFilter = configureKalmanFilter('ConstantVelocity', ...
               centroid, [2, 1], [5, 5], 100);

            % Create a new track.
            newTrack = struct(...
               'id', nextId, ...
               'color', 255*rand(1,3), ...
               'bboxes', bbox, ...
               'scores', score, ...
               'kalmanFilter', kalmanFilter, ...
               'age', 1, ...
               'totalVisibleCount', 1, ...
               'confidence', [score, score], ...
               'predPosition', bbox);

            % Add it to the array of tracks.
            tracks(end + 1) = newTrack; %#ok

            % Increment the next id.
            nextId = nextId + 1;
         end
      end

      function displayTrackingResults()

         displayRatio = 4/3;
         frame = imresize(frame, displayRatio);

         if ~isempty(tracks),
            ages = [tracks(:).age]';
            confidence = reshape([tracks(:).confidence], 2, [])';
            maxConfidence = confidence(:, 1);
            avgConfidence = confidence(:, 2);
            opacity = min(0.5,max(0.1,avgConfidence/3));
            noDispInds = (ages < option.ageThresh & maxConfidence <
option.confidenceThresh) | ...
                  (ages < option.ageThresh / 2);

            for i = 1:length(tracks)
               if ~noDispInds(i)

                  % scale bounding boxes for display
                  bb = tracks(i).bboxes(end, :);
                  bb(:,1:2) = (bb(:,1:2)-1)*displayRatio + 1;
```

```
                bb(:,3:4) = bb(:,3:4) * displayRatio;


                frame = insertShape(frame, ...
                                'FilledRectangle', bb, ...
                                'Color', tracks(i).color, ...
                                'Opacity', opacity(i));
                frame = insertObjectAnnotation(frame, ...
                                'rectangle', bb, ...
                                num2str(avgConfidence(i)), ...
                                'Color', tracks(i).color);
            end
        end
    end

    frame = insertShape(frame, 'Rectangle', option.ROI * displayRatio, ...
                    'Color', [255, 0, 0], 'LineWidth', 3);

    step(obj.videoPlayer, frame);

  end
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)
close(handles.figure1);
end
```