CHAPTER 1

INTRODUCTION

The detection of stationary or moving targets, and tracking them on a real-time video streams is a very important and challenging task in order to protect fields from enemies. The enemies can be a human, an animal or even an object. The field is secures by drones or stationary sticks which include detection and tracking system. Video surveillance is a very popular research topic in computer vision applications that continuously tries to detect, recognize and track the targets. Object detection compromises located objects in the frame of a video sequence. Every tracking method requires a detection method in every single frame. Object tracking is the process of following one or more objects that found on detection process using a camera.

1.1 Motivation:

The area of automated surveillance systems is currently of immense interest due to its implications in the field of security. Surveillance of vehicular traffic and human activities offers a context for the extraction of significant information such as scene motion and traffic statistics, object classification, human identification, anomaly detection, as well as the analysis of interactions between vehicles, between humans, or between vehicles and humans. A wide range of research possibilities are open in relation to visual surveillance and tracking.

Some useful applications of tracking include recognition of interactions between humans, identification of people using gait analysis, and recognition of specific human activities - for instance, the type of tennis stroke being played. Some applications involve both humans and vehicles at the same time, such as simultaneous tracking of humans and vehicles. The efficacy of object tracking in non-surveillance applications is demonstrated by applications such as animal detection and identification in videos. A key step in all of the above is the ability to track a particular object or objects in successive frames. In our research, the primary task is to track vehicles in static camera video as a precursor to future research in intelligent analysis of traffic patterns and detection of anomalous behaviour by vehicles on the road.

1.2 System description

The process of automatic tracking of objects begins with the identification of moving objects. We use an improved background subtraction method in conjunction with a novel yet simple background model to achieve very good segmentation. Once the moving pixels are identified, it is necessary to cluster these pixels into regions, which we refer to as blobs, so that pixels belonging to a single object are grouped together. Single moving objects are often incorrectly separated into two or more subregions because of lack of connectivity between pixels, which usually occurs due to occlusion from other objects (e.g., trees). A blob merging module to merge close-by blobs is implemented. Having established individual moving objects, the next task in tracking is to achieve correspondence between the blobs in one frame and those in the next. Since we work with real-life data, the algorithm is designed to be robust to real-life tracking issues like occlusion, appearance and disappearance of objects, and abrupt change in speed, location, and orientation of objects. The robust tracking system has been satisfactorily tested on various static camera scenes involving both humans and vehicles. A block diagram of the system is given in figure 1.1. An image sequence captured by a still camera is the input to the system. We first perform Motion Segmentation to extract moving blobs in the current frame.

1.3 Objectives:

Automatic tracking of objects can be the foundation for many interesting applications. An accurate and efficient tracking capability at the heart of such a system is essential for building higher level vision-based intelligence. Tracking is not a trivial task given the non-deterministic nature of the subjects, their motion, and the image capture process itself. The goal of the work documented in this thesis is two-fold: (a) to set up a system for automatic segmentation and tracking of moving objects in stationary camera video scenes at the University of Kansas, which may serve as a foundation for higher level reasoning tasks and applications, and (b) to make significant improvements in commonly used algorithms. To achieve the first goal of setting up the tracking system, suitable data structures and algorithms for object segmentation and tracking in image sequences have been designed in the existing KUIM image processing code. The second goal is to make the tracking system more reliable and to contribute to existing research in field of object tracking. This is achieved by (1) Use of velocity flow as an important factor in tracking (2) Automatic learning of spatial thresholds and depth information of objects in different parts of the scene and (3) Design of a new

Bayesian algorithm for probabilistic tracking of objects. These additions to the basic system result in improved tracking as documented in the later sections of this thesis. The entire system has been tested on various videos and the segmentation and tracking results are very encouraging. Though our current focus is on accurate automatic tracking of vehicles, the system is also capable of tracking humans. The major contribution of this thesis is establishing a baseline system for automatic tracking in image sequences by including some novel techniques unique to this thesis work.

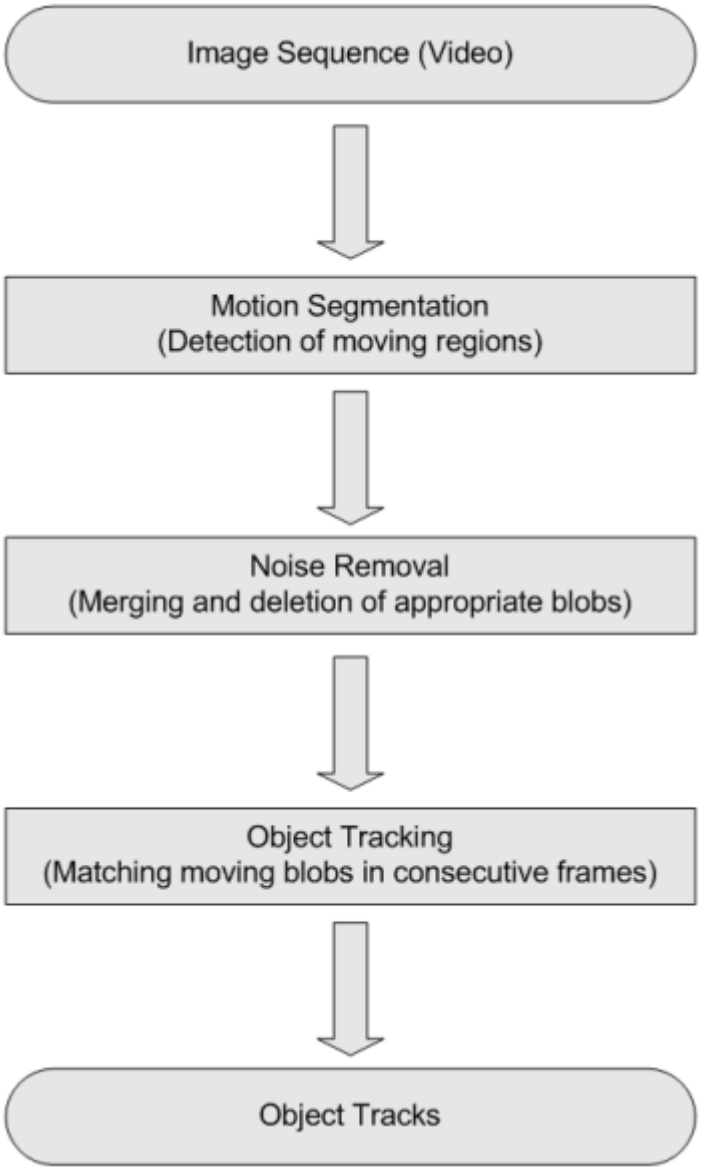


Figure 1.1. Block diagram of the system

very small and are likely to be noise are deleted. Due to partial occlusion, some moving objects get incorrectly segmented into two or more separate blobs. Using color and position

information, such blobs are merged. The Object Tracking module tracks these moving objects over successive frames to generate Object Tracks. Significant improvements are brought into the basic tracking system by using the velocity of moving objects as an important factor in the tracking algorithm. This is documented in section 4.2.

1.4 Problem formulation

The problem was about the ability to recognize different objects in a video, where the objects can be rigid (e.g. container) as well as non-rigid (e.g. clothing). Object tracking could be used to automate the process of presenting information about a certain object displayed in a video instead of manually having to search for the object in, for instance, a product catalog. The project aimed to, as far as possible, answer the questions below.

• What methods for object tracking/image recognition in video are there?

• What separates the methods, and what are the benefits and disadvantages with these?

• Are different methods more suitable for certain applications or environments?

1.5 Delimitation

The aim of this project was to make an implementation of different techniques which could be used to quickly and easily decide which methods that are suitable, or not suitable, for the chosen area of application. With this in mind it was concluded that the comparative work should focus on object tracking groups, or categories, and not on specific algorithms. Since there exists many different approaches, methods, and variants of methods for object tracking it would be an impossible task to thoroughly cover all methods used in some object tracking implementation. Especially since there are proprietary methods whose implementations and designs are not available for this project. This project focused on the, in literature, most common categories or groups of object tracking for rigid as well as non-rigid objects. Aside from object tracking techniques this project does also include other important parts of the tracking process. Since the aim of this project is to review methods for object tracking, and there are time requirements to meet.

CHAPTER 2

LITERATURE SURVEY

In computer vision field, object detection and tracking plays a vital role. Object detection means locating/identifying objects in frame of video sequence and whereas tracking is the process of locating moving object or multiple objects over a period of time using camera. Technically, tracking is estimating trajectory or path of an object in the image. The availability of high power computers, high quality and low cost camera will lead the great deal of interest in object tracking algorithms. Three main key steps for video analysis are : Detection of Interesting moving Objects, Tracking of such objects from frame to frame, Analysis of Object tracks to recognize their behaviour.

The main application areas of object detection and tracking are: Motion based recognition, automated surveillance, video indexing, human-computer interaction, traffic monitoring, vehicle navigation and etc. But object detection and tracking is multifaceted course of action when projecting 3D world on 2D image because of the loss of information, noise present in an image, complex object motion, articulated nature of object, complex object shapes and occlusion. Almost all tracking algorithms assume that the object motion is smooth with no abrupt changes. But practically it is impossible. Even though so many difficulties are exist in object detection and tracking, a number of object detection and tracking algorithms are proposed and implemented. Each and every algorithms are varied with respect to the object representation used, image features and the motion, appearance and shape of the object be modelled for detection and tracking. The shape representation should be combined with appearance representation. There are a various ways to represent the appearance feature of objects. Some of the common appearance representation in the case of object tracking is described which are as follows:

2.1 Object Representation

An entity of an interest is an object. Object can be represented by their shapes and features. There are various object representations available for tracking. The selection of object representation for tracking is based on application. Some of the object representation techniques are: Points representation which is suitable for tracking an object which occupies small region in an image, Primitive geometric shape representation is suitable for tracking both

rigid and non-rigid objects, Object silhouette representation is suitable for tracking complex non rigid shapes, articulated shape model is suitable for tracking articulated object with torso, legs, hands and etc, skeletal model is suitable for tracking both articulated and rigid objects. In order to track the objects, shape representation can be combined with the appearance representation. There are a various ways to represents the appearance feature of objects. Some of the common appearance feature representations are described in is as follows: Probability densities of object appearance which estimates the object's parameters, Templates which carries both spatial and appearance information of objects, Active appearance model which defines the object shape by set of landmarks (i.e., colour, texture or gradient magnitude) and multi view appearance model which encodes different views of an object.

2.2 Object Detection

The object detection is the process of locating objects in the frame of video sequence. Every tracking algorithm requires an object detection mechanism either in every frame or when an object occurs newly in a frame. Most of the object detection mechanism used information from single frame for detecting an object. But some of the object detection mechanism used temporal information which is computed from sequence of frames. It will reduce the false detection rate. There are several object detections mechanisms, which are shown in figure 1. The first step of real time object detection and tracking is to identify the region of interest of the video. Some of the object detection methods are:

1. Point Detector,
2. Background Modelling,
3. Segmentation,
4. Optical Flow and
5. Supervised Classifier.

1. Point Detector

Which are used to find the interesting points in an image. In the literature, commonly used point detectors are Moravec's detector, Harris detector, KLT detector, SIFT detector.

2. Background Subtraction

Which finds the digression from background model which are already built and incoming frames. Some of the background subtraction mechanisms are: Frame differencing Region-based or spatial information, Hidden Markov Model, Gaussian Mixture based Background Model,, Dynamic Texture based background model, Wall flower based background model and Eigen Space decomposition. As illustrated in and , background modelling has two main approaches. One is Recursive algorithm and another one is Non-recursive algorithm. Recursive algorithm do not maintain buffer for background estimation. It recursively updates a background model based on the input frames. So it requires less storage. This technique includes the following methods: approximate median, adaptive background, Gaussian mixture. A non-recursive algorithm uses sliding window approach for estimating background model.

3. Segmentation

Which partitions the images into perceptually similar regions. In, they have been discussed various segmentation techniques that are related to object tracking. They are: Mean shift Clustering, Image Segmentation using Graph-cuts, and Active contours.

4. Optical Flow

Method calculates the image optical flow field and perform clustering according to the optical flow distribution characteristics of images.

5. Supervised Classifier method

An operator is trained to detect the feature of the objects. Some of the supervised classifier methods are: SVM, Neural Networks based detector and adaptive boosting techniques.

2.3 Object Tracking

An object tracker aims to generate trajectory of an object over time by locating its position in every frames of video. Generally, object tracking has two main tasks. One is to detect the object and another one is to establish the correspondence between the object in every frames of video. These tasks can be performed either separately or jointly. The first case, the object in region has been detected by detector mechanism and later connection between objects in sequence of frame has been done by tracker. In the latter case, the object region and correspondence is jointly estimated. It can be done by iteratively updating object location and region information obtained from previous frames .

Here are some of the basic object detection and tracking algorithms:

2.3.1 BACKGROUND MODEL BASED DETECTION AND TRACKING

Qiang Ling et.al, developed feedback based object detection algorithm. It adopts dual layer updating model to update the background and segment the foreground with an adaptive threshold method and object tracking is treated as a object matching algorithm

2.3.2 POINT DETECTION AND TRACKING

A salient feature point based algorithm for multiple object tracking in the presence of partial object occlusion has been proposed. In this method, the extract the prominent feature points from each target object and then use a particle filter based approach to track the feature points in image sequences based on various attributes such as location, velocity and other descriptors. They used rectangular bounding box for object representation. But this algorithm may not successfully track feature points with different velocities. Hence this algorithm needs more flexible object representation and also they used static camera for capturing the video.

2.3.3 POINT DETECTION AND RANDOMIZED FOREST CLASSIFIER FOR TRACKING

The long term visual people detection in complex environment is too difficult and also tracking a varying number of objects entail the problem of associating detected objects to tracked targets. It will lead to data association problem. Tobias Klinger., et.al., proposed a tracking by detection strategy that uses randomized forests as a classifier with kalman filter. Randomized forest builds the strong classifier for multi class problem through aggregating simple decision tree .

2.3.4 SEGMENTATION BASED DETECTION & SILHOUETTE TRACKING

A unified framework for both single and cross camera tracking with affinity constraints using graph matching was proposed in . In this method, they mainly dealt with the problem of existence occlusion in single camera scenario & the occurrence of transition in cross camera scenario and also they consider the data association method in handling occlusion. They consider the tracklet association problem as graph matching with affinity constraints and leverage both person wise and part wise attribute for similarity measurement between tracklets to overcome the uncertainty and noise. The crucial problem caused by cross camera tracking lies in the drastically increasing the data.

2.3.5 POINT DETECTION AND KERNEL TRACKING

Video Surveillance cameras used in metro-stations are always short shot, low quality and nondirective downward view. So, accurately count the pedestrians in the case of not changing the position of the surveillance camera is the most challenging problem. Hence they proposed, novel method to count pedestrians in metro stations based on HAAR like detection and template matching.

2.3.6 BACKGROUND SUBTRACTION BASED MOTION DETECTION AND AGENT BASED TRACKING

In order to overcome the lack of conventional video surveillance system that is based on human perception, they proposed Cognitive Video Surveillance system (CVS) . It is based on Agent based object tracking. CVS offers some important people tracking attributes such as suspect object detection and smart camera cooperation. This Agent based tracking approach is suitable for distributed system. Background subtraction technique is used for detection of people motion. They have used C# for presenting image in binary form and used Waikato Environment for Knowledge Analysis to implement machine learning and data mining algorithm. And also, they used Exponential Moving Average (EMA) especially for human behavior prediction.

2.3.7 KLT POINT DETECTION AND OPTICAL FLOW TRACKING

A Scene feature based algorithm was proposed in for detecting counter flow in security related surveillance in airport. They addressed the two main problems:

1. Most of the cameras deployed in security surveillance networks have poor resolution. It will create negative effects on tracking algorithm.

2. 24/7 basis operation of automatic video analytics algorithm some time will provide higher false positive rate in tracking. To avoid such problems they used novel classifier to identify scene feature in the image and KLT optical flow tracking algorithm.

2.3.8 HYBRID APPROACH: BACKGROUND BASED AND APPEARENCE MODEL BASED DETECTION AND TRACKING.

Due to the variability of background and appearance model, monocular multi object detection with static camera is challenging one. Most of the current methods focus carefully on camera placement to avoid occlusion . Severin Stalder,et. Al., proposed Cascaded Confidence Filter for Improved Tracking by detection. They combined Background and appearance model.

Their approach significantly improve especially in case of partial occlusion, changing background, and similar distracter.

CHAPTER 3

PROBLEMS

Using camera, tracking multiple objects faces many problems. The problems are categories in figure 2.

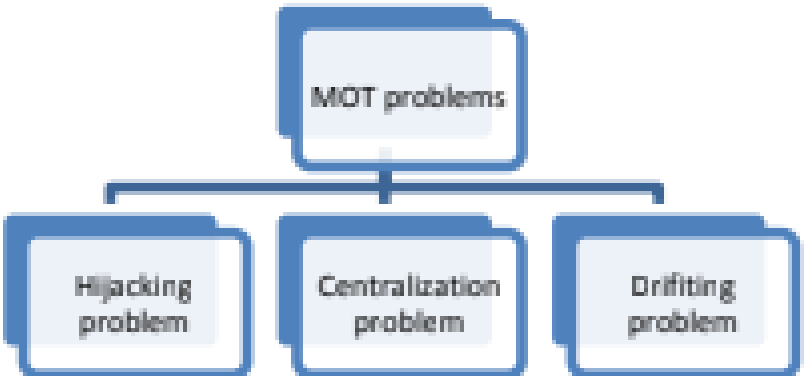


Fig 3.1: Multiple object tracking problems

Before going to details the phenomena of track lets and trajectory needs to discuss. Track let is segmented object traveling path that represent consecutive states of an object. By joining all track lets, objects trajectory is found. Figure 3 shows the track let and trajectory representation of an object.

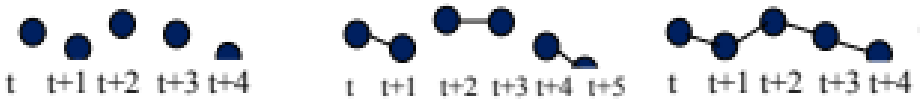


Fig 3.2: States of object in consecutive frame from t to t4 (left), track lets of an object (middle), trajectory of an object.

3.1 Hijacking problem

If two similar colour objects come too close, the tracker of one object can jump to associates object. This problem is called hijacking problem. Figure 4(a) shows the hijacking problem. Here, two similar colour objects come closer and one of the tracker jumps to other object. For this reason, the tracker misses the correct object to track.

3.2 Centralization Problem

Centralization problem occurred when two objects centralized into same position and the tracker misses to track the object each object individually when those object split from each other. Figure 4(b) shows the centralization problem. Here, two objects overlapped each other, when those object detached; the tracker can detect the wrong object.

3.3 Drifting problem

Drifting problem occurred when an object abruptly changes its direction to reverse. In that case, it's become very difficult to track the object; because the motion model doesn't work. Figure 4(c) shows the object drifting problem. Here, the dotted rectangle represents predicted object location in next frame that is predicted by object motion model.

(a) Hijacking problem

(b) Centralization problem

(c) Drifting Problem

Fig3.3. Multiple object tracking problems

3.4 Proposed System modelling

Existing methods are either active or passive. Active methods are reliable but use special hardware, often expensive and intrusive, e.g. infrared cameras and illuminators, wearable devices, glasses with a special close-up camera observing the eyes While the passive systems rely on a standard remote camera only. Many methods have been proposed to automatically detect objects in a video sequence. Several methods are based on a motion estimation in the frame. Typically, the objects are detected by a detector. Next, motion is estimated by 8 different object trackers and dlib library.

3.5 Technological Feasibility

The ability of two state-of-the-art landmark detectors to reliably distinguish between the objects is quantitatively demonstrated on a challenging in-the wild dataset.

A novel real-time object detection algorithm which integrates a landmark detector and a classifier are proposed. The evaluation is done on two standard datasets achieving state of the-art results.

3.6 Operational Feasibility

The object moment is fast. Each object has a little bit different pattern. The pattern differs in the motion. The moment can be detected for approximately every 100-400 ms.

3.7 Economical Feasibility

Python comes with more than 1,000 data packages as well as the Conda package and virtual environment manager, called Anaconda Navigator, so it eliminates the need to learn to install each library independently. Hence, open source software's such as Anaconda Python is used to minimize the cost for the Developer.

CHAPTER 4

DESIGN

This chapter discusses each of the steps involved in our solution to automatic moving target detection and tracking in detail. Fig. 3.1 shows the block diagram of the system and its constituent modules again. An image sequence captured by a still camera is the input to the system. We first perform Motion Segmentation to extract moving blobs in the current frame. Some blobs that are very small and are likely to be noise are deleted. Due to partial occlusion, some moving objects get incorrectly segmented into two or more separate blobs. Using color and position information, such blobs are merged. The Object Tracking module tracks these moving objects over successive frames to generate Object Tracks.

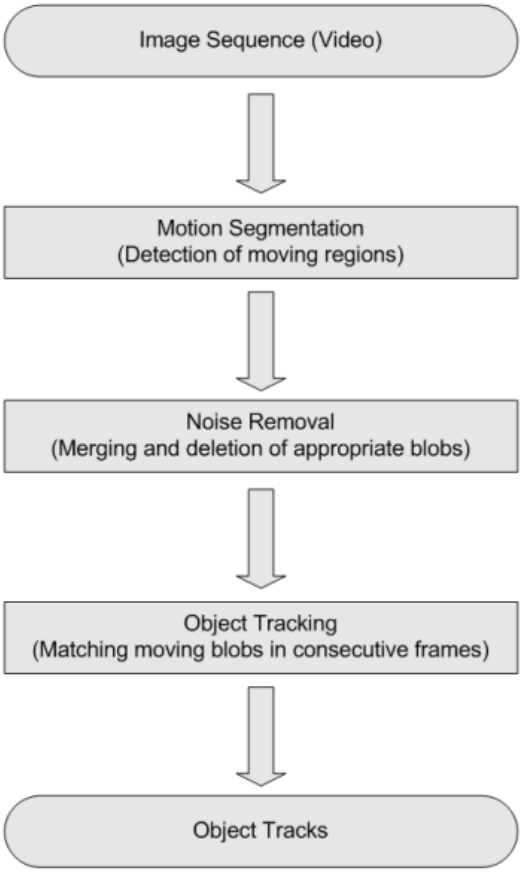


Figure 4.1. Block diagram of the system

4.1 Motion Segmentation:

The first task in any surveillance application is to distinguish the target objects in the video frame. Most pixels in the frame belong to the background and static regions, and suitable algorithms are needed to detect individual targets in the scene. Since motion is the key indicator of target presence in surveillance videos, motion-based segmentation schemes are widely used. We utilize an effective yet simple method for approximating the background that enables the detection of individual moving objects in video frames.

4.1.1 Segmentation using moving average background model

A standard approach for finding moving objects in still-camera video data is to create a model for the background and compare that model with the frame in question to decide which pixels are likely to be the foreground (moving objects) and which the background. There are various methods for developing background models. Our approach is to use a simple average model where the average intensity values for each pixel over a window of N (typically 150) frames is regarded as the background model. The entire background modelling and segmentation process is carried out on grayscale versions of the images. Fast-moving objects do not contribute much to the average intensity value ; thus, the model becomes a reasonable estimate of the background. If I k (y, x) is the intensity level at coordinates Y = y and X = x of the k^{th} frame in the sequence and bg^k (y, x) is the average background model value at (y, x), then

$$bg^k(y,x) = \frac{1}{N} \sum_{j=k-(N/2)}^{k+(N/2)} I^j(y,x)$$

The segmented output is:

$$seg1^k(y,x) = \begin{cases} 1 & \text{, if } |bg^k(y,x) - I^k(y,x)| > T1 \\ 0 & \text{, otherwise} \end{cases} \quad (3.2)$$

where T1 is a predetermined threshold, usually 10.

Fig. 4.2 shows the grayscale version of a frame that is being processed. Fig. 4.3 is an example of how the moving average background model appears in frame 1010 of the video sequence. After subtracting this background model from the current frame and thresholding the difference image, Fig. 4.4 results. The white pixels in the image are the regions that have been segmented as moving pixels in the current frame.

Figure 4.2 Grayscale version of current frame being processed for background segmentation.

Figure 4.3. Moving average background model for current frame 1010

Figure 4.4. Moving pixels segmented after subtraction from average background model and thresholding the difference

This base method does have a major drawback in that the segmented object regions have "tails" due to the effect of the window that is used for the average background model, as illustrated in Fig. 4.5(a), where an envelop has been established around the moving regions in the original colour image. The difference image shows a high value in the region where the object earlier/later existed and where the object now exists, resulting in tail regions behind and before the object. The correct segmentation for the frame, achieved by improvements in the basic algorithm as discussed in section 4.1.2 is shown in Fig. 4.5(b).

Figure 4.5. (a) Tails due to basic segmentation (b) Corrected results after use of Secondary background(section 4.1.2)

4.1.2 Enhanced segmentation using a Secondary Background model

It was observed that the intensity level of a background pixel changes very little from frame to frame. The value of the background in the previous frame is actually a better estimate of the background in the current frame than is the average value over N frames. To take advantage of this, we use a secondary model of the background (called Secondary Background) which is the actual intensity level of background pixels in the previous frame (or in the frame where the pixel was last classified as background). By combining the two models, a near perfect segmentation of the moving objects is obtained. The average model is used to identify moving regions and the Secondary Background is used to identify the valid moving object within these regions. The concept is illustrated in Fig. 4.6. The oval region around the car is the moving region. In frame 3, we can differentiate between the car region (D) and background regions (C and E) by using the Secondary Background for regions D (from frame 2), C (from frame 1), and E (from frame 2).

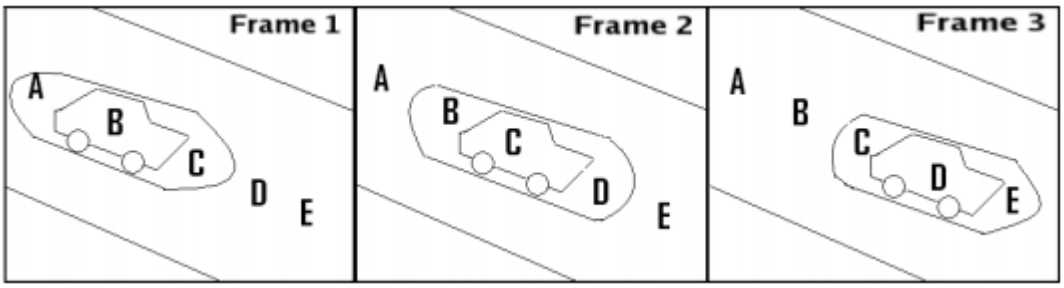


Figure 4.6. Illustration of the Secondary Background concept

After the segmentation, the background model is updated by calculating the average over N frames. The Secondary Background is updated by storing the intensity levels of pixels that were classified as background as the new Secondary Background. For pixels that were classified as foreground, the Secondary Background is left unaltered so that it holds the value of intensity from the frame where the pixel was last classified as background. Secondary Background for frame k is denoted by sbg^k . For initialization, we simply set sbg to the background calculated for the first frame:

$$sbg^1 = bg^1(y, x)$$

where sbg^1 is the initial value for Secondary Background. The segmented output based on the Secondary Background is denoted by:

$$seg2^k(y,x) = \begin{cases} 1 & \text{, if } (|sbg^k(y,x) - I^k(y,x)| > T2 \text{ and } seg1^k(y,x) = 1) \\ 0 & \text{, otherwise} \end{cases} \quad (3.4)$$

T2 is a threshold, usually greater than T1. In our application, T2 was set at 15. We then update the Secondary Background for the next frame:

$$sbg^{k+1}(y,x) = \begin{cases} I^k & \text{, if } seg2^k(y,x) = 0 \\ sbg^k & \text{, otherwise} \end{cases} \quad (3.5)$$

As you can see, the segmentation and background modelling processes are interdependent, with the segmentation result feeding back to the Secondary Background model. The Secondary Background model for the example frame 1010 are shown in Fig. 4.7. Subtraction from current frame and thresholding, as described in equation 4.4, results in the segmentation of the moving pixels as Fig.4.8. This improved segmentation algorithm gives the correct segmentation that was presented in Fig. 4.5(b). On comparing figures 4.3 and 4.7, we can see that the section of road in front of where the cars are found in the current frame is slightly darker in the basic method (Fig. 4.3). The improved background (Fig. 4.7) that uses the Secondary Background model does not exhibit this error. Our background modelling method is much simpler than other adaptive background methods such as Stauffer and Grimson. We found that while these complicated methods promise good results for complex and bimodal backgrounds

Figure 4.7 Secondary Background model for current frame 1010

Figure 4.8. Moving pixels segmented after subtraction from Secondary Background model and thresholding the difference

They have very complicated procedures and are computationally expensive. In essence, our method is similar to Stauffer and Grimson's Gaussian background pixel model, but with a very high learning rate for the model. Our simple implementation was able to produce good results with little computational complexity. A sample of the segmentation results obtained using the Secondary Background model is shown in Fig. 4.5(b).

4.2 Noise removal

Following a segmentation of the image frame into moving and non-moving pixels, region growing is used to locate and identify each moving object. Each group of pixels that are connected to each other is labelled as one blob. An envelope is established around each object of interest by use of morphological erosion and dilation filters. The erosion-dilation operation also removes blobs that are very small and thus likely to be noise. Additional noise is filtered by ignoring blobs less than S pixels in size. The envelop of the grayscale image is copied in the same locations on the colour version of the current image. The colour version of the frame is then analysed to get the statistics for each blob. By averaging the location and colour values for all pixels belonging to a blob, the centroid value of the blob(in Y and X coordinates), and mean R,G, and B values are obtained. These values are used as statistics for matching blobs across frames.

4.3 Representation of a blob

The variables and their description in the data structure used for representing an individual blob are given in Fig. 4.9. Blob ID is the number that is given to a blob. The track number of the blob, as assigned by the tracking algorithm, is stored in this variable. The mean R, mean G, and mean B variables store the average intensity values of red, green, and blue colours of all pixels belonging to the blob. The centroid position of all pixel locations is stored in yCenter and xCenter. The size variable keeps record of the size of the blob in pixels (number of pixels belonging to the blob).

The Blob class used for keeping record of all the individual blobs in the video is given in Fig. 4.10. An array of individual blobs is maintained in the blobs variable. numBlobs is an array that stores the number of blobs in each frame of the video. The getBlobStats function analyses the current frame to generate all the statistics for all the blobs in the frame. Mean R, mean G, mean B, yCenter, xCenter, and size are the statistics that are calculated and stored for each blob in the frame. Connect BlobPixels does a connected pixel based region growing to group together individual pixels into regions (blobs). Clean Blobs function removes any blobs that are smaller than a given threshold size. Thus, very small blobs that may be noise are deleted. mergeBlobs is a function that merges blobs that are very close to each other in position and colour values. The need for mergeBlobs function and some issues involved in its implementation are discussed in chapter 4. drawEnvelope and labelBlobs functions are used to mark out the blobs in the original image. drawEnvelope function uses morphological erosion and dilation operations to draw an envelop around the detected blob in the image. labelBlobs is a function that is designed to write the blobID number near the centroid of all detected blobs in the image.

INDIVIDUAL BLOB STRUCTURE

Variable	Description
BlobID	ID number of blob - usually the number of the track to which the blob belongs
meanR	Average Red value of all pixels in blob
meanG	Average Green value of all pixels in blob
meanB	Average Blue value of all pixels in blob
yCenter	Position of centroid of blob along Y axis (column)
xCenter	Position of centroid of blob along X axis (row)
size	Size of blob in pixels

Figure 4.9. Variables in the individual blob data structure

BLOB CLASS

	Description
Variables:	
blobs	Array of all individual *blob*s in the video
numBlobs	Array that stores the number of *blob*s in each frame of the video
Methods:	
getBlobStats	Analyzes current frame to generate *meanR*, *meanG*, *meanB*, *yCenter*, *xCenter*, and *size* of each *blob* in current frame. Updates *blobs* and *numBlobs* variables.
connectBlobPixels	Does connected pixel analysis to create blobs from pixels
cleanBlobs	Deletes *blob*s smaller in size than a pre-decided threshold
mergeBlobs	Merges blobs that are very similar in *Ycenter*, *Xcenter*, *meanR*, *meanG*, and *meanB* values
drawEnvelope	Draws an envelop around each blob in the image by using erosion and dilation morphological operations
labelBlobs	Writes the *blobID* number close to the centroid of the blob in the image

Figure 4.10. Variables and methods in the Blob class

4.4 Representation of a track

The data structure used to represent an individual track is shown in Fig. 4.11. trackID stores the unique track number for the track. startingFrame and endingFrame variables store the first and last frame in which the track was seen, respectively. lostCount variable keeps count of how many consecutive frames for which a track has been 'lost'. This is kept so that a track that is lost for more than 5 frames can be deleted and closed. y and x keep record of the current position of the track in Y and X directions, respectively. velocityY and velocityX is the velocity of the track calculated over the last 5 frames in Y and X directions, respectively. predictedY and predictedX are the predicted positions for the track in the next frame, calculated based on the current velocity and position of the track. The Track class variables and functions are presented in Fig. 4.12. tracks is an array of all individual tracks in the video. numTracks is an array that stores the number of tracks in each frame of the video. openTracks is an array that keeps a list of all tracks that were active in the previous frame (including tracks that were 'lost', but not deleted). trackCount is the count of total number of tracks in the video. The functions createTrack and deleteTrack create a new track and delete a track, respectively. lostTrack function is called when a track is not found in the current frame. The function increments the

lostCount variable for the track. If the lostCount variable is greater than 5 (the track has been lost in more than 5 consecutive frames), then the lostTrack function calls the deleteTrack function and deletes the track. updateTrack function updates the position, velocity, and prediction variables of the track. EuclideanTracker is the tracking algorithm that finds correspondence between tracks of previous frame and blobs of current frame.

INDIVIDUAL TRACK STRUCTURE-TRACK

Variable	Description
trackID	ID number of track
startingFrame	The frame number when the track first appeared
endingFrame	The frame number when the track last appeared
lostCount	The number of frames for which the track has been "lost", this is 0 if the track was found in the previous frame
y	Location of track along Y direction
x	Location of track along X direction
velocityY	Velocity of track along Y direction
velocityX	Velocity of track along X direction
predictedY	The track's predicted position along Y direction in the next frame
predictedX	The track's predicted position along X direction in the next frame

Figure 4.11. Variables in the individual track data structure

	Description
Variables:	
tracks	Array of all individual *track*s in the video
numTracks	Array that stores the number of *track*s in each frame of video
openTracks	Array that stores a list of active tracks from the previous frame
trackCount	Total number of tracks in the video
Methods:	
createTrack	Creates new track
deleteTrack	Deletes a track
lostTrack	Declares a track as lost, increments *lostCount* for lost track If *lostCount* is greater than 5, calls deleteTrack function
updateTrack	Updates position, velocity, and prediction values for a successfully matched track
EuclideanTracker	Algorithm to find track-to-blob correspondence in each frame based on Euclidean distance

Figure 4.12. Variables and methods in the Track class based on the Euclidean distance between blobs and tracks in order to achieve continuous tracking over consecutive frames.

4.5 Object Tracking

In our implementation, we perform object tracking by finding correspondences between tracks in the previous frame and blobs in the current frame.

4.5.1 Correspondence algorithm

After the moving objects have been segmented as blobs, the next task is to find correspondence between tracks of previous frame and blobs of current frame. Most commonly, a match matrix is calculated for the tracks and the blobs, and the best matches are calculated from the match matrix. Each row in the match matrix corresponds to a track from the previous frame and each column corresponds to a blob from the current frame. Each element in the matrix is, thus, the degree of match between a track and a blob. The values entered in the matrix are usually the distance (Euclidean or Manhattan) between a track and a blob. For instance, in the match matrix consists of the weighted sum of Euclidean distance in position in Y and X coordinates, Manhattan distance in color histogram values, and distance in velocity values between a track and a blob. The smaller the distance, the better the match. In our system, each element in the match matrix is the sum of the Euclidean distance in position values (Y and X coordinate values) and the Euclidean distance in color values (R, G, and B values). The values for Y and X values are normalized against the maximum Y and X dimensions of the images, respectively. Similarly, R, G, and B values are normalized against their maximum possible value, which is 255.

Say $T^{k-1} = \{t_1^{k-1}, t_2^{k-1}, t_3^{k-1}, \ldots, t_{u^{k-1}}^{k-1}\}$ is the set of tracks from the previous frame (k − 1), where t_j^{k-1} is the j^{th} track and u^{k-1} is the total number of tracks in frame (k − 1), and $O^k = \{o_1^k, o_2^k, o_3^k, \ldots, o_{v^k}^k\}$ is the set of moving blobs in frame k, where o_i^k is the i^{th} blob in frame k and v^k is the total number of moving blobs in frame k. The j^{th} row and i^{th} column in the match matrix, denoted by:

$$MM_{j,i} = \sqrt{(\Delta Y/Ydim)^2 + (\Delta X/Xdim)^2} + \sqrt{(\Delta R/255)^2 + (\Delta G/255)^2 + (\Delta B/255)^2} \quad (3.6)$$

where

ΔY is the difference in Y position values,

ΔX is the difference in X position values,

ΔR is the difference in mean red values,

ΔG is the difference in mean green values, and

ΔB is the difference in mean blue values between track j of the previous frame and blob i of the current frame.

Y *dim* and X *dim* are the image dimension values in Y and X coordinates, respectively (240 and 320, in our images).

The Y and X position values of a blob correspond to the location of the centroid of all pixels that have been segmented as belonging to the concerned blob. The mean red, green, and blue values for each blob are calculated by averaging the R, G, and B values for all pixels that have been segmented as belonging to the concerned blob.

4.5.2 Match matrix

Fig. 4.13 shows a sample match matrix for frame 0300 of video sequence 1. For simplicity, only tracks that are present in frame 0295 have been displayed.

(a) Previous frame - 0295 (b) Current frame - 0300

	Blob 1	Blob 2	Blob 3	Blob 4
Track 3	1.18	0.02	0.13	1.10
Track 13	1.21	0.12	0.03	1.10
Track 16	0.36	1.03	1.00	0.21
Track 19	0.11	1.28	1.28	0.33

(c) Match matrix for current frame 0300

Figure 4.13. Match matrix for example frame 0300

The matrix shows that the best matches for tracks 03, 13, 16, and 19 are blobs 2, 3, 4, and 1, respectively.

4.5.3 Factors of uncertainty in tracking

In the case of a noiseless environment and perfect segmentation of blobs, the task of tracking is straightforward. In reality, however, there is often a great deal of uncertainty in the final segmentation, as well as difficulties produced from imperfect knowledge of the scene context. Some common reasons for uncertainty are:

• Valid blobs may fail to be detected due to their small size.

• Valid blobs may be occluded by another moving or static object.

• Some moving objects such as tree branches, though valid moving blobs, are not of interest for our application.

• Some blobs might be incorrectly merged by the blob merging module.

• Objects entering (or leaving) the scene do not have corresponding blobs in preceding (or succeeding) frames .

• An object may change in orientation or be temporarily occluded, leading to a change in its visible surfaces and thus its color statistics or even its size.

All of the above, in addition to jitter and noise from the camera, make the correspondence a difficult problem to solve, particularly for videos that are shot outdoors. Our goal is to develop a tracking scheme that is robust to these sources of uncertainty.

4.5.4 Track assignment procedure

For each motion blob in the first frame, a new track is created. In subsequent frames, based on the match matrix values, matching blobs are sought in the current frame for each track in the previous frame. Matched blobs are assigned track numbers from previous frame. The track statistics are updated with position and color information from the matching blob. We allow tracks to be declared ’lost’ if any of the factors mentioned above causes the algorithm to fail to detect a suitable match for a track. The track is not deleted but is ’kept alive’ for a few frames in hopes that a match will be found. A track is deleted only if a suitable match is not found for L (typically 5) consecutive frames. If there is any blob in the new frame that cannot

be matched to an existing track, then a new track is generated. A list of currently 'active' tracks is maintained for each time step/frame. If there is a tie between multiple tracks for a single blob, then the older of the competing tracks is assigned to the blob. This is because a longer running track is less likely to be noise.

4.6 Results

Some results of the base system are shown below. In the figures where match matrices are presented, the assigned blob-track pairs are shaded differently in the matrix for easy reading. Fig. 4.14 shows the match matrix for a case when a new track has to be initialized to account for a new unmatched blob that occurs in the current frame. Current frame 0290 has two new blobs compared to the previous frame 0285. The match matrix, when solved as explained earlier, results in assigning blobs numbered 2, 3, and 4 to tracks 03, 13, and 16, respectively. Since blobs numbered 1 and 5 are not matched to any active previous tracks, new tracks 19 and 20 are started to account for these, respectively. Fig. 4.15 shows the match matrix when a previously active track gets lost. In this case, track 20, which was active in frame 0290 is now lost due to occlusion behind the tree in the foreground. The match matrix shows that two tracks (16 and 20) are good matches for the blob numbered 4. The algorithm is designed to choose the older of the two tracks. Since longer running track numbers are more reliable and less likely to be noise, we always choose the older numbered track in case of a tie between track numbers to be assigned to a blob. Hence, track 16 is assigned to the blob and track 20 is declared as 'lost'. The reason for choosing an older track over a newer one is clear in this example. As we see in the previous example (Fig. 3.14(b)), track 20 was created due to the erroneous breaking up of

(a) Previous frame - 0285 (b) Current frame - 0290

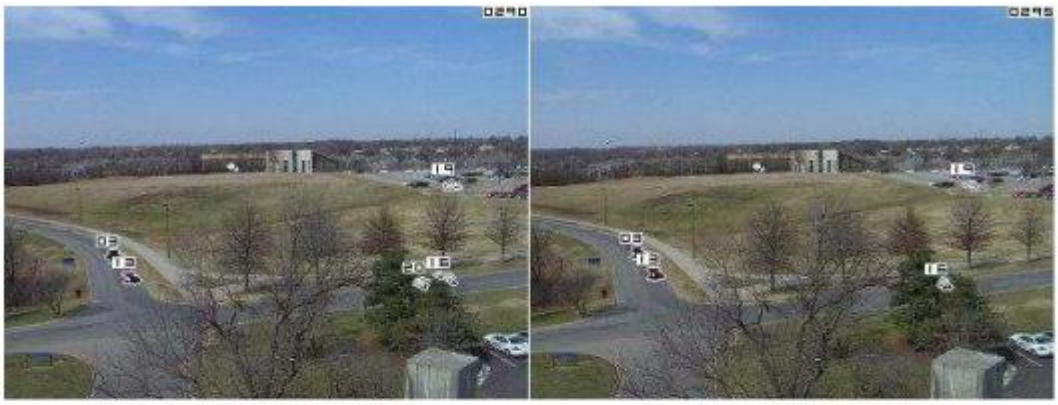
	Blob 1	Blob 2	Blob 3	Blob 4	Blob 5
Track 3	1.30	0.04	0.12	1.22	1.31
Track 13	1.28	0.12	0.03	1.18	1.26
Track 16	0.32	1.15	1.17	0.05	0.22

(c) Match matrix for current frame 0290

Figure 4.14. Match matrix when new track is initialized track 16 because of occlusion.

Choosing the older of the two tracks (16 and 20) ensures that the erroneous track is now deleted.

Fig. 4.16 shows a sequence of frames where the algorithm is able to allow for tracks to be ’lost’ when objects disappear due to occlusion and then recovered when they resurface. Tracks 03 and 08 are being tracked in these frames. Frames 0165 and 0170 (Figs. 4.16(a) and (b)) show both tracks. In frame 0175 (Fig. 4.16(c)), track 08 is ’lost’ due to the car being occluded by the tree. In frame 0180 (Fig. 4.16(d)), both tracks are ’lost’ due to occlusion. In frame 0185 (Fig. 4.16(e)), track 08 is reassigned to the correct blob when the object re-emerges from the occluded region. Finally, in frame 0190 (Fig. 4.16(f)), when both objects have resurfaced, the algorithm recovers the correct track numbers.



(a) Previous frame - 0290 (b) Current frame - 0295

	Blob 1	Blob 2	Blob 3	Blob 4
Track 3	1.28	**0.01**	0.13	1.05
Track 13	1.31	0.13	**0.01**	1.06
Track 16	0.31	1.16	1.16	**0.11**
Track 19	**0.04**	1.27	1.30	0.40
Track 20	0.32	1.29	1.29	0.23

(c) Match matrix for current frame 0295

Figure 4.15. Match matrix when track is lost

(a) Frame 0165 (b) Frame 0170

(c) Frame 0175 - track 08 lost (d) Frame 0180 - both tracks 03 and 08 lost

Figure 4.16. Tracks from subsequence in video 1

CHAPTER 5

IMPLEMENTATION

5.1 Machine Learning Algorithms

The machine learning algorithms that are included in OpenCV are given as follows. All the algorithms are present in the ML library apart from Mahalanobis and K-means, which are present in CVCORE, and the algorithm of face detection, which is present in CV.

5.1.1 Mahalanobis

It is a measure of distance that is responsible for the stretchiness of the data.We can divide out the covariance of the given data to find this out. In case of the covariance being the identity matrix (i.e. identical variance), this measure will be identical to the Euclidean distance.

5.1.2 K-means

It is an unsupervised clustering algorithm which signifies a distribution of data w.r.t. K centers, K being chosen by the coder. The difference between K-means and expectation maximization is that in K-means the centers aren't Gaussian. Also the clusters formed look somewhat like soap bubbles, as centers compete to occupy the closest data points. All these cluster areas are usually used as a form of sparse histogram bin for representing the data.

5.1.3 Normal/Naïve Bayes classifier

It is a generative classifier where features are often assumed to be of Gaussian distribution and also statistically independent from one another. This assumption is usually false. That's why it's usually known as a ―naïve Bayes classifier. That said, this method ‖ usually works surprisingly well.

5.1.4 Decision trees

It is a discriminative classifier. The tree simply finds a single data feature and determines a threshold value of the current node which best divides the data into different classes. The data is broken into parts and the procedure is recursively repeated through the left as well as the right branches of the decision tree. Even if it is not the top performer, it's usually the first thing we try as it is fast and has a very high functionality.

5.1.5 Boosting

It is a discriminative group of classifiers. In boosting, the final classification decision is made by taking into account the combined weighted classification decisions of the group of classifiers. We learn in training the group of classifiers one after the other. Each classifier present in the group is called a weak classifier. These weak classifiers are usually composed of single-variable decision trees known as stumps. Learning its classification decisions from the given data and also learning a weight for its vote based on its accuracy on the data are things the decision tree learns during training. While each classifier is trained one after the other, the data points are re-weighted to make more attention be paid to the data points in which errors were made. This continues until the net error over the entire data set, obtained from the combined weighted vote of all the decision trees present, falls below a certain threshold. This algorithm is usually effective when a very large quantity of training data is available.

5.1.6 Random trees

It is a discriminative forest of a lot of decision trees, each of which is built down to a maximal splitting depth. At the time of learning, every node of every tree is allowed a choice of splitting variables, but only from a randomly generated subset of all the data features. This ensures that all the trees become statistically independent and a decision maker. In the run mode, all the trees get an unweighted vote. Random trees are usually quite effective. They can also perform regression by taking the average of the output numbers from every tree.

5.1.7 Face detector (Haar classifier)

It is an object detection application. It is based on a smart use of boosting. A trained frontal face detector is available with the OpenCV distribution. This works remarkably well. We can train the algorithm for other objects by using the software provided. This works wonderfully for rigid objects with characteristic views.

5.1.8 Expectation maximization (EM)

It is used for clustering. It is a generative unsupervised algorithm it fits N multidimensional Gaussians to the data, N being chosen by the user. It can act as an efficient way for representing a more complex distribution using only a few parameters (i.e. means and variances). Usually used in segmentation, it can be compared with K-means.

5.2 OpenCV Object tracking algorithms

OpenCV provides eight different object tracking implementations that you can use in your own computer vision applications. They are as follows :

1 . BOOSTING tracker

2 . MIL tracker

3 . KCF tracker

4 . CSRT tracker

5 . MedianFlow tracker

6 . TLD tracker

7 . MOOSE tracker

8 . GOTURN tracker

Given below is the brief highlight of each tracker.

5.2.1 BOOSTING Tracker:

Based on the same algorithm used to power the machine learning behind Haar cascades (AdaBoost), but like Haar cascades, is over a decade old. This tracker is slow and doesn't work very well. Interesting only for legacy reasons and comparing other algorithms. (minimum OpenCV 3.0.0)

5.2.2.MIL Tracker:

Better accuracy than BOOSTING tracker but does a poor job of reporting failure. (minimum OpenCV 3.0.0)

5.2.3.KCF Tracker:

Kernelized Correlation Filters. Faster than BOOSTING and MIL. Similar to MIL and KCF, does not handle full occlusion well. (minimum OpenCV 3.1.0)

5.2.4.CSRT Tracker:

Discriminative Correlation Filter (with Channel and Spatial Reliability). Tends to be more accurate than KCF but slightly slower. (minimum OpenCV 3.4.2)

5.2.5 MedianFlow Tracker:

Does a nice job reporting failures; however, if there is too large of a jump in motion, such as fast moving objects, or objects that change quickly in their appearance, the model will fail. (minimum OpenCV 3.0.0)

5.2.6 TLD Tracker:

I'm not sure if there is a problem with the OpenCV implementation of the TLD tracker or the actual algorithm itself, but the TLD tracker was incredibly prone to false- positives. I do not recommend using this OpenCV object tracker. (minimum OpenCV 3.0.0)

5.2.7 MOSSE Tracker:

Very, very fast. Not as accurate as CSRT or KCF but a good choice if you need pure speed. (minimum OpenCV 3.4.1)

5.2.8 GOTURN Tracker:

The only deep learning-based object detector included in OpenCV. It requires additional model files to run. (minimum OpenCV 3.2.0)

According to research it is being proved to :

1.Use CSRT when you need higher object tracking accuracy and can tolerate slower FPS throughput.

2.Use KCF when you need faster FPS throughput but can handle slightly lower object tracking accuracy.

3.Use MOSSE when you need pure speed.

In addition to 8 OpenCV trackers we are using dlib to track the blobs(binary large objects) automatically whereas we use the 8 OpenCV trackers to level 1 implementation.

5.3 Caffe framework

Caffe stands for Convolutional architecture for fast feature embedding. There are several powerful libraries that can be used to design and teach neural networks, including convolutional neural networks such as Theano, Lasagne, Keras, MXNet, Torch, and

TensorFlow. Among them, Caffe is a library that can be used to research and develop real-world applications . Caffe is a completely open-source library that gives open access to deep architectures. The library written in C ++ language is also implemented with matlab and python languages. Developed by Y. Jia, BVLC center. Caffe offers unit tests for accuracy, experimental rigor and installation speed, depending on the best practices of software engineering. In addition, the code is also well suited for research use because of its modularity and clean separation of the network definition from the actual implementation Models and optimizations can be done through the setting file without coding. The transition between CPU and GPU is seamless. Due to the high processing speed, it can be useful to use it in the image classification problem, which requires processing with millions of images. Caffe open source software is preferred because of these advantages.

5.4 Datasets

Dataset In practice, a pre-trained model using the ImageNet dataset is used. For testing, we use MobileNet SSD dataset. Database contains images of categorized objects. The number of objects contained within the categories varies. In general, the number ranging from 40 to 800 in each category is around 50 in most categories. The size of the images is 300x200 pixels.

Some images of the dataset related to one object will be as follows:

figure 5.1: sample data set about car object

5.5 Multiple Object Detection Approach

5.5.1 Flowchart

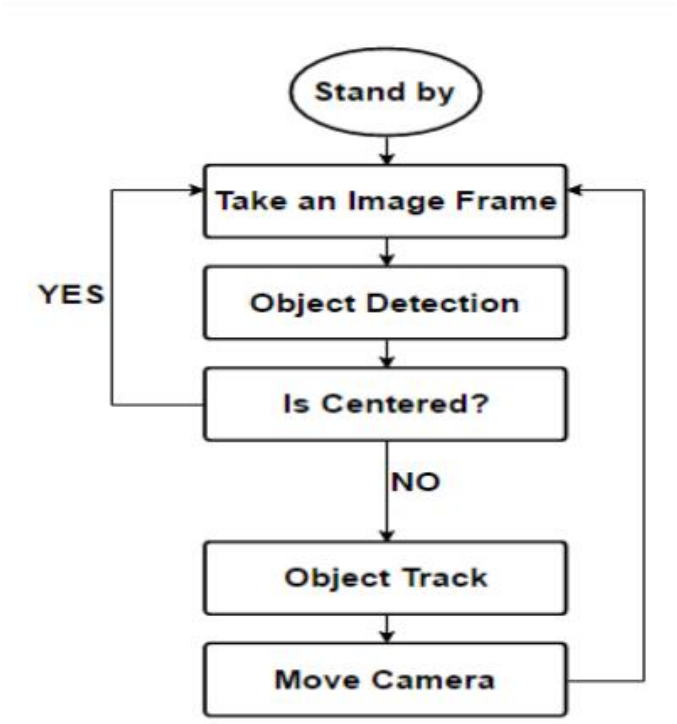


figure 5.2: Flow chart of multiple object tracking

Object tracking process can be simply defined in three steps with an order and these steps are controlled by the computer autonomously.

1. Detection of interested object in video stream.

2. Tracking target object in every single frame.

3. Analyzing the tracking process to recognize the behavior.

Human brain can easily understand the visual frames which seen by eyes, but computers cannot do it without any special methodology. If we compare human and computer, camera module is the eye of the computer and the computer vision is the way that how brain understand the image. Computer vision is a field that includes some methods, processing and analyzing techniques for understanding images. Also known as image analysis, scene analysis and image understanding.

5.6 LEVEL 1 Functioning:

In level 1, we provide the option for the user to manually select the object that he wants to track. In this implementation we use the hardware component i.e,. external web camera.The implementation can also be done by providing the video as input. The project also has a provision of storing the vidoes in the path specified.

5.6.1 Steps involved

Step 1: Image acquisition

Utilizing a web camera introduced inside the automobile we can get the picture of the driver. Despite the fact that the camera creates a video clip, we have to apply the developed algorithm on each edge of the video stream. This paper is only focused on the applying the proposed mechanism only on single frame. The used camera is a low cost web camera with a frame rate of 30 fps in VGA mode. Logitech Camera is used for this process is shown in figure below.

figure 5.3: External web camera module

Step 2: Dividing into Frames

We are dealing with real time situation where video is recorded and has to be processed. But the processing or application of algorithm can be done only on an image. Hence the captured video has to be divided into frames for analysing.

Step 3: Object detection

In this stage we detect the region containing the images that match the dataset. The object tracking algorithms can be used for detection of objects in every frame. By object detection we means that locating the object in a frame or in other words finding location of objects through a type of technology with the use of computer. The frame may be any random frame. Only objects havings features that are in selected zone detected and all others types of objects that are not selected are ignored.

Step 4: Source code

```
from imutils.video import VideoStream
import argparse
import imutils
import time
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
#ap.add_argument("-v", "--video", type=str,
# help="path to input video file")
ap.add_argument("-t", "--tracker", type=str, default="kcf",
help="OpenCV object tracker type")
ap.add_argument("-w", "--webcam", type=int, default=0,
help="index of webcam on system")
args = vars(ap.parse_args())

# initialize a dictionary that maps strings to their corresponding
# OpenCV object tracker implementations
OPENCV_OBJECT_TRACKERS = {
"csrt": cv2.TrackerCSRT_create,
"kcf": cv2.TrackerKCF_create,
"boosting": cv2.TrackerBoosting_create,
"mil": cv2.TrackerMIL_create,
"tld": cv2.TrackerTLD_create,
"medianflow": cv2.TrackerMedianFlow_create,
"mosse": cv2.TrackerMOSSE_create
```

```
}
# initialize OpenCV's special multi-object tracker
trackers = cv2.MultiTracker_create()
# if a video path was not supplied, grab the reference to the web cam
if not args.get("video", False):
print("[INFO] starting video stream...")
vs = VideoStream(src=args["webcam"]).start()
time.sleep(1.0)
# otherwise, grab a reference to the video file
else:
vs = cv2.VideoCapture(args["video"])
# loop over frames from the video stream
while True:
# grab the current frame, then handle if we are using a
# VideoStream or VideoCapture object
frame = vs.read()
frame = frame[1] if args.get("video", False) else frame
# check to see if we have reached the end of the stream
if frame is None:
break
# resize the frame (so we can process it faster)
frame = imutils.resize(frame, width=600)

# grab the updated bounding box coordinates (if any) for each
# object that is being tracked
(success, boxes) = trackers.update(frame)
# loop over the bounding boxes and draw then on the frame
for box in boxes:
(x, y, w, h) = [int(v) for v in box]
cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# show the output frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF
```

```
# if the 's' key is selected, we are going to "select" a bounding
# box to track
if key == ord("s"):
# select the bounding box of the object we want to track (make
# sure you press ENTER or SPACE after selecting the ROI)
box = cv2.selectROI("Frame", frame, fromCenter=False,
showCrosshair=True)

# create a new object tracker for the bounding box and add it
# to our multi-object tracker
tracker = OPENCV_OBJECT_TRACKERS[args["tracker"]]()
trackers.add(tracker, frame, box)

# if the `q` key was pressed, break from the loop
elif key == ord("q"):
break

# if we are using a webcam, release the pointer
if not args.get("video", False):
vs.stop()

# otherwise, release the file pointer
else:
vs.release()

# close all windows
cv2.destroyAllWindows()
```

Step 5: Result Analysis

After the application of the algorithm, the input is recorded along with the trackers and stored in the path provided as shown in chapter 6

5.7 LEVEL 2 functioning:

In level 2, the objects are automatically detected whenever the objects in the dataset comes into frame.

Steps involved

Step 1: Image acquisition

Utilizing a web camera introduced inside the automobile we can get the picture of the driver. Despite the fact that the camera creates a video clip, we have to apply the developed algorithm on each edge of the video stream. This paper is only focused on the applying the proposed mechanism only on single frame. The used camera is a low cost web camera with a frame rate of 30 fps in VGA mode.

Step 2: Dividing into Frames

We are dealing with real time situation where video is recorded and has to be processed. But the processing or application of algorithm can be done only on an image. Hence the captured video has to be divided into frames for analysing.

Step 3: Object detection

In this stage we detect the region containing the images that match the dataset. The object tracking algorithms can be used for detection of objects in every frame. By object detection we means that locating the object in a frame or in other words finding location of objects through a type of technology with the use of computer. The frame may be any random frame. Only objects having features that are in dataset detected and all others types of objects that are not present in the dataset are ignored.

Step 4:Source Code

```
from imutils.video import VideoStream
import numpy as np
import sys
import argparse
import imutils
import time
import cv2
```

```
from urllib.request import urlopen


host = 'http://192.168.0.101:8080/'
url = host + 'shot.jpg'

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("--prototxt", required=True,
help="path to Caffe 'deploy' prototxt file")
ap.add_argument("--model", required=True,
help="path to Caffe pre-trained model")
ap.add_argument("--source", required=True,
help="Source of video stream (webcam/host)")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
"bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
"dog", "horse", "motorbike", "person", "pottedplant", "sheep",
"sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initialize the video stream, allow the cammera sensor to warmup,
print("[INFO] starting video stream...")

if args["source"] == "webcam":
```

```
vs = cv2.VideoCapture(0)

time.sleep(2.0)

detected_objects = []
# loop over the frames from the video stream
while True:
# grab the frame from the threaded video stream and resize it
# to have a maximum width of 400 pixels
if args["source"] == "webcam":
ret, frame = vs.read()
else:
imgResp=urlopen(url)
imgNp=np.array(bytearray(imgResp.read()),dtype=np.uint8)
frame=cv2.imdecode(imgNp,-1)
frame = imutils.resize(frame, width=800)
# grab the frame dimensions and convert it to a blob
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
0.007843, (300, 300), 127.5)

# pass the blob through the network and obtain the detections and
# predictions
net.setInput(blob)
detections = net.forward()

# loop over the detections
for i in np.arange(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the prediction
confidence = detections[0, 0, i, 2]

# filter out weak detections by ensuring the `confidence` is
# greater than the minimum confidence
```

```
if confidence > args["confidence"]:
# extract the index of the class label from the
# `detections`, then compute the (x, y)-coordinates of
# the bounding box for the object
idx = int(detections[0, 0, i, 1])
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")

# draw the prediction on the frame
label = "{}: {:.2f}%".format(CLASSES[idx],
confidence * 100)
detected_objects.append(label)
cv2.rectangle(frame, (startX, startY), (endX, endY),
COLORS[idx], 2)
y = startY - 15 if startY - 15 > 15 else startY + 15
cv2.putText(frame, label, (startX, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
# show the output frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF
# if the `q` key was pressed, break from the loop
if key == ord("q"):
break
# do a bit of cleanup
cv2.destroyAllWindows()
```

Step 5: Result Analysis

After the application of the algorithm, the input is taken and displayed and shown in chapter 6.

Chapter 6

Results

In this section, we show experimental results of OpenCV object tracking method. The proposed method has been tested and evaluated in a series of image sequences demonstrating challenging tracking scenarios. Results from several representative input video sequences are presented below. The first such image sequence consists of 1280 frames and shows a person moving around. Characteristic snapshots demonstrating tracking results are shown in fig no.6.1 and 6.2 .The sequence scenario is as follows.

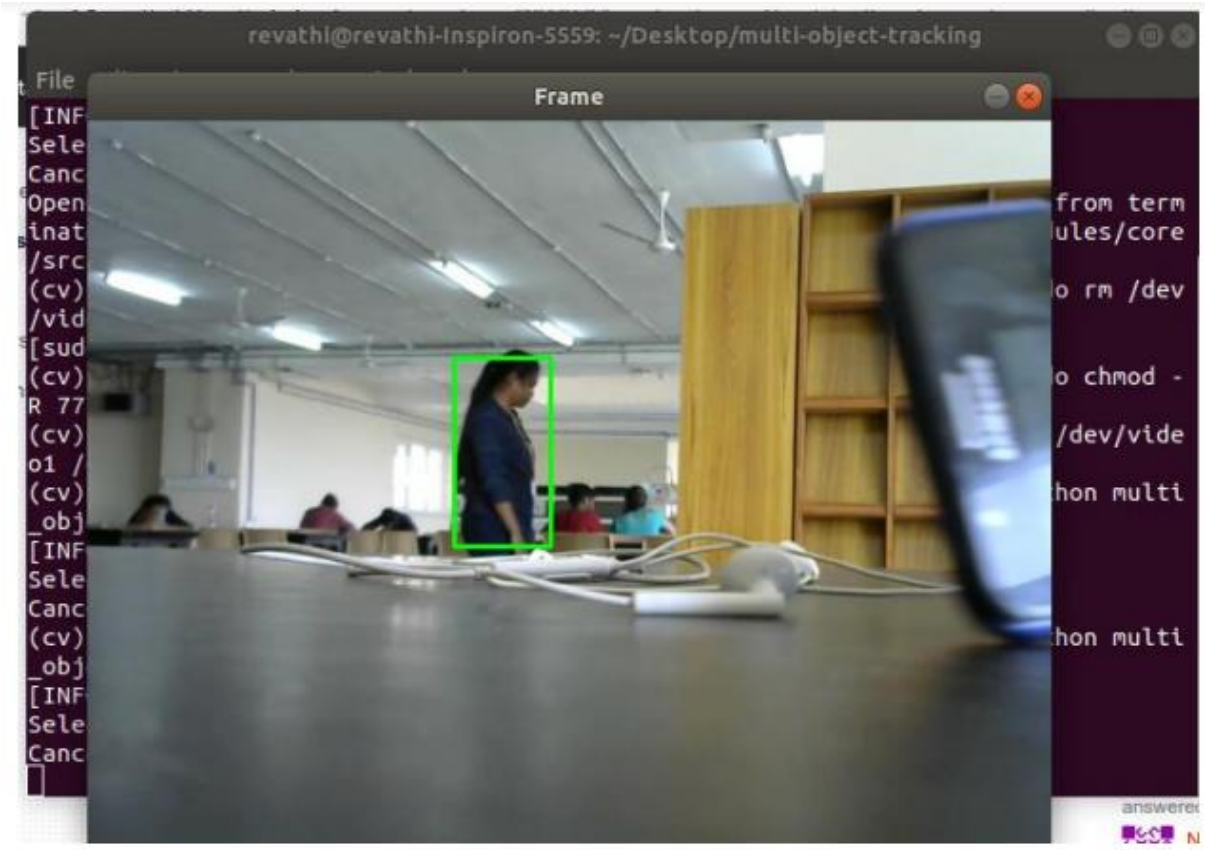


figure 6.1: Output obtained by level 1 functioning

The snapshot of the level 1 tracking result in fig 6.1 shows the tracking of a single person within the frame when manually selected.

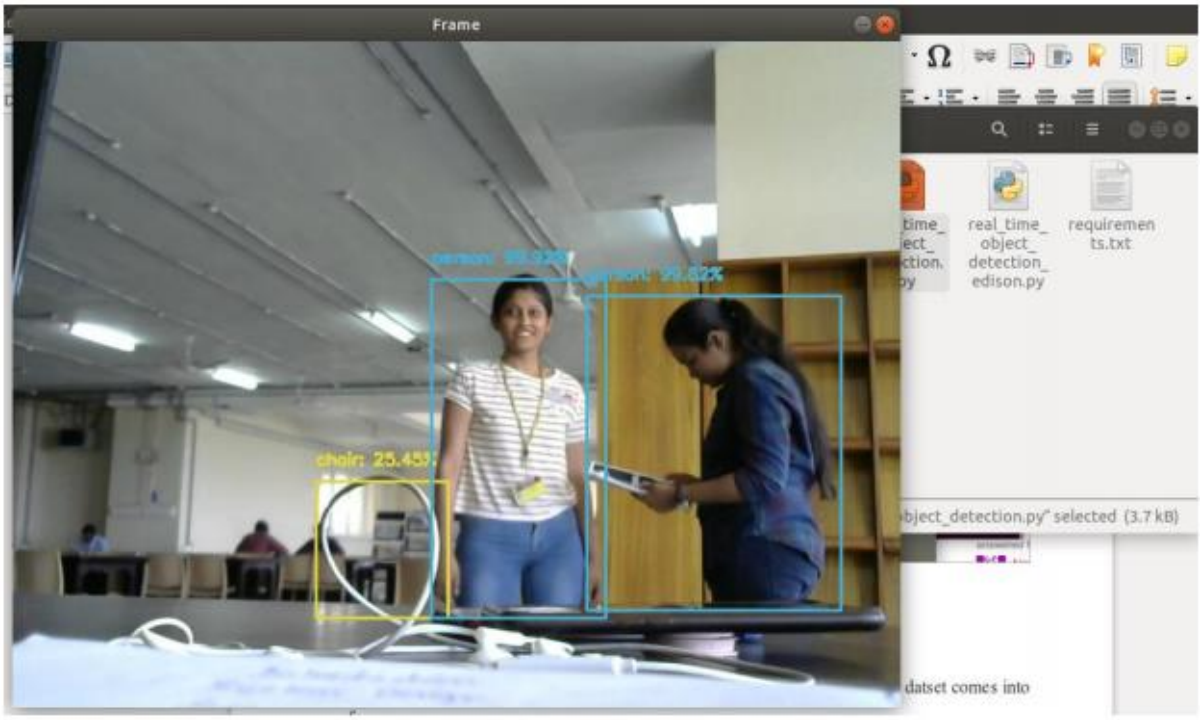


figure 6.2: Output obtained by second level functioning

The snapshot of the level 2 tracking result in fig 6.2 shows the Automatic tracking of multiple objects within the frame .

CHAPTER 7

CONCLUSION

Real time object detection and tracking on video stream is a very crucial topic of surveillance systems in field applications. To easily accessible product, the project constructed as low cost project. In this project, several methods are presented. We implemented different detecting and tracking methods. Algorithms works well for detection and tracking. The results are good for starting. Haar cascade method gives the most useful results for our project.

CHAPTER 8

REFERENCES

[1] Sangho Park and J. K. Aggarwal. Recognition of two-person interactions using a Hierarchical Bayesian Network. In IWVS ’03: First ACM SIGMM International Workshop on Video Surveillance, pages 65–76, New York, NY, USA, 2003. ACM Press.

[2] Ju Han and Bir Bhanu. Individual recognition using gait energy image. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(2):316–322, 2006.

[3] Sudeep Sarkar, Jonathon Phillips, Zongyi Liu, Isidro Robledo Vega Patrick Grother, and Fellow-Kevin W. Bowyer. The HumanID gait challenge problem: Data sets, performance, and analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(2):162–177, 2005.

[4] M. Petkovic, Z. Zivkovic, and W. Jonker. Recognizing strokes in tennis videos using Hidden Markov Models. In Proceedings of the IASTED International Conference Visualization, Imaging and Image Processing, Marbella, Spain, 2001.

[5] Nils T Siebel and Steve Maybank. Real-time tracking of pedestrians and vehicles. In Proceedings of the 2nd IEEE International Workshop on Perfor116 mance Evaluation of Tracking and Surveillance (PETS´2001), Kauai, USA, December 2001. CD-ROM proceedings.

[6] Deva Ramanan, David A. Forsyth, and Kobus Barnard. Building models of animals from video. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(8):1319–1334, 2006.

[7] Manjunath Narayana and Donna Haverkamp. A Bayesian algorithm for tracking multiple moving objects in outdoor surveillance video. In Fourth Joint IEEE International Workshop on Object Tracking and Classification in and Beyond the Visual Spectrum, In conjunction with IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2007, June 2007.

[8] Weiming Hu, Tieniu Tan, Liang Wang, and S. Maybank. A survey on Visual Surveillance of object motion and behaviors. IEEE Transactions on Systems, Man and Cybernetics, Part C, 34(3):334–352, 2004.

[9] Guohui Li, Jun Zhang, Hongwen Lin, D. Tu, and Maojun Zhang. A moving object detection approach using Integrated Background template for smart video sensor. In Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference, volume 1, pages 462–466, May 2004.

[10] Tao Zhao and Ram Nevatia. Tracking multiple humans in crowded environment. CVPR, 02:406–413, 2004.

[11] Chris Stauffer and W. Eric L. Grimson. Learning patterns of activity using real-time tracking. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8):747–757, 2000. 117

[12] Yaser Sheikh and Mubarak Shah. Bayesian modeling of dynamic scenes for object detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(11):1778–1792, 2005.

[13] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. Performance of optical flow techniques. CVPR, 92:236–242.

[14] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. Artificial Intelligence, 17(1-3):185–203, 1981.

[15] J. Wang and E. Adelson. Spatio-temporal segmentation of video data. In SPIE Proc. Image and Video Processing II, 1994.

[16] L. Biancardini, E. Dokadalova, S. Beucher, and L. Letellier. From moving edges to moving regions. In Proceedings Second Iberian Conference IbPRIA 2005, volume 1, pages 119–127, 2005.

[17] Gerald Kuhne, Stephan Richter, and Markus Beier. Motion-based segmentation and Contour-based classification of video objects. In MULTIMEDIA ’01: Proceedings of the Ninth ACM International Conference on Multimedia, pages 41–50, New York, NY, USA, 2001. ACM Press.

[18] S. Intille, J. Davis, and A. Bobick. Real-time closed-world tracking. IEEE CVPR, pages 697–703, 1997.

[19] P. Kumar, S. Ranganath, W.M. Huang, and K. Sengupta. Framework for real-time behavior interpretation from traffic video. ITS, 6(1):43–53, March 2005. 118

[20] Rudolph Emil Kalman. A new approach to Linear Filtering and Prediction Problems. Transactions of the ASME–Journal of Basic Engineering, 82(Series D):35–45, 1960.

[21] Dieter Koller, Joseph Weber, and Jitendra Malik. Robust multiple car tracking with Occlusion Reasoning. In ECCV (1), pages 189–196, 1994.

[22] Jianguang Lou, Tieniu Tan, and Weining Hu. Visual vehicle tracking algorithm. IEE Electronics Letters, 38(18):1024–1025, August 2002.

[23] Y. Bar-Shalom. Tracking and Data Association. Academic Press Professional, Inc., San Diego, CA, USA, 1987.

[24] Michael Isard and Andrew Blake. Condensation – Conditional density propagation for visual tracking. International Journal of Computer Vision, 29(1):5– 28, 1998.

[25] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. ACM Computing Survey, 38(4):13, 2006.

[26] Christopher Rasmussen and Gregory D. Hager. Probabilistic Data Association methods for tracking complex visual objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(6):560–576, 2001.

[27] Reinhard Koch. 3D-Scene modeling from image sequences. ISPRS Archives, 34, 2003.

[28] Reihard Koch, Marc Pollefrys, and Luc Van Gool. Realistic 3-D scene modeling from uncalibrated image sequences. In Proceedings of the International Conference on Image Processing, ICIP 99, pages 500–504, 1999. 119

[29] Karsten Muller, Aljoscha Smolic, Michale Drose, Patrick Voigt, and Thomas Weigand. 3-D reconstruction of a dynamic environment with a fully calibrated background for traffic scenes. IEEE Transactions on Circuits and Systems for Video Technology, 15(4):538–549, April 2005.

[30] Ali Azarbayejani and Alex P. Pentland. Recursive estimation of motion, structure, and focal length. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(6):562–575, 1995.Press. 121