VISION-ASSISTED OBJECT TRACKING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KEMAL ARDA ÖZERTEM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY, 2012

Approval of the thesis:

VISION-ASSISTED OBJECT TRACKING

submitted by **KEMAL ARDA ÖZERTEM** in partial fulfilment of the requirements for the degree of **Master of Science in Electrical and Electronics Enginering Department, Middle East Technical University** by**,**

Prof. Dr. Canan Özgen _____
Dean, Graduate School **of Natural and Applied Sciences**

Prof. Dr. İsmet ERKMEN _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. A. Aydın Alatan _____
Supervisor, **Electrical and Electronics Engineering, METU**

Prof. Dr. Mübeccel Demirekler _____
Co-Supervisor, **Electrical and Electronics Engineering, METU**

Examining Committee Members:

Prof. Dr. M. Kemal Leblebicioğlu _____
Electrical and Electronics Engineering, METU

Prof. Dr. A. Aydın Alatan _____
Supervisor, Electrical and Electronics Engineering, METU

Prof. Dr. Gözde Bozdağı Akar _____
Electrical and Electronics Engineering, METU

Assist. Prof. Dr. Afşar Saranlı _____
Electrical and Electronics Engineering, METU

Assist. Prof. Dr. Murat Efe _____
Electronics Engineering, Ankara University

Date: ___07.02.2012____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Kemal Arda Özertem

Signature:

ABSTRACT

VISION-ASSISTED OBJECT TRACKING

Özertem, Kemal Arda
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. A. Aydın Alatan

February 2012, 150 Pages

In this thesis, a video tracking method is proposed that is based on both computer vision and estimation theory. For this purpose, the overall study is partitioned into four related subproblems. The first part is moving object detection; for moving object detection, two different background modeling methods are developed. The second part is feature extraction and estimation of optical flow between video frames. As the feature extraction method, a well-known corner detector algorithm is employed and this extraction is applied only at the moving regions in the scene. For the feature points, the optical flow vectors are calculated by using an improved version of Kanade Lucas Tracker. The resulting optical flow field between consecutive frames is used directly in proposed tracking method. In the third part, a particle filter structure is build to provide tracking process. However, the particle filter is improved by adding optical flow data to the state equation as a correction term. In the last part of the study, the performance of the proposed approach is compared against standard implementations particle filter based trackers. Based on the simulation results in this study, it could be argued that insertion of vision-based optical flow estimation to tracking formulation improves the overall performance.

Keywords: Moving object detection, background modeling, feature extraction, optical flow, particle filter, video tracking.

ÖZ

GÖRME YARDIMLI NESNE TAKİBİ

Özertem, Kemal Arda
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. A. Aydın Alatan

Şubat 2012, 150 Sayfa

Bu tezde, bilgisayarla görme ve kestirim teorisine dayanan bir video takip yöntemi önerilmektedir. Bu amaçla, yapılan tüm çalışma birbiriyle ilişkili dört alt probleme ayrılmıştır. İlk bölüm hareketli nesne algılamasıdır; hareketli nesne algılaması için iki farklı arka plan modelleme yöntemi geliştirilmiştir. İkinci bölüm ise öznitelik çıkarımı ve video çerçeveleri arasındaki görsel akış tahmininden oluşmaktadır. Öznitelik çıkarımı yöntemi olarak, tanınan bir köşe algılayıcı algoritması kullanılmış ve bu çıkarım sadece sahnedeki hareketli bölgelere uygulanmıştır. Öznitelik noktaları için, görsel akış vektörleri Kanade Lucas takipçisinin geliştirilmiş sürümü kullanılarak hesaplanmıştır. Elde edilen ardışık çerçeveler arasındaki görsel akış alanı doğrudan önerilen takip yönteminde kullanılmıştır. Üçüncü bölümde, takip etme işlemini sağlaması için bir parçacık filtresi yapısı kurulmuştur. Fakat, parçacık filtresi durum denklemine düzeltme terimi olarak görsel akış verisinin eklenmesiyle geliştirilmiştir. Çalışmanın son bölümünde, önerilen yaklaşımın performansı standart parçacık filtresi tabanlı takipçilerle karşılaştırılmıştır. Elde edilen simülasyon sonuçlarına dayanarak, görsel akışın takip formülasyonunda kullanılmasının tüm performansı arttırdığı söylenebilir.

Anahtar kelimeler: Hareketli nesne algılaması, arka plan modellemesi, öznitelik çıkarımı, görsel akış, parçacık filtresi, video takibi.

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my supervisor, Prof. Dr. Aydın Alatan, for his guidance, stimulation, encouragement, friendship and sharing his experiences throughout the research. I regard it as a priviledge to work with him.

I am very grateful to my co-supervisor, Prof. Dr. Mübeccel Demirekler, for her contributions, constructive critisism and valuable comments on my study.

I would like to thank my colleagues from Roketsan for their help and friendship. I have learned much from their suggestions and experiences.

I am grateful to Emre Özkan for his constant support throughout my Ms Program at METU.

I also would like to also express my thanks to my colleague and my friend Seyit Tunç for discussions we had on both technical and non-technical matters.

I am thankful to my brother Umut Özertem for the inspiration and the motivation he gave to me. This thesis has never been accomplished without his support. He is always more than a usual brother for me.

Finally, I would like to express gratitude to my Dad for his endless love, support and patience over the years. He never stopped believing in me.

To the memories of my Ma and Grandma

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

LIST OF ABBREVIATIONS

ARMA:	AutoRegressive Moving Average
ASIR:	Auxilary Sampling Importance Resampling
EM:	Expectation Maximization
fps:	Frame Per Second
HMM:	Hidden Markov Model
KDE:	Kernel Density Estimation
KF2D:	Two Dimensional Kalman Filter
KF2D-OF:	Two Dimensional Kalman Filter with Optical Flow Data
KF4D:	Four Dimensional Kalman Filter
KLT:	Kanade Lucas Tracker
MSE:	Mean Square Error
PCA:	Principled Component Analysis
PETS:	Performance Evaluation of Tracking and Surveillance
PDF:	Probability Density Function
PF2D:	Two Dimensional Particle Filter
PF2D-OF:	Two Dimensional Particle Filter with Optical Flow Data
PF4D:	Four Dimensional Particle Filter
RGB:	Red Green Blue
RPF:	Regularized Particle Filter
SIR:	Sampling Importance Resampling
SIS:	Sequential Importance Sampling

CHAPTER 1

INTRODUCTION

Especially for security and surveillance, traffic control and medical imaging systems, video tracking becomes vital in everyday life. Remarkable improvements have been achieved in video tracking systems in the last few decades. As the computer performances increase significantly, computationally expensive algorithms can be used in video tracking and this provides a significant increase in the robustness of trackers. Furthermore, the availability of high quality and inexpensive video cameras on the market enables the users to get test videos easily. Thus, the performance of trackers can be tested rapidly for many different test cases.

Generally, most of the trackers suffer from noise in images, complex object motion, non-rigid structure of objects, the loss of information caused by projection of the 3D world into 2D image plane, partial or full occlusion, scene illumination changes and real-time processing requirements. A tracker cannot cope with all of these difficulties by itself. Therefore, according to the type of application, one should impose some constraints on the motion or appearance of objects and strengthen the tracking algorithm for the remaining problems. Furthermore, a tracker generally needs to know what to track in the scene. This step can be done automatically by the algorithm or manually by the user. Object detection algorithms help to trackers for detecting the objects in the scene automatically.

In the literature, there are various types of algorithms with different tracking approaches. Tracking algorithms can be divided mainly into two categories; computer vision based algorithms [3] and estimation theory based algorithms [33].

These two disciplines are different and slightly unaware of each other; however, their purposes are the same. In this thesis study, these two different approaches are examined and combined together for tracking problems.

1.1 OVERVIEW OF THE THESIS

In order to build a reliable tracker structure a synthesis of both computer vision based algorithms and estimation theory based algorithms are aimed in this study. Here, the background modeling, the optical flow calculation and feature extraction techniques are mostly accepted as computer vision based algorithms and the particle filter is a recent estimation theory based algorithm.

In order to obtain information about the objects in the scene, moving object detection is done initially. For the moving object detection, background modeling techniques are applied. These are namely background modeling using nonparametric Kernel Density Estimation (KDE) and Bayesian approach for object detection. These two algorithms and the conventional temporal median estimator are implemented to test video, which has a nominal camera motion. Among the results, the best results are due to Bayesian approach for object detection algorithm. For this reason, Bayesian approach is chosen to use for moving object detection in the remaining parts of this thesis work.

After the moving regions in the scene are determined, a well-known feature extraction algorithm, based on cornerness metrics for intensity images, is used. After the algorithm is test on the test video, the results show that the extracted feature points are quite satisfying. The reason behind the use of feature extraction approach arises from the necessity of the use of another computer vision based algorithm, optical flow. Since it would be computationally expensive to calculate optical flow for each pixel in every frame, it is calculated for only feature points. An improved version of Kanade Lucas Tracker (KLT) is employed to estimate optical flow vectors.

On the other hand, for the estimation theory based part, a generic particle filter structure is built. In order to test the reliability of the particle filter, a simple linear dynamic system is solved by both Kalman filter, which gives the optimal solutions, and the generic particle filter. The results are close enough for acceptable number of particles.

The key idea of this thesis work is putting the optical flow vector into particle filter's state equation as a correction term. The tracking performance is tested for three different cases. In the first case, the positions on x and y directions are used as two dimensional state vector. For the second case, the optical flow correction term is added to the first case. In the last case, the positions and the velocities on x and y directions are used as four dimensional state vector. The tracking results are compared in order to observe the effects of the optical flow term.

As a summary, the chapters of this thesis work correlated with each other. In other words, the output of the Chapter 2 is used in Chapter 3. Similarly, the outputs of Chapter 3 and 4 are used for a comparative analysis of tracking process in Chapter 5.

1.2 OUTLINE OF THE THESIS

The structure of the thesis follows the aforementioned four main steps.

Chapter 2 focuses on background modeling for moving object detection. First, some background information about background modeling is reviewed, and then a literature survey on background modeling is given. KDE is expressed and two background modeling techniques are introduced. Simulation results for these techniques are also given.

Chapter 3 is devoted to feature extraction and optical flow. Firstly, the optical flow is explained in detail, and then an improved version of KLT algorithm is introduced. Simulation results for this algorithm are presented also. Afterwards, as the feature extraction method "Shi-Tomasi corner detection" algorithm is presented

briefly. Lastly, the implementation results for “Shi-Tomasi corner detection” algorithm are stated.

In Chapter 4, detailed explanation of the particle filter is given. This chapter begins with introducing some basic concepts for particle filters and then the commonly used particle filter algorithms are expressed. The problems of particle filters are also examined in detail. The implementation of a generic particle filter is achieved and its simulation results are presented.

In Chapter 5, the methodology to combine these algorithms together is argued and the final method for video tracking is proposed. The tracking performance is investigated for three different cases. Especially, the effects of the optical flow term to tracking performance are observed.

Finally, Chapter 6 gives a summary of the thesis and concluding remarks.

CHAPTER 2

MOVING OBJECT DETECTION

Almost all visual tracking systems start tracking process with moving object detection. Moving object detection aims to segment regions corresponding to moving objects from the rest of a given image. Subsequent processes such as tracking are greatly dependent on accurate detection of moving objects. A common approach for object detection is to use information in a single frame. However, some object detection methods make use of the temporal information computed from a sequence of frames to reduce the number of false detections. This temporal information is usually in the form of frame differencing, which highlights changing regions in consecutive frames. If the object regions in the image are defined, it is then the tracker's task to perform object correspondence from one frame to the next to generate the tracks.

2.1 Modeling the Background

Accurate detection of moving objects is a necessary step to have a stable tracking and the process of moving object detection usually involves background modeling. In many vision systems (e.g. surveillance systems), typically, stationary cameras are used. Since the cameras are stationary, the detection of moving objects can be achieved by comparing each new frame with a representation of the scene background. This process is called background subtraction and the scene representation is called the background model.

An important issue in building a reliable background representation is choosing the features to be used in this representation. In the literature, a variety of features have

been used for background modeling, including pixel-based features (e.g., pixel intensity, edges and disparity) and region-based features (e.g., block correlation). The selection of the features affects how the background model tolerates changes in the scene and the granularity of the detected foreground objects [1].

The fundamental assumption that the sensor remains stationary among each consecutive frame allows using statistical background modeling techniques for the detection of moving objects [2]. However, this assumption does not necessarily imply a stationary background. In any indoor or outdoor scene, there are changes that occur with time. It is important that the background model can tolerate these changes, either by being invariant to them or by adapting itself to them. The changes can be local, affecting only part of the background such as moving tree branches, or global, affecting the entire background such as changes in illumination. Recently, non-adaptive methods for background modeling become less popular because of the need for a manual initialization. Without reinitialization, errors in the background accumulate over time, and because of this reason; non-adaptive methods are useful only in highly supervised, short-term tracking applications without significant changes in the scene [3].

The changes in the scene clearly affect the performance of the background modeling algorithm, and the study of these changes is essential to understand the motivations behind different background modeling techniques. Illumination based changes may be gradual due to the location change of the sun. It also might be sudden due to switching the lights on/off, or a change between cloudy and sunny conditions. Motion based changes can occur due to small camera displacements, which is caused by wind or ground vibration. There also might be a motion in parts of the background; e.g., tree branches moving with the wind or rippling water. A robust background modeling algorithm should consider all of these conditions.

2.1.1 Related Work on Background Modeling

Since late 70's, differencing of adjacent frames in a video sequence has been used for object detection in stationary cameras. However, later it was realized that

straightforward background subtraction was unsuitable for many of real-world situations; e.g., lightning conditions change over time and the background itself contains movement. Thus, some statistical techniques were introduced to model each background pixel in order to cope with these difficulties.

Many researchers have proposed methods to address some of the issues with background modeling. The proposed methods can be classified into two main categories, as the *methods which use pixel-based features* and the *methods which use region-based features*. Most of the work on background modeling falls into the first category, since pixel-based features are more suitable for background modeling. In region-based approaches, the detection unit is a whole image block and therefore, they are only suitable for coarse detection. A brief review of relevant work is provided here.

2.1.1.1 Pixel-Based Background Modeling Methods

Among all the other features, pixel intensity is the most commonly used feature in background modeling. It became popular following the work of Wren et al. [4]. If the intensity value of a pixel is observed over time in a completely static scene, then the pixel intensity can be reasonably modeled with a Gaussian distribution. In [4], Wren et al. propose modeling the color of each pixel, $I(x,y)$, of a stationary background with a single three dimensional Gaussian, $I(x,y) \sim N(\mu(x,y), \Sigma(x,y))$, and the model parameters, the mean and the covariance can be learned from color observations in consecutive frames. Then, the pixel-wise background model is derived and the likelihood of each pixel color can be computed. Hence, each pixel can be labeled, whether it belongs to the background or not. This model can also be adapted to slow changes in the scene using a simple adaptive filter by recursively updating the model. Similar approaches that use Kalman Filtering for updating are also proposed in [5] and [6].

However, it was promptly realized that the single Gaussian PDF for modeling the uncertainty of each pixel color was unsuitable to most outdoor situations. In outdoor environments, the scene background is typically not completely static. Repetitive

motions of background objects, shadows or reflectance often causes multiple pixel colors which belong to the background. Therefore, a single Gaussian assumption for the PDF of the pixel intensity will not hold. A substantial improvement in background modeling is achieved by using multimodal statistical models to describe per-pixel background color [7]. Friedman and Russell, [8], and independently Stauffer and Grimson, [9], [3] proposed modeling the intensity of each pixel as a mixture of Gaussians. In [9], the pixel intensity was modeled by a mixture of K number of Gaussian distributions (K is typically a small number ranging from 3 to 5). The mixture is weighted by the frequency with each of the Gaussians, which models each background pixel. In [3], a pixel in the current frame is compared to every Gaussian density in the background model. If these two parameters match, the mean and variance of the matched Gaussian distribution are updated. Otherwise, a new Gaussian with the mean, which is equal to the current pixel color and some initial variance, is added into the mixture. Thus, each pixel can be classified based on whether the matched distribution represents the background process. Similarly, in [8], a mixture of three Gaussian distributions is used to model the pixel value for traffic surveillance applications. The pixel intensity is modeled as a weighted mixture of three Gaussian distributions corresponding to road, shadow, and vehicle distribution, and the adaptation of the Gaussian mixture models can be achieved using an incremental version of the EM algorithm. Haritaoglu et al. [10] build a model of background variation that is a bimodal distribution constructed from order statistics of background values during a training period. The background modeling is achieved by representing each pixel by its minimum and maximum intensity values, and the maximum intensity difference between consecutive frames observed during the training period. For adaptation, these three values are updated periodically in time.

All of the models mentioned above are based on statistical modeling of pixel intensity with the ability to adapt the model to the changes in the background. Pixel intensity is not invariant to illumination changes; model adaptation makes it possible for such techniques to adapt to gradual changes under illumination. However, a sudden change in illumination can cause problems for these models.

Another limitation of these approaches is the need to specify the number of Gaussians, for the EM algorithm or the *K*-means approximation.

Furthermore, Gaussian mixture models do not explicitly model the spatial dependencies of neighboring pixel colors that may be caused by nominal motion. In order to deal with this problem, El Gammal et al. [1] propose nonparametric estimation methods for per-pixel background modeling. In [1], kernel density estimation (KDE) is used to establish foreground/background membership, and since KDE is a data-driven method, multiple modes in the intensity of the background are also handled naturally without any need for parameter tuning, e.g. the number of Gaussians in the Gaussian Mixture Model. Each pixel in the current frame is matched not only to the corresponding pixel in the background model, but also to the nearby pixel locations. Thus, this method can handle camera jitter or small movements in the background. Similarly in [2], the background data is modeled as a single distribution by using nonparametric density estimation methods over a joint domain-range representation. Therefore, the multi-modal spatial uncertainties can be directly handled. This method can provide high levels of detection accuracy in the presence of nominal camera motion and dynamic textures; however, they have the disadvantage that they require a significant amount of computational time, which limits their use in real-time systems in practice.

An alternate approach for background modeling is to represent the intensity variations of a pixel in an image sequence as discrete states corresponding to the events in the environment [7]. Hidden Markov models (HMMs) have been used for this purpose in [11] and [12]. In [11], a three-state HMM is used to model the intensity of a pixel for a traffic-monitoring application, where the three states correspond to the background, shadow, and foreground. By using HMM, Rittscher et al. classified small blocks of an image as belonging to one of these three states. In [12], Stenger et al. used HMMs for making a decision in environments which have two global states that are the arrival and departure of a train for outdoor application and two positions of a light source for indoor application. Certain events, which are difficult to model by using unsupervised background modeling techniques, can be modeled by the help of HMMs by using supervised training samples. Additionally,

it imposes a temporal continuity constraint on the pixel intensity. Therefore, a foreground pixel is expected to remain as a part of the foreground for a period of time before it becomes a part of the background again.

Alternatively, edge features have also been used to model the background. The advantage of using edge features is that edge features are less sensitive to illumination changes compared to color features. However, the major drawback of using edge features is that it would only be possible to detect edges of foreground objects instead of the dense connected regions. In [13], foreground edges are detected by comparing the edges in each new frame with an edge map of the background.

2.1.1.2 Region-Based Background Modeling Methods

The second category of methods uses region models of the background. Compared to the pixel-based methods, there are less publications in the literature that exploit region-based methods for background modeling. In [14], Toyama et al. proposed a three tiered algorithm that used region-based scene information in addition to per-pixel background model, in which region and frame level information serve to verify pixel-level inferences. At the pixel level, the preliminary classifications of foreground versus background are achieved by Wiener filtering. Afterwards at the region level, inter-pixel relationships are considered and foreground regions consisting of homogeneous color are filled in. The frame level watches for a sudden change and if most of the pixels in a frame are exposed to a sudden change, it is assumed that the pixel-based color background models are no longer valid.

Another global method proposed by Oliver et al. [15] employs eigendecomposition of sample images to detect objects. For k input frames of size $n \times m$, a matrix B of size $k \times (nm)$ is formed by row-major vectorization of each frame and eigenvalue decomposition is applied to the covariance of B as given in below

$$C = (B - \mu)^T (B - \mu) \quad \textbf{(1)}$$

The background is then represented by the most descriptive η eigenvectors that describe all possible illuminations in the field of view. Thus, this approach is much less sensitive to illumination. For the detection of the foreground objects, first current image is projected onto eigenspace, and then the Euclidean distance between the input image and the projected image is calculated and thresholded.

In order to deal with time-varying background, Monnet et al. [16] and Zhong et al. [17] simultaneously proposed models of image regions as an autoregressive moving average (ARMA) process, which provide a methodology to learn (by using PCA) and predict the motion patterns in a scene. An ARMA process is a time series model that consists of sums of the autoregressive and the moving-average components, where an autoregressive process can be described as a weighted sum of its previous values and a white noise error [7].

Block-based approaches have been also used for modeling the background. Block matching has been widely used for change detection between consecutive frames. Hsu et al. [18] fit each image block to a second-order bivariate polynomial, and the remaining variations are assumed as noise. In order to detect blocks with statistically significant changes, statistical likelihood is used. In [19], each block was represented with its median template over the background learning period and its block standard deviation. In order to detect the objects, each block is correlated with its corresponding template, and blocks, which have relatively higher deviation compared to the measured standard deviation, are considered to be foreground.

2.1.2 Kernel Density Estimation

Statistical modeling, where a process is modeled as a random variable in a feature space with an associated PDF, is a useful tool for background modeling. The PDF can be represented parametrically by using a specified statistical distribution that is assumed to approximate the actual distribution, with the associated parameters estimated from training data. Alternatively, nonparametric approaches, which

estimate the PDF directly from the data without any assumptions about the underlying distribution, can also be used. This feature prevents from choosing a model and to estimate its distribution parameters.

KDE (also known as Parzen windowing) is a non-parametric way of estimating the PDF of a random variable. The disadvantages of histograms provide the motivation for kernel density estimators. As a problem, the histograms are not differentiable, and they depend on the width and the end points of the bins. For removal of the dependence on the end points of the bins, kernel estimators place a kernel function at the center of each data point. If a smooth kernel function is chosen, then the result will be a smooth density estimator, and for the bin-width problem, there are many methods for finding the optimum bandwidth of kernel density estimator [20], [21]. In this technique, the underlying PDF is estimated as

$$f(x) = \sum_i \alpha_i K(x - x_i) \quad \textbf{(2)}$$

where K is a "kernel function" centered at the data points in feature space x_i, $i = 1 \ldots n$, and α_i are weighting coefficients. In many applications, Gaussian kernel function and uniform weights are used. Kernel density estimators asymptotically converge to any density function and this property makes these techniques quite general and applicable to many vision problems, where the underlying data densities are not known.

Given a sample $S = \{x_i\}_{i=1 \ldots N}$ from a distribution with density function $p(x)$, an estimate $\hat{p}(x)$ of the density at x can be calculated using the relation:

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} K_\sigma (x - x_i) \quad \textbf{(3)}$$

where K_σ is a kernel function (or a window function) with a bandwidth σ as shown below:

$$K_\sigma (t) = (1/\sigma) K(t/\sigma) \quad \textbf{(4)}$$

Kernel function is a non-negative symmetric function that integrates to one, and the bandwidth of the kernel is a smoothing parameter. A kernel function $K(w)$ must satisfy the following conditions [41]:

$$\begin{aligned} &\int K(w)dw = 1, \\ &\int wK(w)dw = 0, \\ &\int ww^T K(w)dw = I \end{aligned} \qquad \textbf{(5)}$$

The first equation given in (5) accounts for the fact that the sum of the kernel function over the whole region is unity. The second equation imposes the constraint that the means of the marginal kernels $K_i(w_i)$, are all zero. Finally, the third equation states that the marginal kernels are all pairwise uncorrelated and that each has unit variance.

Equation (3) can be thought as estimating the PDF by averaging the effect of a set of kernel functions centered at each data point. Alternatively, since the kernel function is symmetric, this computation can also be thought as averaging the effect of a kernel function centered at the estimation point and evaluated at each data point. For KDE, the simplest approach would be to use a fixed bandwidth for all the samples. Although such an approach is a reasonable compromise between complexity and the quality of approximation, the use of variable bandwidth can usually lead to an improvement in the accuracy of the estimated density. Intuitively, it is desired to choose the bandwidth as small as the data allows; however, there is always a tradeoff between the bias of the estimator and its variance. Smaller bandwidth is more appropriate in regions of high density, since a larger number of samples enable a more accurate estimation of the density in these regions. On the other hand, a larger bandwidth is more appropriate in low density areas where few sample points are available.

For higher dimensions, products of one dimensional kernels can be used as given in Equation 6,

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N}\prod_{j=1}^{d} K_{\sigma_j}\left(\frac{(x - x_i)_j}{\sigma_j}\right) \quad \textbf{(6)}$$

where the same kernel function is used in each dimension with a suitable bandwidth σ_j for each dimension. A range of kernel functions have been commonly used in the literature: uniform, triangular, biweight, triweight, Epanechnikov, Gaussian, etc. Gaussian kernel is preferred for its continuity, differentiability, and ease of use. Selection of the Gaussian as a kernel function is different from fitting the distribution to a Gaussian model. Here, the Gaussian is only used as a function to weight the data points. Unlike parametric fitting of a mixture of Gaussians, KDE is a more general approach that does not assume any specific shape for the density function. The major drawback of using the nonparametric kernel density estimator is its computational cost, since evaluating the PDF value at any point in the feature space requires a summation over all data samples.

2.2 Background Modeling Using Nonparametric Kernel Density Estimation

In this section, a background modeling technique which is similar to the method in [1] is described. Pixel intensity is used as the basic feature for modeling the background. A sample of intensity values for each pixel in the frame is stored by the model. These samples are used to estimate the density function of the pixel intensity distribution. The model can also estimate the probability that a newly observed intensity value belongs to foreground/background. The model is updated periodically at each frame. Thus, it can deal with the cases where the background is not completely static; however, it contains small motions, such as camera jitter or moving objects in the background, like tree leaves, waves etc.

Let $x_1, x_2, \ldots, x_N$ be a sample of intensity values for a pixel. The pixel intensity PDF can be estimated by implementing kernel density estimation to these samples. Here, the Gaussian kernel is chosen. Therefore, the density can be estimated as,

$$\Pr(x_t) = \frac{1}{N}\sum_{i=1}^{N}\prod_{j=1}^{d}\frac{1}{\sqrt{2\pi\sigma_j^2}}e^{-\frac{1}{2}\frac{(x_{t_j}-x_{i_j})^2}{\sigma_j^2}} \quad \textbf{(7)}$$

where x_t is a d-dimensional color feature and $K_{\sigma j}$ is a kernel function with bandwidth σ_j in the j^{th} color space dimension. A pixel in the current frame is labeled as foreground if the probability estimate is under a threshold value. This threshold is global and applied over all the frames. Equation (7) is computationally expensive; however, since most of the image is covered with background, for most of the pixels, the partial sum will pass over the threshold quickly. This situation helps obtaining a faster implementation. The most recent N sample frames are used for kernel density estimation and the adaptation is achieved by accepting recent samples and forgetting earlier samples.

Kernel bandwidth is a critical parameter for kernel density estimation. If the bandwidth is chosen too small or too high, then the result will be a ragged or an over-smoothed density estimate. Here, an adaptive bandwidth [1], which is based on intensity values of the samples, is used. By using the samples, the median m of $|x_i - x_{i+1}|$ for each consecutive pair (x_i, x_{i+1}) is calculated for each color channel independently. Using the median of the absolute deviation helps to handle with pixel intensity changes, which is caused from non-stationary background. Therefore, the adaptive bandwidth can be selected as [1]

$$\sigma = \frac{m}{0.68\sqrt{2}} \quad \textbf{(8)}$$

As it is well known, 3 color channels; red, green and blue (RGB) are strongly correlated. Instead of using these 3 color channels, the same amount of color information could be carried by using only 2 color channels. This will also decrease the computational load, and leads to a faster implementation. For these reasons, chromaticity coordinates are defined in (9) by normalizing RGB channels [22].

$$r = \frac{R}{(R+G+B)}$$
$$g = \frac{G}{(R+G+B)} \quad \textbf{(9)}$$
$$b = \frac{B}{(R+G+B)}$$
$$r+g+b=1$$

Therefore, for the kernel density estimation 2 dimensional color feature, (r, g) is only utilized. Employing chromaticity coordinates also helps for not detecting the small changes in illumination, which generally causes from shadows. On the other hand, using chromaticity coordinates has the disadvantage of losing information about illumination. Due to this reason, a measure of illumination at each pixel shown in (10) is included. The shadow removal is achieved by using a threshold to the ratio of the current lightness value and the expected lightness value.

$$s = R+G+B \quad \textbf{(10)}$$

For suppressing the false detections due to small motion in the image, another measure is defined. Let x_t be the observed value of the pixel x, which is detected as foreground at time t. If there is a background pixel in the neighborhood of x, then, the maximum probability $P_N(x_t)$ is also checked. Therefore, a detected pixel x is considered to be a part of the background only if $P_N(x_t)$ is above a certain threshold. 8-neighborhood is used here for suppressing the false detections. A step-by-step implementation of this method is given in Table 1.

Table 1 – Main algorithm of background modeling using KDE

- Import N sample frames
- Calculate r, g and s (by using Equation (9) and (10)) individually for each pixel in the samples
- For r and g dimensions, calculate the median of $| x_i - x_{i+1} |$
- Calculate the bandwidth by using Equation (8)
- Calculate the density by using kernel density estimation given in (7)
- If the probability for a pixel is under a certain threshold, then the pixel is labeled as foreground
- Perform shadow removal by using a threshold to the ratio $s_{current}/s_{expected}$
- Apply suppression of the false detection for each foreground pixel by using a threshold to $P_N(x_t)$

2.3 A Bayesian Approach for Object Detection

In this section, a Bayesian background modeling approach, which is similar to the method in [2] is described. The main idea is that there is a useful relation between the intensities of the neighboring pixels, and this dependency can be used for detection for the scenes which has a non-stationary background or nominal camera motion. As described in [2], a temporal persistence criterion is taken into consideration for accurate detection. According to temporal persistence, true foreground objects keep their colors and spatial positions in time, which means their color transformation and motion changes slowly. Additionally, the foreground information at time t will be used at time t+1.

In this method, the background and the foreground are modeled individually. And the object detection is maintained by using a likelihood ratio classifier. The features are represented by a joint domain-range representation, where spatial coordinates (x,y) is the domain and color space (r,g,b) is the range. By using nonparametric kernel density estimation technique over the joint domain-range representation, the entire background can be modeled as a single distribution $f_{R,G,B,X,Y}(r,g,b,x,y)$. For building the background model, it is assumed that, before time t all pixels in the sample set $\psi_b=\{y_1, y_2 \ldots y_n\}$ belong to the background. With the help of this sample

set, the probability of each pixel belonging to the background at time t can be calculated by using kernel density estimator. As described in Section 2.2, the Gaussian function is selected as the kernel function; therefore, the kernel density estimator is similar to previous method given in (7).

First, the initialization of the background model is set to zero and the initialization of the foreground model is set to a uniform function. Furthermore, the number of bins in each dimension is adjusted for background and foreground densities. The assumption is that there are no objects until time t, so that the background PDF can be learned by using the sample set of first t frames. During the initial learning stage, 5 dimensional data for each pixel in the sample set is settled by putting ones in the background PDF. For taking into consideration of correlation in intensities of neighboring pixels, weighted values, which is smaller then one, are given to neighboring bins in the PDF.

At any time instance, the probability of observing a foreground pixel at any location (i,j) of any color is uniform. Once a foreground region is been detected at time t, there is an increased probability of observing a foreground region at time t+1 in the same neighborhood with similar color distribution. Therefore the foreground probability consists of a mixture of a uniform density function and the kernel density function that is estimated from the samples of the foreground as given below [2]:

$$P(x \mid \psi_f) = \alpha\gamma + (1-\alpha)m^{-1}\sum_{i=1}^{m}\varphi_H(x - z_i) \quad \textbf{(11)}$$

where $\alpha << 1$ is the mixture weight and γ is a random variable with uniform probability. If an object is detected in the previous frame, then α becomes a quite small number (in the implementation of the algorithm 0.01 is used) and the probability of observing the same colors of that object in the same neighborhood increases with respect to the second term in (11). If there are no detected objects in the previous frame, then α remains being a relatively large number, which makes foreground probability a nearly uniform function. Very similar to initial learning stage, for the background probability, ones are given to 5 dimensional data, where

the background is detected and some weighted values are given to their neighboring bins. Therefore, both background and foreground models are updated at each frame. Two forgetting factors; one for foreground ρ_f and another one for background ρ_b are introduced to keep the algorithm adaptive to the changes in the scene. In order to obtain adaptation, foreground and background models do not memorize all the past frames; they forget the frames with respect to their forgetting factors. Since the foreground changes faster than the background; ρ_f is typically higher than ρ_b.

For deciding whether a pixel belongs to foreground, a likelihood ratio classifier is used as follows.

$$-\ln\frac{P(x\mid\psi_b)}{P(x\mid\psi_f)}=-\ln\frac{n^{-1}\sum_{i=1}^{n}\varphi_H(x-y_i)}{\alpha\gamma+(1-\alpha)m^{-1}\sum_{i=1}^{m}\varphi_H(x-z_i)} \quad \textbf{(12)}$$

If this likelihood ratio is less than a threshold value, the pixel is labeled as foreground. The threshold also balances the trade-off between sensitivity and robustness. A step-by-step implementation of the method is given below.

Table 2 – Main algorithm of Bayesian approach for object detection

- Initialize the background and foreground models
- Train the background by using the frames until time *t*, where there are no objects in the scene
- Decide foreground pixels by using likelihood ratio classifier as (12)
- Adjust the parameter α whether there is an object in the scene or not. Update the foreground probability with respect to (11)
- Remove the frame that was ρ_f frames before from the foreground model
- Update the values of the background pdf with respect to the detected background area and its neighborhood
- Remove the frame that was ρ_b frames before from the background model

2.4 Comparative Analysis of Moving Object Detection

In this section, evaluation results of different moving object detection methods are presented. An uncompressed test video, which has a dimension of 360 by 240 pixels, a frame rate of 15 fps and duration of 33 seconds, is selected for simulations. The same test video has been used during the experimental evaluations in [2]. The algorithms are tested in the presence of nominal camera motion, which consists of approximately 12 pixels. In the test video, there is a railroad scene, which has no foreground objects in the first 250 frames. Then a man and a car enter the scene from different sides. In addition, an occlusion occurs as they move towards each other. The test video is introduced in Figure 1.

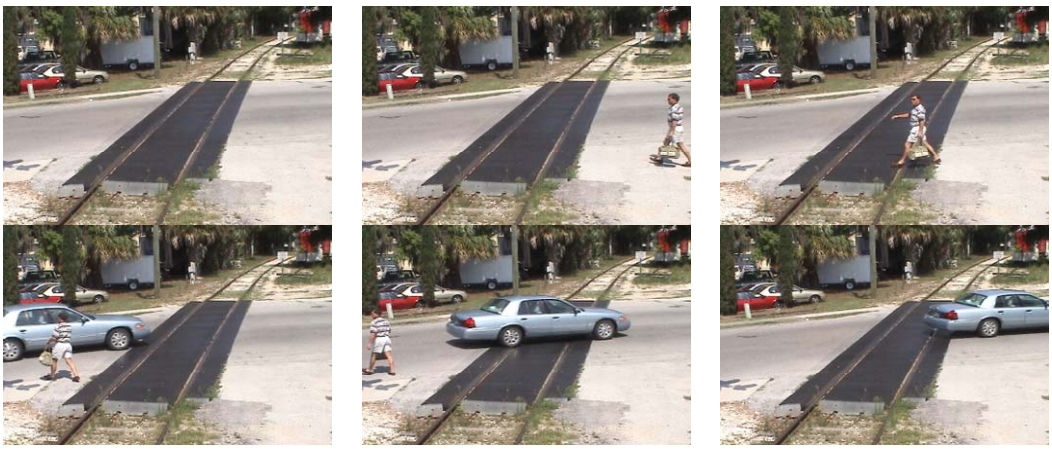

Figure 1 - Scenes from test video

Three different moving object detection methods are compared here. The first method is a simple temporal median estimator. It assumes the temporal median of the video as being the background model and subtracts current frame from this background model. By using a threshold to the result, it simply detects moving objects in the scene. The median size is an important parameter for temporal median estimator. If the median size is selected too small then the results would be very noisy. On the other hand, selecting a too large median size slows down the algorithm because sorting a large number of frames is computationally high. Additionally, it requires a lot of memory. Because of these reasons, after some experiments the median size is selected as 20 for this simulation. The results for temporal median estimator are shown in Figure 2.

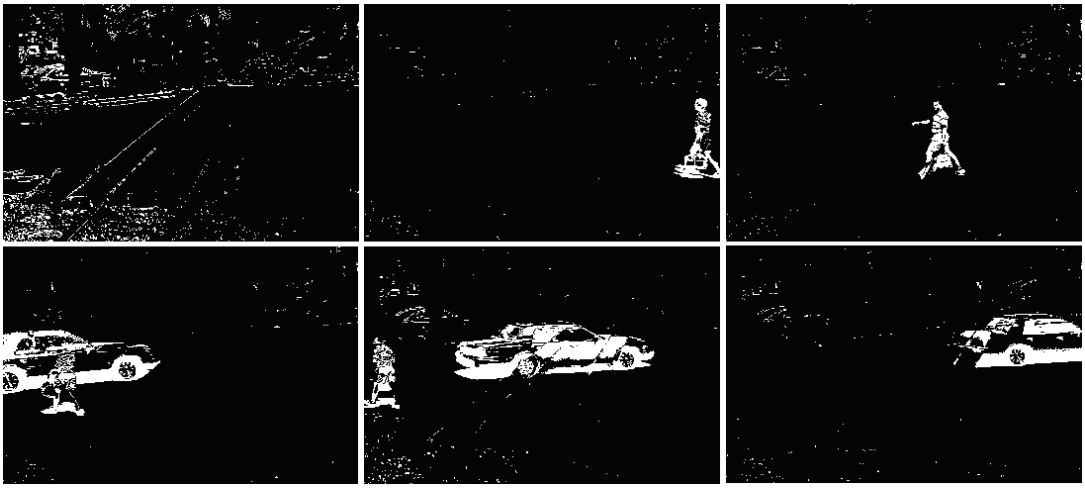

Figure 2 – Simulation results for temporal median estimator

When the simulation results for temporal median estimator are analyzed one can observe that temporal median estimator cannot handle nominal camera motion sufficiently. Especially, in the beginning of the video, when there are no objects in the scene, the detection performance is quite poor and a lot of false alarms occur due to camera motion. Additionally, there are some large holes in the foreground objects, which is also not quite desirable.

The results for background modeling using nonparametric KDE method are presented in Figure 3.

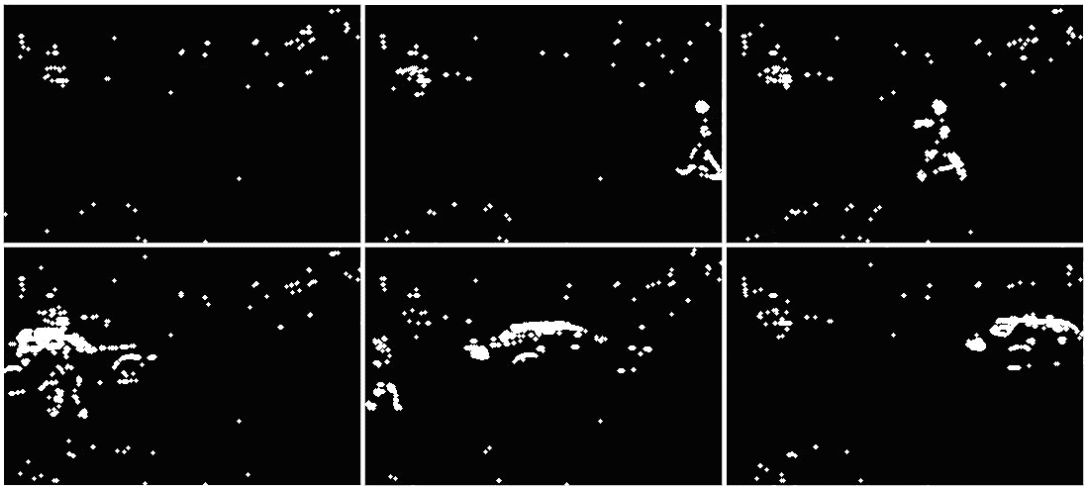

Figure 3 - Simulation results for background modeling using nonparametric KDE

First 200 frames of the test video are given as samples for background modeling using nonparametric KDE method. The simulation results have similarity with previous method. The results are still noisy, mainly due to the nominal camera motion. While the algorithm tries to get rid of the effects of nominal camera motion, it loses some object points. Similarly, the objects are not detected as a whole block; contrarily, there are still big holes in the objects. However, it gives better results if it is compared to temporal median estimator, especially when there are no objects in the scene.

The results for Bayesian approach for object detection method are given Figure 4.

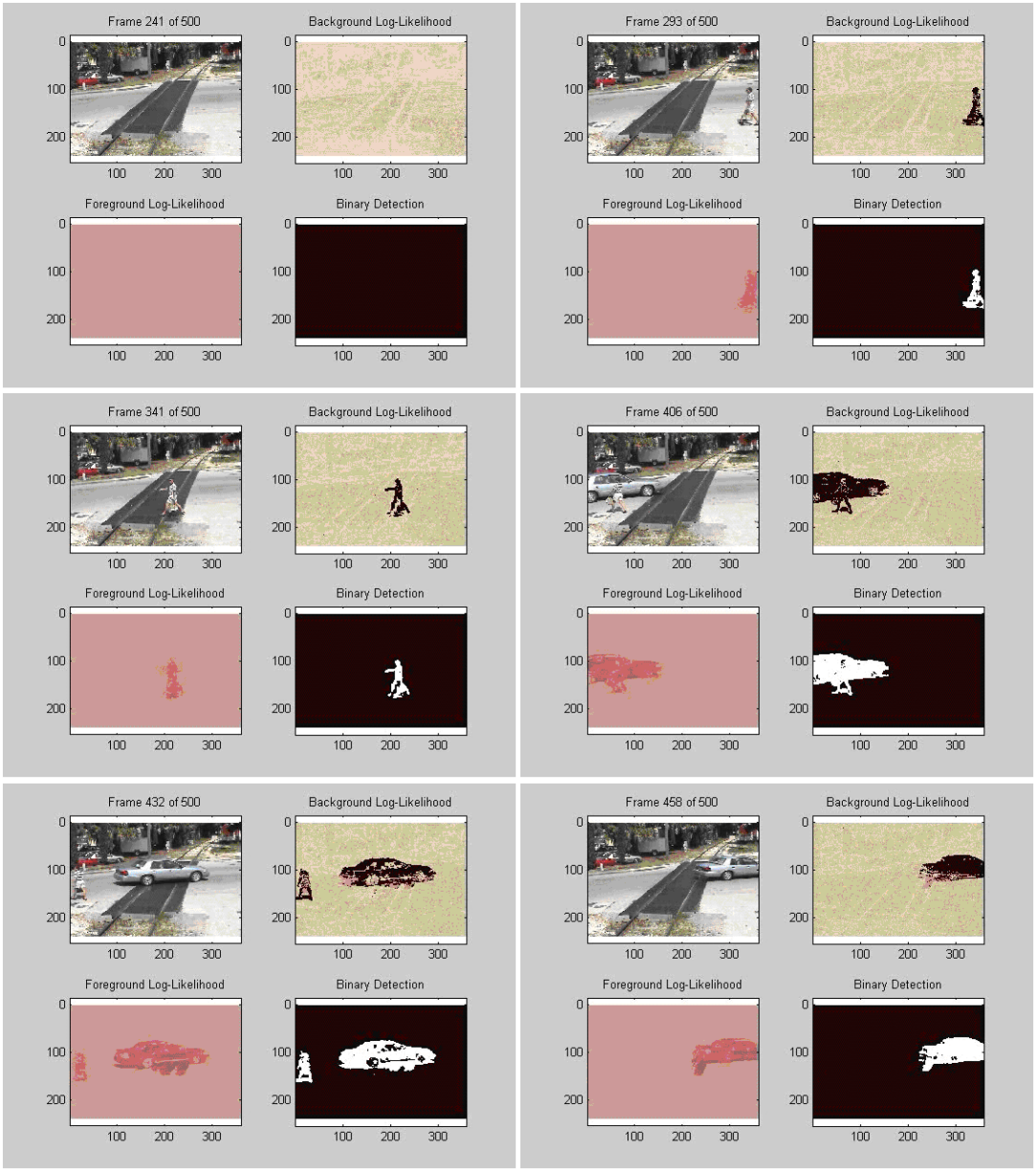

Figure 4 – Simulation results for Bayesian approach for object detection

Similarly, first 200 frames are assumed to be a sample set for Bayesian approach for object detection method. Background and foreground log likelihood results with respect to (12) are also presented in the simulation results. When both the foreground and the background are modeled by KDE simultaneously, the results show a clear improvement over the two earlier methods. First, the simulation results show that, the method is robust for nominal camera motion. Other methods, temporal median estimator and background modeling using KDE, have some difficulties to handle the nominal camera motion and misclassify some background

points, as foreground. However, Bayesian approach remains robust against such false alarms for the whole video sequence and has a misdetection rate near zero. Furthermore, the objects in the scene are detected as a whole block with no large holes. This property provides also an important advantage for extracting the features by using optical flow data for object tracking.

To sum up the simulation results, the performance of temporal median estimator and background modeling using nonparametric KDE methods are relatively poor as compared to Bayesian approach that models both foreground and background densities at each frame. Bayesian approach provides a high detection rate, and has a very reasonable false alarm rate even in the presence of nominal camera motion. Due to these reasons, Bayesian approach is preferred for moving object detection part of this thesis work and the results presented in the following sections are based on this method.

CHAPTER 3

FEATURE EXTRACTION & KANADE LUCAS FEATURE TRACKER

Optical flow is a commonly used technique in object detection and tracking. However, it is computationally expensive and also unreliable to calculate optical flow for each pixel in every frame, which makes it unsuitable for real-time applications since dense estimation slows down the speed of the whole tracking system. In order to overcome this difficulty, a common approach to estimate optical flow only at sparse feature points will be utilized in this thesis. Feature extraction is simply transforming the input data into a reduced representation set of features. Feature extraction algorithm is only implemented to the output of previous chapter; the regions of moving objects. Additionally, optical flow is calculated only for feature points, which speeds up the tracking system significantly. For feature extraction, among many other alternatives, a well-known algorithm, namely 'Shi-Tomasi corner detection' will be used. Moreover, for the optical flow calculations, an improved version of Kanade Lucas Feature Tracker (KLT) will be used. In this chapter, first some brief information about related concepts are given and then the algorithms for feature extraction and optical flow are presented in full detail along with simulation results.

3.1 Image Pyramid Representation

Scenes may contain objects of many sizes or objects can be at various distances from the viewer. Naturally, these objects will have features of many sizes. Therefore, if the images are used only at a single scale for processing, then some information may be missed at other scales.

The image pyramid is a data structure designed to support efficient scaled convolution through reduced image representation. It consists of a sequence of copies of an original image in which both sample density and resolution are decreased in regular steps [23]. In other words, image pyramid representation is a multi-scale representation, in which images are exposed to repeated smoothing and subsampling. G_0, the bottom of the pyramid, is equal to original image and higher levels of the pyramid, which have a reduced resolution at every level, are obtained through a highly efficient iterative algorithm. G_0 is lowpass filtered and subsampled by a factor of two to obtain the next pyramid level, G_1. The remaining pyramid levels are generated with the same repetitions of the filter or subsample steps.

There are two main types of pyramids; lowpass pyramids and bandpass pyramids. In lowpass pyramid, firstly, to eliminate potential aliasing effects, the image is convolved by a smoothing filter to remove high frequency signal components. After this step the image is subsampled by a factor of two along each coordinate direction. From the bottom to the top of the pyramid, at every level the spatial sampling density decreases and the result will be a set of gradually more smoothed images. On the other hand, for the bandpass pyramid the difference between adjacent levels are used. To take the pixelwise difference between adjacent levels, there is an interpolation process between representations at adjacent levels of resolution.

Pyramid construction is equivalent to convolving the original image with a set of Gaussian-like weighting functions [23]. The weighting functions of lowpass pyramid closely resemble Gaussian density function. Due to this reason, lowpass pyramids are also known as Gaussian pyramids. Similarly, the weighting functions of bandpass pyramid are similar to Laplacian operator and bandpass pyramids are also known as Laplacian pyramids. Weighting functions of Gaussian and Laplacian pyramids for three successive levels are shown in Figure 5.

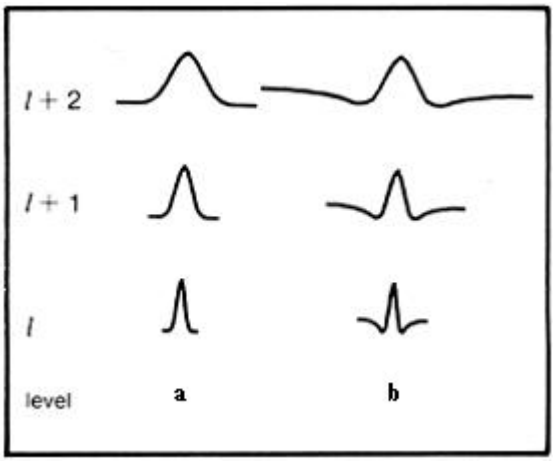

Figure 5 – Equivalent weighting functions. The functions for Gaussian pyramid are shown in (a) and the functions for Laplacian pyramid are shown in (b).

Let G_0, G_1,... are the levels of a Gaussian pyramid. To calculate the levels of Laplacian pyramid, one should obtain further level from current Gaussian pyramid level, and then predict current Gaussian level from further level. The error in this prediction gives the current Laplacian pyramid level. In other words, current Gaussian pyramid level G_l is reduced to G_{l+1}, and then G_l is estimated by expending G_{l+1}. The result of this estimation is G'_l and the difference between G_l and G'_l is equal to L_l, which is current Laplacian pyramid level. The typical outputs for Gaussian and Laplacian pyramids are given in Figure 6. The effects of lowpass filtering are clearly apparent for Gaussian pyramid. For the Laplacian pyramid, the edge features are enhanced and become more salient by size.

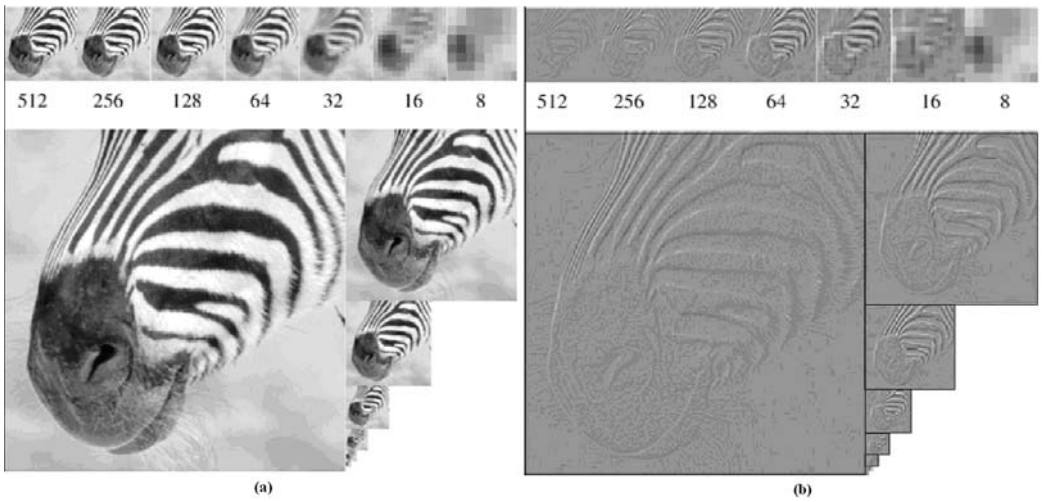

Figure 6 – Image pyramids for first six levels (a) Gaussian pyramid and (b) Laplacian pyramid

The pyramid offers a useful image representation for many areas. It is widely used in image enhancement, data compression, image analysis and computing multi-scale image features. In summary, image pyramid representation provides a convenient and useful multi-resolution format for the multiple scales in the visual scenes in all its aspects.

3.2 Optical Flow

Optical flow is the distribution of apparent velocities of movement of brightness patterns in an image [24]. In other words, optical flow is an approximation of the local image motion based upon local derivatives in a given sequences of images. It specifies how much each image pixel moves between adjacent images. Optical flow can be understood as a motion field; however, it is a totally different concept. To illustrate, assume a rotating Lambertian sphere with a static light source producing a static image. In this case, there is a nonzero motion field but the calculated optical flow is zero. Similarly, a stationary sphere with a moving light source produces drifting intensities. In this case, there is a zero motion field however; the calculated optical flow is nonzero. These examples are illustrated in Figure 7.

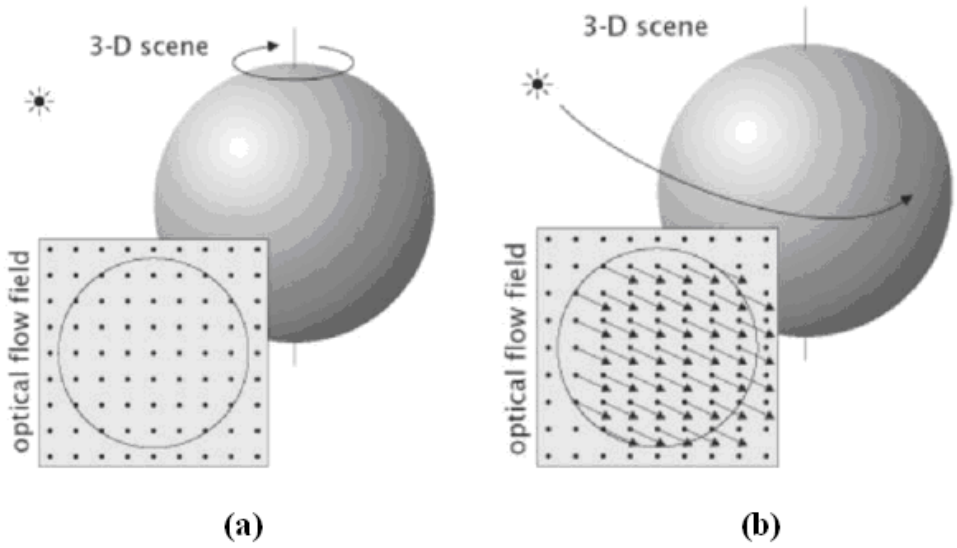

Figure 7 – In (a) the scene has nonzero motion field with zero optical flow field, in (b) the scene has zero motion field with nonzero optical flow field

Optical flow is based on the fundamental idea that, image radiance keeps the same at the next time instant for the corresponding point. It can be formulated as:

$$E(x+u\delta t, y+v\delta t, t+\delta t) = E(x,y,t) \tag{13}$$

where $E(x,y,t)$ denotes the intensity for location (x,y) at time t and (u,v) is the unknown velocity of the point at location (x,y). Since the motion field is continuous, first order Taylor Series Expansion can be used; this could be a valid assumption, if there is not a rapid motion in the scene, since all the higher order terms 'e' in the first order Taylor Series Expansion are assumed to be zero. By using first order Taylor Series Expansion given in (14), optical flow constraint equation shown in (16) can be easily obtained.

$$\begin{aligned} &E(x+u\delta t, y+v\delta t, t+\delta t) = E(x,y,t) \\ &E(x,y,t) + \delta x\frac{\partial E}{\partial x} + \delta y\frac{\partial E}{\partial y} + \delta t\frac{\partial E}{\partial t} + e = E(x,y,t) \\ &\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0 \end{aligned} \tag{14}$$

If the terms on the left hand side given in the 3rd condition in (14) are rearranged as given in (15) and inserted again into the 3rd condition in (14), finally, Equation (16) is obtained.

$$\frac{\partial E}{\partial x} = E_x \quad \frac{dx}{dt} = u \quad \frac{dy}{dt} = v \quad \frac{\partial E}{\partial t} = E_t \tag{15}$$

$$E_x u + E_y v + E_t = 0 \tag{16}$$

The aim of the optical flow is to estimate the (*u,v*) vector, which is shown in Figure 8. Unfortunately, the component of (*u,v*) vector, that is perpendicular to brightness gradient direction can not be observed; this is a physical phenomenon named *aperture problem*. Figure 9 shows one typical example for the aperture problem, where a line is moving up and to the right direction, is viewed through a circular aperture. In this case, it is impossible to recover the correct full image velocity, however, only the image velocity normal to the line [25].

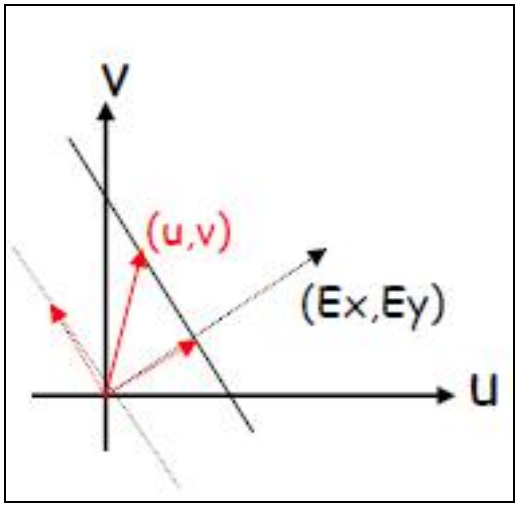

Figure 8 – Illustration of optical flow vector

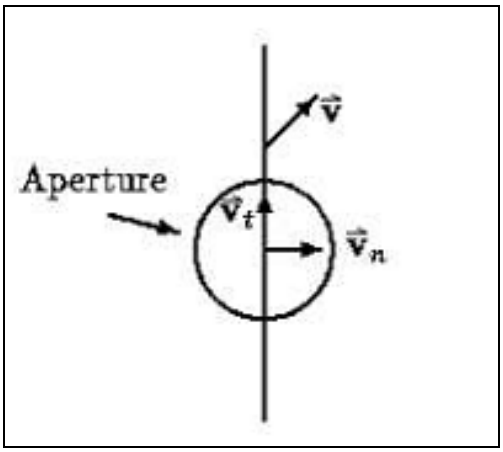

Figure 9 – An illustration for aperture problem

Since there is one equation with two unknowns, extra constraints, such as rigid body and smoothness of neighboring motion vectors assumptions should be introduced. Rigid body assumption assumes that all objects in the scene are rigid and no shape changes allowed. Smoothness assumption arises from the observation that neighboring pixels generally belong to the same surface. Therefore, it is assumed that neighboring pixels have nearly the same image motion. However, smoothness assumption usually fails at moving object boundaries. In that case, smoothness assumption, in the direction perpendicular to the boundary, can be stopped and directional smoothness constraint can be used. After these assumptions the problem is reduced to minimization of the relation below:

$$e = e_s + \lambda e_c \tag{17}$$

where e_s and e_c can be expressed as below

$$\begin{aligned} e_s &= \iint ((u_x^2 + u_y^2) + (v_x^2 + v_y^2))dxdy \\ e_c &= \iint (E_x u + E_y v + E_t)^2 dxdy \end{aligned} \tag{18}$$

Here e_s is the smoothness term and e_c is the error term. λ is a user defined regularization parameter. Minimization of (17) can be achieved by applying some calculus variations, such as Euler equations to (17) directly. Alternatively and more commonly, a discrete version of this integral equation can also be minimized. There are two well known approaches to minimize this equation and estimate the optical

flow. These are namely Horn and Schunk method [24] and Kanade-Lucas method [42]. In the next section, these methods are explained in detail.

3.2.1 Horn and Schunk Method

The Horn and Schunk method is an optical flow estimator which estimates motion based on the local gradient and local difference of two consecutive frames. To solve the aperture problem, a global smoothness constraint, which assumes smoothness in the flow over the whole image, is imposed. Horn and Schunk method achieves solutions, which show more smoothness, by minimizing the distortions in the flow. In order to achieve this aim, the cost function in (19), which is an extended version of (17), should be minimized.

$$e = \iint \left(\lambda \left(E_x u + E_y v + E_t \right)^2 + \left(\left| \nabla u \right|^2 + \left| \nabla v \right|^2 \right) \right) dx dy \qquad \textbf{(19)}$$

To go any further, the partial derivatives E_x, E_y, E_t and the Laplacian of the flow velocities $\nabla^2 u$, $\nabla^2 v$ are required. Firstly, the derivatives of brightness are estimated by using eight brightness measurements. It is crucial to estimate these derivatives consistently, which means they should refer to same point at the same time [24]. Therefore, the estimation of horizontal, vertical and time derivatives can be obtained as shown in (20), (21) and (22).

$$E_x \approx \frac{1}{4} \{ E_{i,j+1,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j,k.+1} + E_{i+1,j+1,k+1} - E_{i+1,j,k+1} \} \qquad \textbf{(20)}$$

$$E_y \approx \frac{1}{4} \{ E_{i+1,j,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i,j+1,k} + E_{i+1,j,k+1} - E_{i,j,k.+1} + E_{i+1,j+1,k+1} - E_{i,j+1,k+1} \} \qquad \textbf{(21)}$$

$$E_t \approx \frac{1}{4} \{ E_{i,j,k+1} - E_{i,j,k} + E_{i+1,j,k+1} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j+1,k.} + E_{i+1,j+1,k+1} - E_{i+1,j+1,k} \} \qquad \textbf{(22)}$$

In these equations, j corresponds to the x direction in the image, i corresponds to the y direction and k stands for the time domain. As mentioned before, the Laplacian of

the flow velocities should be also estimated. One convenient approximation takes the following forms given in (23) and (24).

$$\nabla^2 u \approx 3.\left(\bar{u}_{i,j,k} - u_{i,j,k}\right) \tag{23}$$

$$\nabla^2 v \approx 3.\left(\bar{v}_{i,j,k} - v_{i,j,k}\right) \tag{24}$$

where the local averages $\bar{u}$ and $\bar{v}$ are given below in (25) and (26).

$$\bar{u}_{i,j,k} = \frac{1}{6}\{u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k} + \frac{1}{12}\{u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k} \tag{25}$$

$$\bar{v}_{i,j,k} = \frac{1}{6}\{v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\} + \frac{1}{12}\{v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k}\} \tag{26}$$

The Laplacian is estimated by subtracting the value at a point from a weighted average of the values at neighboring points. Figure 10 illustrates the assignment of weights to neighboring points.

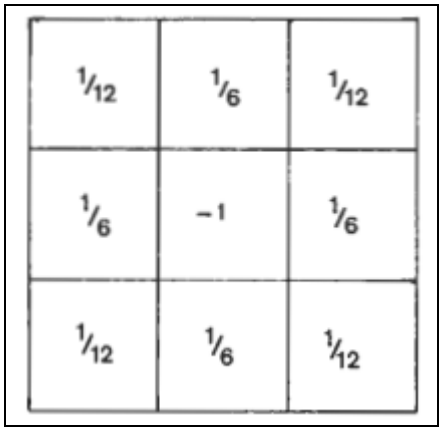

Figure 10 – Assignment of weights to neighbors

Now the cost function *e* can be minimized by differentiating it with respect to *u* and *v* and equating the derivatives to zero. After this minimization process, the results are given in (27) and (28).

$$(1 + \lambda E_x^2)u + \lambda E_x E_y v = \bar{u} - \lambda E_x E_t \tag{27}$$

$$\lambda E_x E_y u + (1 + \lambda E_y^2) v = \bar{v} - \lambda E_y E_t \quad \textbf{(28)}$$

Solving (27) and (28) by using one of the standard mathematical methods would be difficult, since the corresponding matrix would be quite large. Therefore, the velocity estimations can be computed by using estimated derivatives and the average of the previous velocity estimations. Eventually, the velocity vectors of the optical flow are obtained through an iterative approach by using (29) and (30).

$$u^{n+1} = \bar{u}^n - E_x \frac{(E_x \bar{u}^n + E_y \bar{v}^n + E_t)}{(1 + \lambda(E_x^2 + E_y^2))} \quad \textbf{(29)}$$

$$v^{n+1} = \bar{v}^n - E_y \frac{(E_x \bar{u}^n + E_y \bar{v}^n + E_t)}{(1 + \lambda(E_x^2 + E_y^2))} \quad \textbf{(30)}$$

3.2.2 Kanade Lucas Method

Kanade Lucas method is a two frame differential method for optical flow estimation. Although it has been almost 30 years, since the method was first proposed, it is still one of the most popular methods for calculating the optical flow. The additional constraint, which is needed for the estimation of the optical flow, is introduced by assuming that the flow in a local neighborhood around the central pixel is constant. To obtain this structure, Kanade Lucas method generally uses blocks for every pixel in the image. And it tries to minimize the optical flow equation within these blocks.

$$\min \sum_{(i,j) \in B} \left(E_x^{ij} u + E_y^{ij} v + E_t^{ij} \right)^2 \quad \textbf{(31)}$$

In order to make this minimization process one should take the derivatives with respect to *u* and *v*, and equate them to zero.

$$\sum_{(i,j)\in B}\left(E_x^{ij}u+E_y^{ij}v+E_t^{ij}\right)E_x^{ij}=0$$
$$\sum_{(i,j)\in B}\left(E_x^{ij}u+E_y^{ij}v+E_t^{ij}\right)E_y^{ij}=0 \quad \textbf{(32)}$$

After some mathematical manipulations the estimation for optical flow vector can be obtained as [26].

$$\begin{bmatrix}\hat{u}\\ \hat{y}\end{bmatrix}=\begin{bmatrix}\sum_{(i,j)\in B}E_x^{ij}E_x^{ij} & \sum_{(i,j)\in B}E_y^{ij}E_x^{ij}\\ \sum_{(i,j)\in B}E_x^{ij}E_y^{ij} & \sum_{(i,j)\in B}E_y^{ij}E_y^{ij}\end{bmatrix}^{-1}\begin{bmatrix}-\sum_{(i,j)\in B}E_t^{ij}E_x^{ij}\\ -\sum_{(i,j)\in B}E_t^{ij}E_y^{ij}\end{bmatrix} \quad \textbf{(33)}$$

The advantage of Kanade Lucas method is its robustness in presence of noise with compared to the point-wise methods. On the other hand, the flow information that Kanade Lucas method provides fades out quickly across motion boundaries. Additionally, it provides little or no flow information in the inner parts of uniform regions of the image, because it is an entirely local method.

3.3 An Improved Version of KLT

Kanade Lucas Tracker is a feature tracker algorithm, which is commonly used in the literature. In this thesis work, an improved version of KLT [27] will be used. In this section, first the details of the implementation are given and then the improvements of the algorithm are explained. Finally, the simulation results of the algorithm are presented.

3.3.1 Pyramidal Feature Tracking

This algorithm uses two grayscale images as inputs, namely *I* and *J*. These two images usually correspond to two consecutive frames for a video. Let $u = [u_x\ u_y]^T$ be an image point on the first image *I*. The aim of the feature tracking algorithm is to find the corresponding point, $v = u + d = [u_x+d_x\ u_y+d_y]^T$, on the second image *J*, which is similar to the point on *I*. The vector $d = [d_x\ d_y]^T$ is the optical flow vector. The optical flow vector can be defined as the vector that minimizes the cost function which is given below:

$$\varepsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x,y) - J(x+d_x, y+d_y))^2 \quad \textbf{(34)}$$

where w_x and w_y determines the size of the integration window, which is referred as block in Kanade Lucas method. For the first step of the algorithm, the pyramidal representation should be built for the two input images. Let L be the pyramid level and the 0^{th} level of the pyramid be equal to the original image itself. Then, the pyramid representation is built recursively as shown in Equation 35,

$$\begin{aligned} I^L(x,y) &= \frac{1}{4} I^{L-1}(2x,2y) + \frac{1}{8}(I^{L-1}(2x-1,2y) + I^{L-1}(2x+1,2y) \\ &+ I^{L-1}(2x,2y-1) + I^{L-1}(2x,2y+1)) + \frac{1}{16}(I^{L-1}(2x-1,2y-1) \\ &+ I^{L-1}(2x+1,2y+1) + I^{L-1}(2x-1,2y+1) + I^{L-1}(2x+1,2y+1)) \end{aligned} \quad \textbf{(35)}$$

Maximum value of L or the pyramid height is picked heuristically yet, 2, 3 or 4 are practical values. Since the image size will be too small, going beyond level 4 does not make much sense for typical image sizes.

In order to obtain pyramidal feature tracking, the optical flow is computed at the deepest pyramid level firstly and then the result is propagated to the upper level as an initial guess. The same process is repeated iteratively until the 0^{th} level is reached. Accordingly, the cost function defined in Equation 34 should be modified.

$$\varepsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-w_x}^{u_x^L+w_x} \sum_{y=u_y^L-w_y}^{u_y^L+w_y} (I^L(x,y) - J^L(x+g_x^L+d_x^L, y+g_y^L+d_y^L))^2 \quad \textbf{(36)}$$

Here, $g^L=[g_x{}^L\ g_y{}^L]^T$ is the initial guess for optical flow at level L and $d^L=[g_x{}^L\ d_y{}^L]^T$ is the residual optical flow vector at level L. While passing to upper level, L to L-1, the new initial guess would be

$$g^{L-1} = 2(g^L + d^L) \quad \textbf{(37)}$$

After reaching to the 0^{th} level, the final optical flow d can be calculated by using the following relation.

$$d = g^0 + d^0 \tag{38}$$

Using image pyramid structure helps each residual optical flow vector d^L to be kept small, due to the effect of each initial guess g^L. This provides the algorithm to handle large pixel motions in the scene, while keeping the size of integration window relatively small.

3.3.2 Iterative Kanade Lucas Optical Flow Calculation

At each pyramid level, the algorithm tries to calculate optical flow vector by using Kanade Lucas method iteratively. Here, the superscript L is not used for simplicity. For clarity purposes, new images A and B are defined as follows,

$$\begin{aligned} A(x,y) &= I^L(x,y) \\ B(x,y) &= J^L(x+g_x^L, y+g_y^L) \end{aligned} \tag{39}$$

where $A(x,y)$ is defined over a window size $(2w_x+3)\times(2w_y+3)$ instead of $(2w_x+1)\times(2w_y+1)$. This change will provide an advantage while calculating the spatial derivatives. The displacement vector and the image position vector are also renamed as $\bar{v}=[v_x\ v_y]^T=d^L$, and $p=[p_x\ p_y]^T=u^L$, respectively. With respect to this new notation, minimized cost function can be rewritten as

$$\varepsilon(v_x,v_y) = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} (A(x,y) - B(x+v_x, y+v_y))^2 \tag{40}$$

In order to find the displacement vector $\bar{v}=[v_x\ v_y]^T$ that minimizes the cost function, one should take the first derivative of cost function and equate it to zero as shown in Equation 41.

$$\frac{\partial \varepsilon(\bar{v})}{\partial \bar{v}} = -2 \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} (A(x,y) - B(x+v_x, y+v_y)) \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \quad \textbf{(41)}$$

First order Taylor series expansion can be applied to Equation 41.

$$\frac{\partial \varepsilon(\bar{v})}{\partial \bar{v}} \approx -2 \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} (A(x,y) - B(x,y) - \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \bar{v}) \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \quad \textbf{(42)}$$

Here, the difference between $A(x,y)$ and $B(x,y)$ is the temporal image derivative $\delta I(x,y)$ and the matrix $\begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}$ is the image gradient vector $\begin{bmatrix} I_x \\ I_y \end{bmatrix}$. The image derivatives can be calculated from $A(x,y)$ as given below

$$\begin{aligned} I_x(x,y) &= \frac{\partial A(x,y)}{\partial x} = \frac{A(x+1,y) - A(x-1,y)}{2} \\ I_y(x,y) &= \frac{\partial A(x,y)}{\partial y} = \frac{A(x,y+1) - A(x,y-1)}{2} \end{aligned} \quad \textbf{(43)}$$

After some mathematical manipulations [27], (42) can be rewritten in terms of image derivatives.

$$\frac{1}{2}\begin{bmatrix} \frac{\partial \varepsilon(\bar{v})}{\partial \bar{v}} \end{bmatrix}^T \approx \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \left(\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \bar{v} - \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix} \right) \quad \textbf{(44)}$$

For simplicity, the first term in the paranthesis given in (44) is symbolized by G, whereas the second term is symbolized by $\bar{b}$. Accordingly, (44) becomes,

$$\frac{1}{2}\begin{bmatrix} \frac{\partial \varepsilon(\bar{v})}{\partial \bar{v}} \end{bmatrix}^T = G\bar{v} - \bar{b} \quad \textbf{(45)}$$

The optimum optical flow vector can be calculated by using:

$$\bar{v}_{opt} = G^{-1}\bar{b} \quad \textbf{(46)}$$

Equation 46 is the main relation for the well-known Kanade Lucas optical flow equation. However, it is valid only if the motion or the pixel displacement in the

scene is relatively small. Otherwise, the first order Taylor series expansion, which is used in (42), will not hold. Therefore, one should follow an iterative way and calculate the optical flow vector multiple times to get a more accurate solution. For the iterative version of the algorithm, let k be the iterative index and let the previous iterations provide an initial guess as $\overline{v}^{k-1}=[v_x^{k-1}\ v_y^{k-1}]^T$. Then, the translated image considering $\overline{v}^{k-1}$ will be

$$B_k(x,y)=B\left(x+v_x^{k-1},y+v_y^{k-1}\right) \tag{47}$$

Similar to Equation 40, the aim is to find the residual pixel motion vector $\overline{\eta}^k=[\eta_x^k\ \eta_y^k]$ which minimizes the cost function given below

$$\varepsilon\left(\eta_x^k\eta_y^k\right)=\sum_{x=p_x-w_x}^{p_x+w_x}\sum_{y=p_y-w_y}^{p_y+w_y}\left(A(x,y)B_k\left(x+\eta_x^k,y+\eta_y^k\right)\right)^2 \tag{48}$$

For finding the solution, one should follow the same steps, which is given in Equations 41-46 and do the one step Kanade Lucas optical flow computation. Eventually, the residual pixel motion can be computed by using Equation 49.

$$\eta^{-k}=G^{-1}\overline{b_k} \tag{49}$$

where $\overline{b}_k$ is the image mismatch vector, which is defined as

$$\overline{b_k}=\sum_{x=p_x-w_x}^{p_x+w_x}\sum_{y=p_y-w_y}^{p_y+w_y}\left(\begin{bmatrix}\delta I_k(x,y)I_x(x,y)\\ \delta I_k(x,y)I_y(x,y)\end{bmatrix}\right) \tag{50}$$

In (50), the spatial derivatives are calculated only once by using (43). The matrix G also stays constant; however, k^{th} image difference δI_k should be calculated as follows:

$$\delta I_k(x,y)=A(x,y)-B_k(x,y) \tag{51}$$

By using residual pixel motion vector, which is calculated in (49), new pixel displacement guess for the next iteration step *k+1* will be calculated as follows,

$$\bar{v}^{k} = \bar{v}^{k-1} + \bar{\eta}^{k} \quad \textbf{(52)}$$

The number of iterations is a user selected parameter; however, practically in 5 iterations the convergence can be reached [27]. While choosing the number of iterations, one should always remember that there is a tradeoff between accuracy and computation time. Finally, the final solution for the optical flow vector is obtained as

$$\bar{v} = d^{L} = \bar{v}^{k} = \sum_{k=1}^{K} \bar{\eta}_{k} \quad \textbf{(53)}$$

where *K* is the total iteration number. It should be finally noted that this overall procedure should be repeated for all levels of image pyramids of the input images.

3.3.3 Summary of the Algorithm

In this part, the important steps of the algorithm are summarized to present the entire algorithm. There are two input images namely *I* and *J*, which correspond to two consecutive frames of a video. The main purpose is to find the corresponding point in the second image of a point in the first image. First of all, the pyramid representations for two input images are built up and pyramidal guesses are initialized. The algorithm has two for loops; the outer for loop is for pyramid levels and inner for loop is for the calculation of the optical flow vectors iteratively. The algorithm starts with the top of the pyramid or smallest image to processing, in which motion will be expressed with less number of pixels. It is also meaningful for not violating Taylor series expansion. Then, the horizontal and vertical derivatives are calculated with the help of (43). After that by using these derivatives, the spatial gradient matrix *G* can be constituted.

In the inner for loop, the optical flow vectors are calculated iteratively. To do this, firstly, the difference between the 1st and 2nd image are taken; however, pyramidal

guess (g_x^L and g_y^L) and optical flow guess (v_x^{k-1} and v_y^{k-1}) are also taken into consideration by using (39) (2nd condition in (39)), (47) and (51). Therefore, these calculations are made in sub pixel accuracy. Then, by using image differences and horizontal and vertical derivatives, the image mismatch vector $\overline{b}_k$ can be constituted. Finally, by the help of (49), the optical flow vector, which will be used as an estimate for the next iteration can be calculated. After the overall procedure is repeated for all the levels of the pyramid, a final optical flow vector for corresponding location can be obtained. Surely, the output of this algorithm is only for a single pixel. If one wants the results for all the pixels, then the algorithm should be repeated for each pixel.

In order to provide complete understanding, the main algorithm of KLT is presented in the Appendix.

3.3.4 Improvements of the Algorithm

In this part, the advantages of improved version KLT algorithm are explained. First, using the image pyramid approach provides the algorithm not to violate Taylor series expansion assumption, because for higher levels of image pyramid the probability of having a relatively rapid motion in the scene becomes smaller and smaller. By using image pyramid one can also handle more motion, e.g. relatively fast objects in the scene. That is because while going from top to bottom of the pyramid the result of the upper level is always given to the bottom level as an initial estimate. Besides, the optical flow vectors are calculated in an iterative way. In other words, one can get better results at every step, but the tradeoff is computation time. In addition, these calculations are made in sub pixel accuracy; therefore this makes the algorithm more robust.

3.3.5 Simulation Results for Improved Version of KLT Algorithm

In this section, simulation results of the improved version of KLT algorithm are presented. For the implementation of improved version of KLT algorithm, the number of pyramid levels is selected as 3 and the window size is selected as 5 x 5. First, the algorithm is tested against a moving car scene. The car is moving in

horizontal direction and there is also a camera pan. In Figure 11, the input frames for moving car scene are shown. The picture sizes of these frames are 320×240.

Figure 11 – A moving car scene

For all pixels, the optical flow vectors are calculated and illustrated in Figure 12 as a *needlegram*. To observe the vectors clearly, it is also zoomed into the moving car region. As one can easily observe from Figure 13, the optical flow vectors show mainly the correct direction for the zoomed region. Moreover, to show the outputs 'quiver' function of MATLAB is used. The 'quiver' function makes some normalization because of this reason if all the background is moving; 'quiver' is not an optimal way to visualize the results. Beacuse in that case the movements of the objects are not emphasized. But if not, it is suitable to visualize the results.

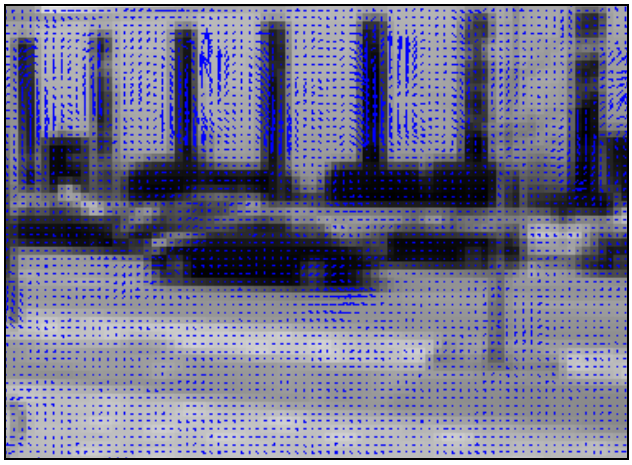

Figure 12 – Calculated optical flow vectors

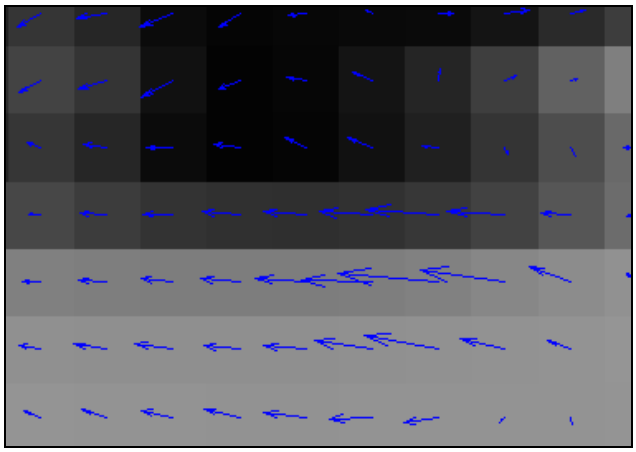

Figure 13 – Zoomed into the moving car region

Second, the algorithm is tested against a crowded airport scene. As one can easily see from Figure 14, the blonde lady in the foreground and the lady in the background are moving significantly. In this video clip, sizes of these frames are 720×576.

Figure 14 – A crowded airport scene

Similarly, for all the pixels, the optical flow vectors are calculated and it is zoomed near the head of the blonde lady to see the vectors clearly. The results are shown in Figure 15 and Figure 16.

Figure 15 – Calculated optical flow vectors

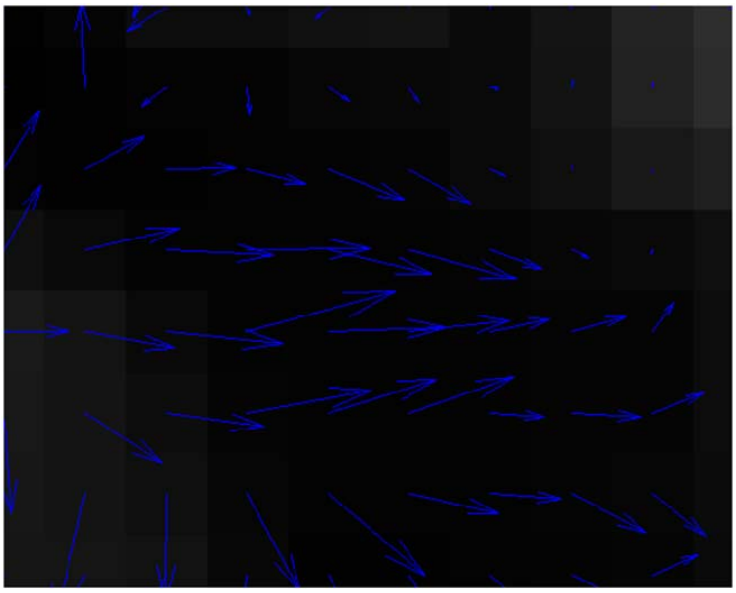

Figure 16 – Zoomed into near the head of blonde lady

Similarly, the results of the improved version of KLT algorithm are satisfying for zoomed region. However, if the optical flow vectors for each pixel in the scene are investigated, it can be observed that the results are noisy and there are also some false alarms. In other words, the performance of improved version of KLT algorithm is not stable for the overall scene. Because of this reason, improved

version of KLT algorithm can not be used by itself for tracking purposes; it needs a tracker algorithm to perform reliably.

3.4 Finding the Features

In some cases, input data may be quite large to be processed and it may have redundant data in it. In these circumstances, the input data should be transformed into a set of features and expressed with these features. This transformation process is called feature extraction. In other words, feature extraction is a special form of dimensionality reduction.

For a visual tracking system, calculating the optical flow vector for each pixel in every frame brings about a huge computation load. Moreover, this computation load causes to slow down the speed of the tracking system. In order to avoid this, the feature extraction algorithm is implemented only to moving regions in a frame, which is the output of previous chapter. Additionally, the optical flow vector is calculated only for the feature points. As a result, one gets the feature points in moving regions with their optical flow vectors in the end.

For feature extraction step, an algorithm called ‘Shi-Tomasi corner detection’ [28] is used. The algorithm proposes a feature selection criterion. The process of discovering regularities, which can be made easier and less time consuming by removing features of the data that are irrelevant or redundant, is named feature selection [29]. In the next part feature extraction algorithm ‘Shi-Tomasi corner detection’ is explained in full detail.

3.4.1 Shi-Tomasi Corner Detection

Theoretically, having more features provides one more discriminating power. However, in practice, this is not always the case. The key point in machine learning algorithms is that finding the good and sufficient number of features. Due to many and useless features cause slow down the algorithm and may even mislead it. Selecting the right features for tracking and selecting the features, which correspond to reasonable physical points in the real world is a difficult job to do. Feature

selection methods are based on some measure of texturedness or cornerness. Corners, standard deviation in spatial intensity and zero crossings of the Laplacian of image intensity are commonly used in the literature. These interest operators are usually based on a preconceived idea and the resulting features are not guaranteed to be best for tracking algorithm to produce good results [28].

This algorithm proposes a principled feature selection criterion rather than traditional interest or cornerness measures. Specifically, it chooses the features, which make the tracker work best. In this formulation, *G* matrix from previous topic will be used. As a reminder *G* matrix is as given below.

$$G = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \left(\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \quad \textbf{(54)}$$

If the eigenvalues of *G* are analyzed, two relatively small eigenvalues correspond to a constant intensity profile within a window. A relatively large and a relatively small eigenvalue represent a texture pattern. Two relatively large eigenvalues clues corners or salt and paper textures within a window. The intensity variation within a window is limited by maximum pixel value, accordingly the larger eigenvalue cannot be arbitrary large.

In the light of the foregoing, the feature selection criterion of this algorithm will be min $(\lambda_1,\lambda_2)>\lambda$, where λ_1,λ_2 are the eigenvalues of *G* and λ is a predefined threshold. Therefore, if the minimum eigenvalue of *G* matrix is above a threshold then the window is accepted.

3.5 Simulation Results

As mentioned before, the feature extraction process will be applied only the moving regions in the scene, which is the output of previous chapter. So the two chapters are connected with each other. In this section, the simulation results of this combination are presented. The same test video, which is used in previous chapter, is selected here for comparison of the results.

The test video, which is illustrated in Figure 1, has 360×240 dimensions, 15 fps frame rate, 33 seconds duration and an average of 12 pixels nominal camera motion. In the first 250 frames, there are no foreground objects in the scene and first 200 frames are used in the training part of the background modeling. Therefore, the simulation here starts from 201th frame. Before the simulation, the expectation is to obtain the features, which are only from moving regions in the scene. In other words, the features should be extracted from white regions in Figure 4. The simulation results are given in Figure 17.

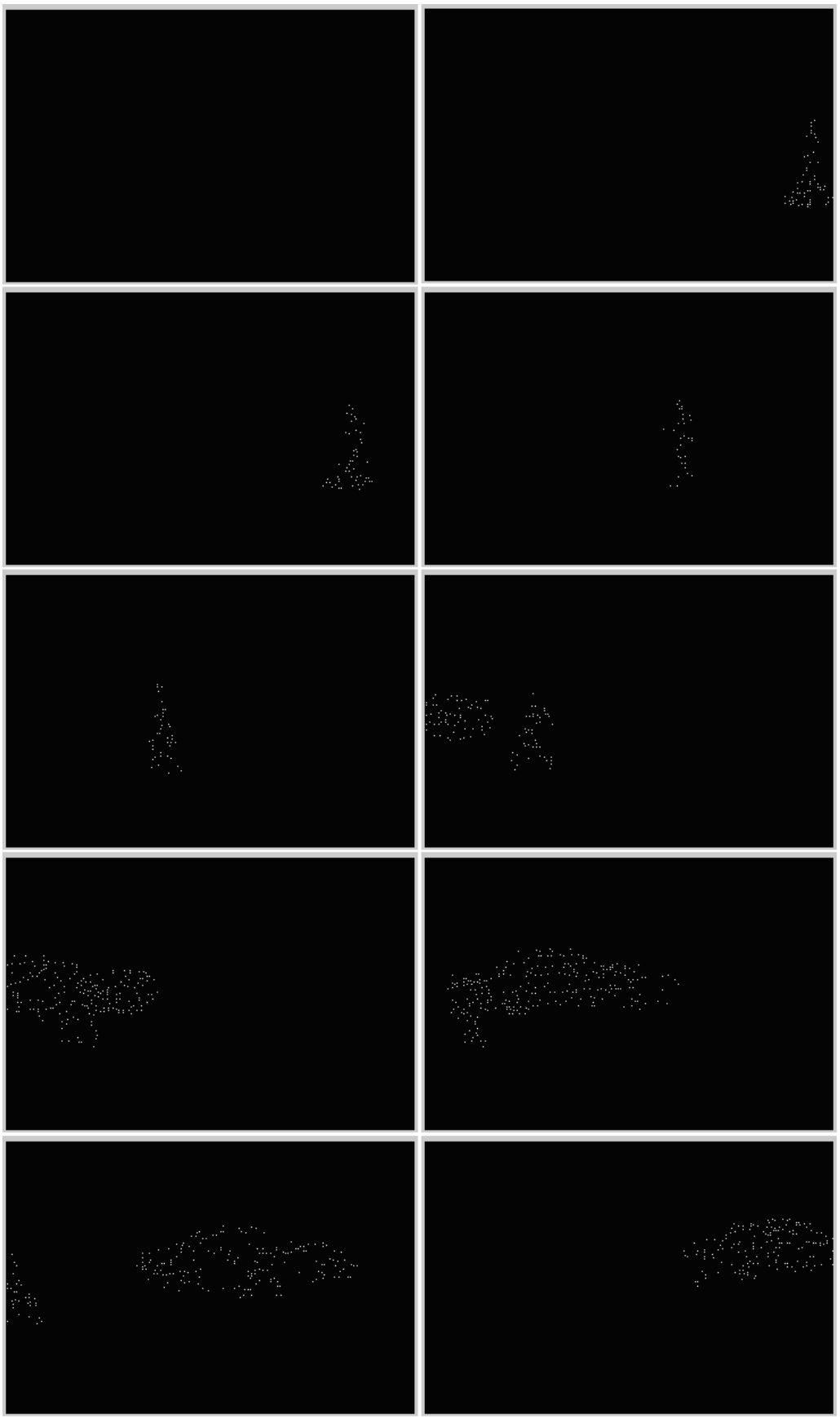

Figure 17 – Simulation results for feature extraction

It is clear that the simulation results align well with the expectation. The feature points are extracted only from moving regions, plus they are quite satisfying. A silhouette of a walking man and a moving car can be easily seen. In addition to this, the optical flow information of these feature points is also available. This optical flow information of these feature points will be used in the next chapter.

CHAPTER 4

TRACKING USING PARTICLE FILTER

Since they are introduced into the literature, particle filters have become very popular as a solution of tracking problem. It owes its popularity to its treatment to the tracking problem with a different approach. The methods like Kalman filters deal with a simplified version of the actual complex model. They can achieve an exact solution by using the simplified model. Kalman filters achieve optimal results for linear systems. Therefore, finding an exact solution for the simplified model can be advantageous for some systems, which are linear or almost linear. However, a simplified model may also be inadequate for some real-life cases. If the non-linearity of the system is high, simplified models do not reflect the actual conditions of the systems. To obtain exact analytic solution for non-linear and non-Gaussian cases, the particle filter offers an approximate solution by using actual complex model instead of an exact solution by using simplified model. Therefore, the selection of the filter is directly related with the system. On the other hand, the assumptions of the methods like Kalman filters are generally strong. Particle filter does not rely on any local linearization technique or any functional approximation. The trade-off for this flexibility is increase in the computational cost.

The particle filters are widely used in visual and radar tracking, navigation, communication, econometrics and image restoration. In this thesis work, a synthesis of particle filter and the all visual data from previous chapters will be used for visual tracking. In this chapter the particle filter is treated exhaustively. Firstly, some basic concepts related with particle filter are provided. Afterwards, the particle filter is given in full details and finally the implementation details and results are explained.

4.1 Particle Filter

4.1.1 Basic Concepts

Estimation problems can be categorized into three main groups, namely filtering, smoothing and prediction. Filtering involves the derivation of information about the quantity of interest at time *t* by using all the data available up to and including *t*. In smoothing, which is an a posteriori form of estimation, some past value of the quantity of interest is estimated with the data available up to and including *t*. The aim of prediction is to extract some information about the future value of the quantity of interest. Prediction is an a priori form of estimation.

A dynamic system can be analyzed with the help of at least two important models. Without these models a dynamic system would be meaningless and it would be very hard to make inference about the system. These models are the system model, which describes the evolution of the state with time, and the measurement model, which relates the noisy measurements to the state [30]. Real world systems commonly require estimation, when a measurement is received. Using a recursive filter would be the appropriate solution for this kind of problems. For recursive filtering approach there is no need to store complete data, because the received data will be processed sequentially. There is no reprocessing step for the existing data, if a new measurement becomes available. Beginning with Kalman filter, all the recursive filters rely on two stages: prediction and update. In the prediction stage, the state PDF is predicted by the help of the system model from one measurement time to the next. Usually prediction makes a translation and deformation to the state PDF because of the random noise. In the update stage, the latest measurement is used to modify the prediction PDF.

Prediction and update stages arise from Bayes rule, which denotes the filtering distribution by using the likelihood and the predictive density. The first term in (55) is the filtering distribution, the next term is the likelihood and the third term is the predictive density.

$$p(x_{t+1} \mid y_{1:t+1}) \propto p(y_{t+1} \mid x_{t+1}) p(x_{t+1} \mid y_{1:t}) \quad \textbf{(55)}$$

In many cases the likelihood function is usually known, however the predictive distribution of the state is an integral, which depends on the filtering density of previous period. The predictive distribution is stated in (56).

$$p(x_{t+1} \mid y_{1:t}) = \int p(x_{t+1} \mid x_t) p(x_t \mid y_{1:t}) dx_t \quad \textbf{(56)}$$

In practice, computing these densities is quite difficult. Generally, the filtering density of previous period is a complicated function of potentially high dimensional vector y_t and that prevents apply general Bayesian updating methods. Analytical solutions can be achieved for a few cases e.g. linear and Gaussian models. For all other cases, simulation based methods are frequently preferred.

Monte Carlo methods are stochastic sampling approaches, which are based on the use of random numbers and probability statistics. These methods are widely used in computer simulations of physical and mathematical systems. Monte Carlo methods aim to tackle the complex systems, which are analytically intractable, and allow one to examine with them. In mathematics, the Monte Carlo methods are ordinarily used to evaluate complicated integrals like the integral, which is given in (56). This process is called Monte Carlo integration.

Importance sampling is a general Monte Carlo integration method. It approximates the filtering density of previous period, thus Equation 56 becomes a sum over the individual particles instead of an integral.

$$p(x_t \mid y_{1:t}) = \sum_{i=1}^{N} w_t^i \delta_{x_t^i} \quad \textbf{(57)}$$

Importance sampling approach forms the basis of particle filter that is expressed in Section 4.1.2 in more detail.

4.1.2 Sequential Importance Sampling

The sequential importance sampling (SIS) algorithm is a Monte Carlo method, which lay a groundwork for most sequential Monte Carlo filters. The sequential Monte Carlo approach is based on implementation of a recursive Bayesian filter by Monte Carlo simulations. This approach is also known as condensation algorithm, bootstrap filtering, interacting particle approximations, survival of the fittest and particle filtering. The basic idea of particle filter is to represent posterior density by using a number of independent random variables called particles, which are sampled directly from state space, and their associated weights. The posterior density is then updated by involving the new observations. Thereby, particle filter solves the integral computation problem of Equation 56 by a discrete approximation to the filtering density. This approximation becomes an equivalent representation of the posterior PDF and the SIS filter approaches the optimal Bayesian estimator as the number of particles becomes very large. That obtains flexibility for non-Gaussian, multi-modal PDFs, because any arbitrary distribution can be easily represented with this approach.

For the details of SIS algorithm, let X_k be the sequence of all states up to time k, $p(X_k|Z_k)$ be the joint posterior density at time k and $p(x_k|Z_k)$ be its marginal density. The weights are also normalized for the sum of them to equal 1. The joint posterior density at k can be approximated by using (58) [30]:

$$p(X_k \mid Z_k) \approx \sum_{i=1}^{N} w_k^i \delta(X_k - X_k^i) \quad \textbf{(58)}$$

The discrete approximation is good enough if the samples are drawn from the posterior PDF. However, in practice one can not use the posterior PDF for sampling, because there is no explicit representation of posterior PDF. The solution of this problem is choosing the weights using the principle of importance sampling. This principle relies on sampling from another importance density, called prior belief $q(.)$ that the samples can be easily generated. In that case, the approximation in (58) is still correct up to a normalization constant, unless the particles are weighted according to (59).

$$w_k^i \alpha \frac{p(X_k^i \mid Z_k)}{q(X_k^i \mid Z_k)} \quad \textbf{(59)}$$

To derive the weight update equation, it is assumed that an approximation to $p(X_{k-1}|Z_{k-1})$ is formed with the samples at time step k-1. In order to approximate $p(X_k|Z_k)$ with the new samples at time k, the importance density will be chosen according to (60).

$$q(X_k \mid Z_k) = q(x_k \mid X_{k-1}, Z_k) q(X_{k-1} \mid Z_{k-1}) \quad \textbf{(60)}$$

By using Bayes rule, $p(X_k|Z_k)$ can be also expressed as given below.

$$p(X_k \mid Z_k) = \frac{p(z_k \mid X_k, Z_{k-1}) p(X_k \mid Z_{k-1})}{p(z_k \mid Z_{k-1})} = \frac{p(z_k \mid x_k) p(x_k \mid x_{k-1})}{p(z_k \mid Z_{k-1})} p(X_{k-1} \mid Z_{k-1}) \quad \textbf{(61)}$$

One can easily notice that, the denominator of Equation 61 does not depend on x. For this reason the term in the denominator can be thought as a normalizing variable such that the probability sums up to 1.

$$p(X_k \mid Z_k) \propto p(z_k \mid x_k) p(x_k \mid x_{k-1}) p(X_{k-1} \mid Z_{k-1}) \quad \textbf{(62)}$$

Consequently, the weight update equation can be achieved by substituting the (60), (61) and (62) into the Equation 59.

$$w_k^i = w_{k-1}^i \frac{p(z_k \mid x_k^i) p(x_k^i \mid x_{k-1}^i)}{q(x_k^i \mid X_{k-1}^i, Z_k)} \quad \textbf{(63)}$$

If only the filtered estimate of posterior $p(x_k|Z_k)$ is required at each time step, one can easily assume that the importance density is independent from the path X^i_{k-1} and the history of observations Z_{k-1}. In that case, there is no need to store all the history of observation, storing x_k^i is enough. The final weight update equation and the modified approximation about the posterior filtered density are stated successively in (64) and (65).

$$w_k^i \propto w_{k-1}^i \frac{p(z_k \mid x_k^i) p(x_k^i \mid x_{k-1}^i)}{q(x_k^i \mid x_{k-1}^i, z_k)} \quad \textbf{(64)}$$

$$p(x_k \mid Z_k) \approx \sum_{i=1}^{N} w_k^i \delta(x_k - x_k^i) \quad \textbf{(65)}$$

The performance of SIS algorithm depends on the selection of the importance density and the accuracy of the importance sampling approximation [31]. SIS algorithm is quite important and most of the particle filter algorithms are based on it. Pseudo-code of SIS algorithm is given in Table 3 [32].

Table 3 – Pseudo-code of SIS algorithm

$\left(\{x_k^i, w_k^i\}_{i=1}^N\right) = SIS\left(\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^N, z_k\right)$

- FOR $i = 1 : N$
  - Draw $x_k^i \sim q(x_k \mid x_{k-1}^i, z_k)$
  - Update the importance weight according to Equation 64
- END FOR

Calculate total weight: $t = SUM\left(\left\{\tilde{w}_k^i\right\}_{i=1}^N\right)$

- FOR $i = 1 : N$
  - Normalize $w_k^i = t^{-1}\tilde{w}_k^i$
- END FOR

4.1.3 Degeneracy Problem

Using another importance density instead of the posterior distribution may cause an increase of the unconditional variance of importance weights over time. Importance weights with large variances bring about inaccurate estimates. This problem is called weight degeneracy problem. In practice, after a few iterations, most particles have negligible weights; the weights are concentrated on a few particles only. This is disadvantageous since a lot of computational effort is wasted to updating those trivial particles, whose contribution to the approximation of posterior density is

almost zero. To calculate the degree of degeneracy of an algorithm, effective sample size can be introduced.

$$N_{eff} = \frac{N}{1 + Var\left(w_k^{*i}\right)} \tag{66}$$

$$w_k^{*i} = \frac{p\left(x_k^i \mid Z_k\right)}{q\left(x_k^i \mid x_{k-1}^i, z_k\right)} \tag{67}$$

However effective sample size cannot be evaluated exactly, because of true weight, which is given in (67). For this reason, an estimate of effective sample size can be used as a measure of degeneracy.

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N} (w_k^i)^2} \tag{68}$$

Here w^i_k is the normalized weight, which can be obtained by using (63). For two extreme cases, if the weights are uniform N_{eff} will reach its maximum value, N, and if all the weights equal to zero except only one weight, then N_{eff} will take its minimum value which is equal to 1. As a result, one can easily claim that:

$$1 \leq N_{eff} \leq N \tag{69}$$

Having a small N_{eff} indicates that, the algorithm has a severe degeneracy level. Clearly, it is an undesirable effect for particle filters. In order to cope with this situation, one can use too many samples; therefore the effect of degeneracy will be reduced naturally. However, this brute force approach is impractical. Instead of this, resampling can also fix the degeneracy problem more practically.

4.1.4 Resampling

The basic idea of resampling is to eliminate particles that have small weights and to concentrate on particles with large weights [30]. In other words, the total number of

particles is kept the same, while increasing the number of particles in high probability regions and decreasing the number of particles in low probability regions. Resampling process involves a mapping from $\{x^i_k, w^i_k\}$ into $\{x_k^{i^*}, 1/N\}$ with uniform weights. Then, the approximation about the posterior filtered density is made with this new set of random samples, which is generated by resampling.

SIS algorithm needs a resampling process, whenever a significant degeneracy is observed. This decision can be given by threshold the effective sample size. SIS algorithm with resampling approach constitutes the generic particle filter. The working principle of generic particle filter is illustrated in Figure 18 [32].

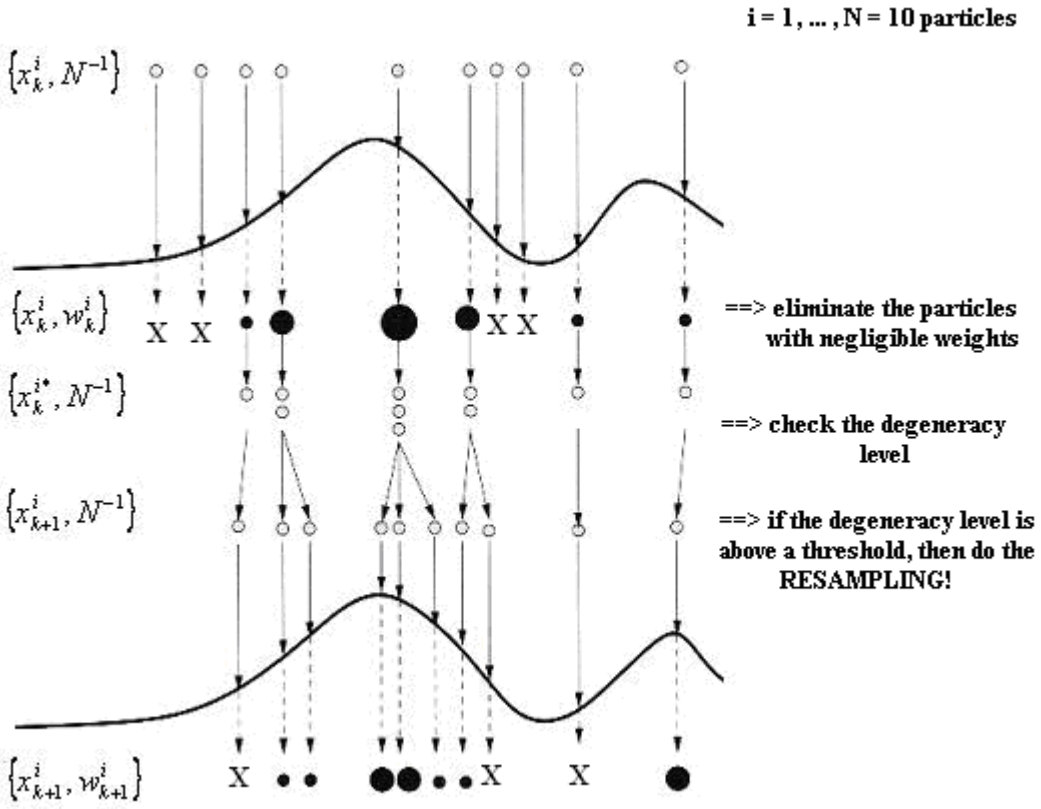

Figure 18 – The working principle of generic particle filter

4.1.5 Sample Impoverishment Problem

Resampling process reduces the negative effects of degeneracy; however it brings another problem. If a vast of majority of the weights is placed on a few particles,

these particles with high weights are statistically sampled many times. This causes a loss of diversity among the particles, which is also known as sample impoverishment problem. Sample impoverishment problem simply arises from resampling process, where the samples are drawn from a discrete distribution instead of a continuous distribution.

If the process noise of the system is very small, then the negative effects of sample impoverishment are perceived easily. Because having a very small process noise leads all particles to collapse to a single point after a few iterations. Introducing an additional noise to the samples, which is called jittering or roughening, can reduce the effect of sample impoverishment. However, in general, when process noise in the state dynamics is zero, using a particle filter is not entirely appropriate [32].

To deal with sample impoverishment problem, there are two commonly used techniques namely; resample-move algorithm and regularization. These techniques are mentioned in forthcoming parts. In Section 4.1.6, the importance of selection of importance density is explained. Selecting an appropriate importance density will also protect the particle filter algorithm from negative effects of sample impoverishment.

4.1.6 Selection of Importance Density

Selection of the importance density directly affects the performance of a particle filter. In order to find the optimal importance density function, one should choose the function, which maximizes the variance of importance weights.

$$q(x_k \mid x_{k-1}^i, z_k)_{opt} = p(x_k \mid x_{k-1}^i, z_k) = \frac{p(z_k \mid x_k, x_{k-1}^i)\, p(x_k, x_{k-1}^i)}{p(z_k \mid x_{k-1}^i)} \quad \textbf{(70)}$$

By substituting Equation 70 into (64), one can find the weight update equation for optimal importance density case.

$$w_k^i = w_{k-1}^i \int p(z_k \mid x_k)\, p(x_k \mid x_{k-1}^i) dx_k \quad \textbf{(71)}$$

This choice of importance density would be optimal since the conditional variance of the true weight would be zero in this case. Equation 71 also indicates that the importance weights at time k should be computed before the particles are propagated to time k. However, evaluating an integral over the new state can rarely be done. There are two cases that one can use the optimal importance density. If x_k is a member of a finite set, then the integral in (71) becomes a sum. In this case, sampling from $p(x_k|x_{k\text{-}1}^{\text{i}}, z_k)$ would be possible. The second case is a system, where the state dynamics are nonlinear and the measurements are linear and the noises are additive Gaussian. Such a system can be summarized as;

$$x_k = f_{k-1}(x_{k-1}) + v_{k-1} \quad \textbf{(72)}$$

$$z_k = H_k x_k + w_k \quad \textbf{(73)}$$

where the noises are mutually independent zero-mean white Gaussian sequences with the covariances $Q_{k\text{-}1}$ and R_k. For such system, both the optimal importance density and $p(z_k|x_{k\text{-}1})$ would be Gaussian [32].

$$p(x_k \mid x_{k-1}, z_k) = N(x_k; a_k, \Sigma_k) \quad \textbf{(74)}$$

$$p(z_k \mid x_{k-1}) = N(z_k; b_k, S_k) \quad \textbf{(75)}$$

$$a_k = f_{k-1}(x_{k-1}) + \Sigma_k H_k^T R_k^{-1}(z_k - b_k) \quad \textbf{(76)}$$

$$\Sigma_k = Q_{k-1} - Q_{k-1} H_k^T S_k^{-1} H_k Q_{k-1} \quad \textbf{(77)}$$

$$S_k = H_k Q_{k-1} H_k^T + R_k \quad \textbf{(78)}$$

$$b_k = H_k f_{k-1}(x_{k-1}) \quad \textbf{(79)}$$

For other cases, one must tend to a suboptimal choice of importance density. Local linearization techniques can be used to obtain a suboptimal approximation to the optimal importance density. These techniques are generally based on a Gaussian approximation to $p(x_k|x^i_{k\text{-}1}, z_k)$.

An alternative convenient suboptimal choice is to select the importance density as transitional prior. This choice is commonly preferred since it is intuitive and easy to implement. However, it does not take measurements into account.

$$q(x_k \mid x^i_{k-1}, z_k) = p(x_k \mid x^i_{k-1}) \quad \textbf{(80)}$$

$$w^i_k \propto w^i_{k-1} p(z_k \mid x^i_k) \quad \textbf{(81)}$$

The selection of importance density is a critical design step for particle filters, which should be emphasized by the users.

4.1.7 Sampling Importance Resampling Filter

The sampling importance resampling (SIR) filter, which is also known as bootstrap filter, is another Monte Carlo method that is commonly used for recursive Bayesian filtering problems. SIR filter has mild requirements that the likelihood function is available and the states can be simulated. It also assumes that state dynamics and measurement functions are given and sampling from process noise can be done. The difference between SIS and SIR filters is that in SIR filter; resampling step is performed at every time index; whereas in SIS filter, resampling is taken whenever needed. Because of this reason, SIS filter is less computationally expensive.

Like SIS filter, the choice of importance density plays a crucial role in the performance of SIR filter. According to Equation 80, samples from $p(x_k|x^i_{k\text{-}1})$ are needed for the selection of importance density. To meet this requirement, a process noise sample,$v^i_{k\text{-}1}$, is generated by using the process noise PDF and set:

$$x_k^i = f_{k-1}(x_{k-1}^i, v_{k-1}^i) \quad \textbf{(82)}$$

In this case the weights will be as given in (80), however because of the resampling process at every time index the equation will simplify to:

$$w_k^i \propto p(z_k \mid x_k^i) \quad \textbf{(83)}$$

SIR filter can be inefficient and sensitive to outliers, since the importance density of SIR filter is independent of measurements. Another disadvantage of SIR filter is that it can be exposed to sample impoverishment easily, because it has a resampling step every time index. However, SIR algorithm is easy to implement, like SIS algorithm. A pseudo-code of SIR algorithm is given in Table 4.

Table 4 – Pseudo-code of SIR algorithm

$\left(\{x_k^i\}_{i=1}^N\right) = SIR\left(\{x_{k-1}^i\}_{i=1}^N, z_k\right)$

- FOR $i = 1 : N$
  - Draw $x_k^i \sim p(x_k \mid x_{k-1}^i)$
  - Calculate $\tilde{w}_k^i = p(z_k \mid x_k^i)$
- END FOR

Calculate total weight: $t = SUM\left(\left\{\tilde{w}_k^i\right\}_{i=1}^N\right)$

- FOR $i = 1 : N$
  - Normalize $w_k^i = t^{-1}\tilde{w}_k^i$
- END FOR

$\left(\{x_k^i\}_{i=1}^N\right) = RESAMPLE\left(\{x_k^i, w_k^i\}_{i=1}^N\right)$

4.1.8 Other Related Particle Filters

In the literature, there are many different particle filter and most of them are based on SIS algorithm. The differences between these algorithms are generally the choice of the importance sampling density and the modification of the resampling step. Because of this reason, the remaining algorithms are explained in general terms.

The auxiliary SIR (ASIR) filter is proposed to correct some imperfections of SIR filter. It performs a resampling process at time *k*-1, before the particles are propagated at time *k* and new samples points will be closer to true state because of the use of current measurement. The weight update equation of ASIR filter is given in (84).

$$w_k^j = \frac{p(z_k \mid x_k^j)}{p(z_k \mid \mu_k^{i^j})} \tag{84}$$

Here, i^j corresponds to the index of the particle at time *k*-1. ASIR filter is a more robust algorithm in the case of small process noise; however its performance degrades in the case of high process noise.

The regularized particle filter (RPF) uses a continuous approximation of the posterior density at resampling step instead of a discrete approximation. The remaining algorithm is identical to SIR algorithm. RPF draws samples from the approximation given below.

$$p(x_k \mid Z_k) \approx \sum_{i=1}^{N} w_k^i K_h(x_k - x_k^i) \tag{85}$$

$$K_h(x) = \frac{1}{h^{n_x}} K\left(\frac{x}{h}\right) \tag{86}$$

Here, K_h is rescaled kernel density and n_x is the dimension of the state vector. Using kernel approximation is reasonable when the dimension of the state is low. In practice, RPF performs better than SIR filter especially when the sample impoverishment is severe.

4.1.9 The Problems of Particle Filters

Except degeneracy and sample impoverishment, particle filters have also some other problems. The optimal number of particles, which is needed for particle

filters, is always unknown. Using too few particles may cause that the improbable states have very few or no particles. This leads a lag time between the occurrence of the event and the response of the states as becoming more likely. Additionally, using few particles increases the variance, thus the estimations will be inaccurate.

Therefore the number of particles should be increased in order to make sure that all states are represented well. However, in that case the algorithm needs too many particles to represent very small probabilities. Therefore, each state gets more particles and computational cost increases enormously. In addition, lots of computation power is wasted on the states, which already have many particles. That will obviously affects the performance of particle filter negatively. Additionally, the performance of particle filters also degrades quickly as the state dimension increases [33].

To determine the sufficient number of samples, trial and error method should be followed; the number of particles is increased until the observed error falls to a steady level [34]. The system model and the noise are also related to the number of particles. If the nonlinearity in system model or the non-Gaussian noises increase, then potentially more particles are needed. In conclusion, the number of particles in a particle filter can be regarded as a tuning parameter for users.

4.2 The Implementation of Particle Filter

For this thesis work, a generic particle filter structure is built according to the information given throughout Chapter 4. In order to check, whether the algorithm works well enough or not, a simple linear dynamic system is solved by Kalman filter and generic particle filter. If the results are close enough, one can say that the particle filter algorithm works well, because for linear dynamic systems Kalman filter gives the optimal results.

The dynamic system, that is chosen to use here, has a constant velocity. The state vector indicates the position and the velocity on x and y directions, therefore it is four dimensional. The measurements are two dimensional; only the positions on x

and y directions. The state and measurement equations of such a system are given in (87) and (88).

$$x_k = Fx_{k-1} + w_k \tag{87}$$

$$y_k = Hx_k + v_k \tag{88}$$

Here, w_k is process noise which is normally distributed, with zero mean and Q covariance. Similarly, measurement noise v_k is also normally distributed, with zero mean and R covariance. These terms in the system equations are clarified below:

$$F = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} T^2/2 & 0 \\ T & 0 \\ 0 & T^2/2 \\ 0 & T \end{bmatrix} \tag{89}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.1406 & 0.5625 & 0 & 0 \\ 0.5625 & 2.25 & 0 & 0 \\ 0 & 0 & 0.1406 & 0.5625 \\ 0 & 0 & 0.5625 & 2.25 \end{bmatrix} \tag{90}$$

$$R = \begin{bmatrix} 3.1623 & 0 \\ 0 & 3.1623 \end{bmatrix} \tag{91}$$

T is the sampling time, which is chosen as 0.5 for this implementation. The next step is generating the measurements until k = 100 by using (87) and (88).

Afterwards, by applying standard Kalman filter equations one can easily get the Kalman filter solution to this problem.

In order to reach to particle filter solution, first the weights are initialized as a uniform distribution. Then the standard SIS algorithm, which is given in Table 3, is applied. If the effective sample size, which is expressed in (68), falls below the half of the number of particles, then resampling process is done. The results for different number of particles are stated in Figure 19-24.

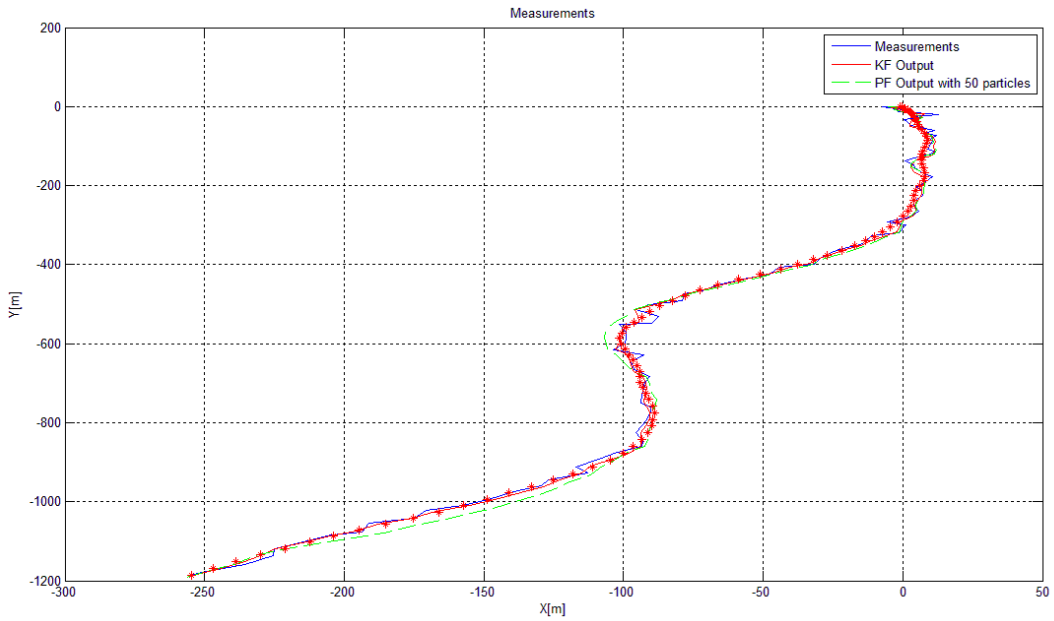

Figure 19 – The comparison between Kalman filter and particle filter with 50 particles

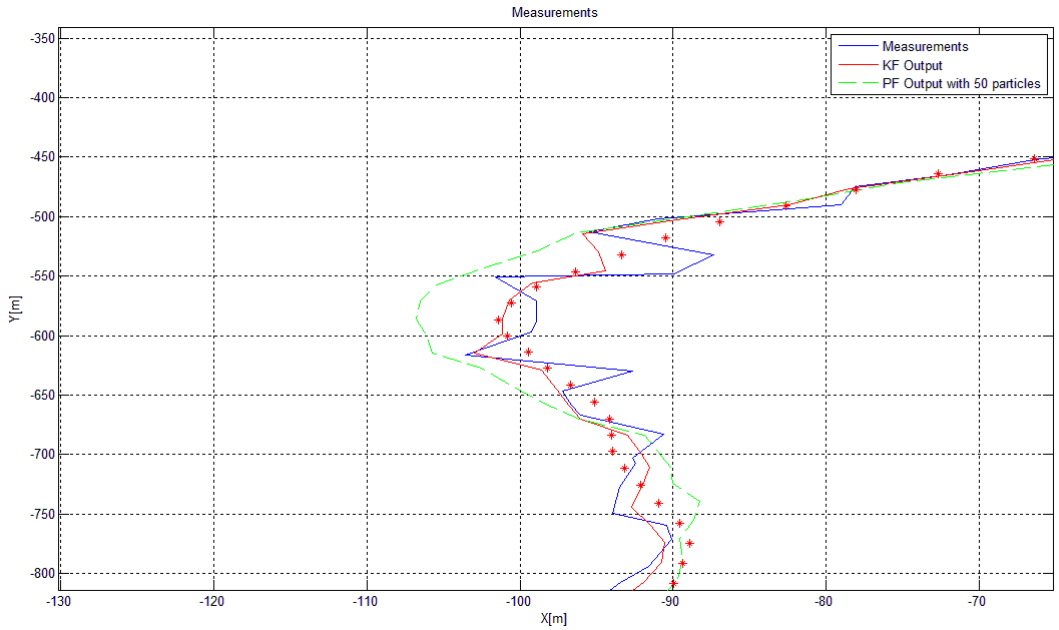

Figure 20 – Zoomed into the blob in the middle of the sequence

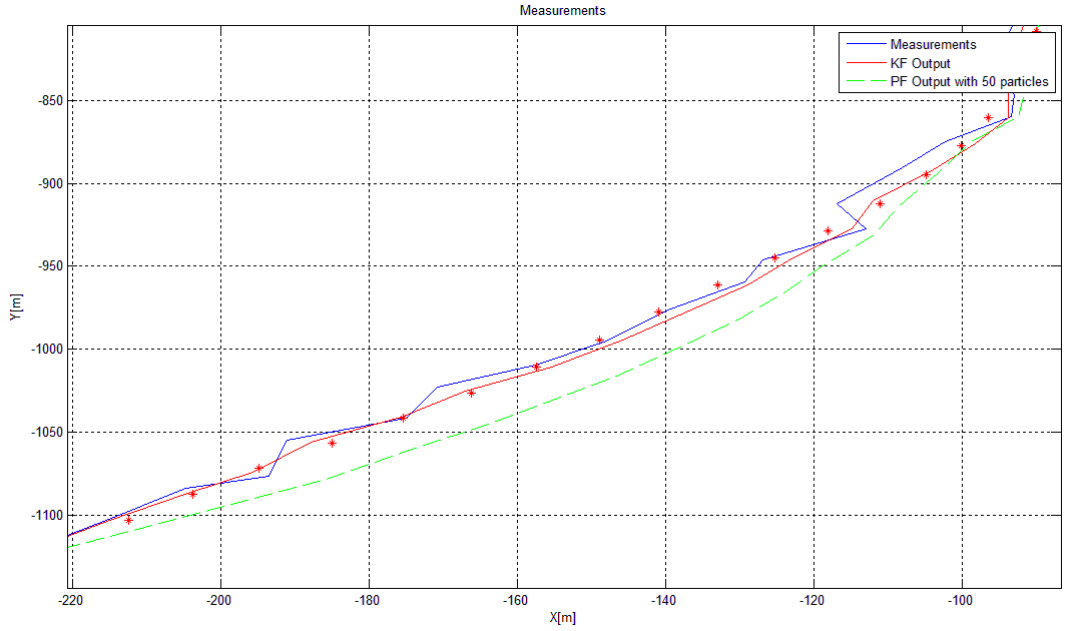

Figure 21 – Zoomed into the end of the sequence

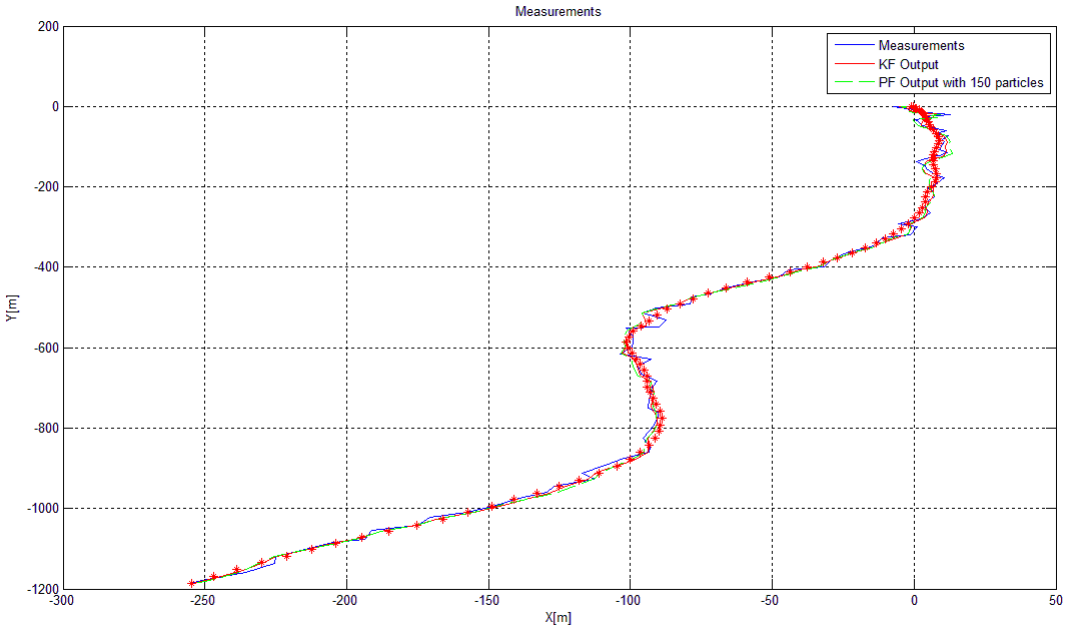

Figure 22 – The comparison between Kalman filter and particle filter with 150 particles

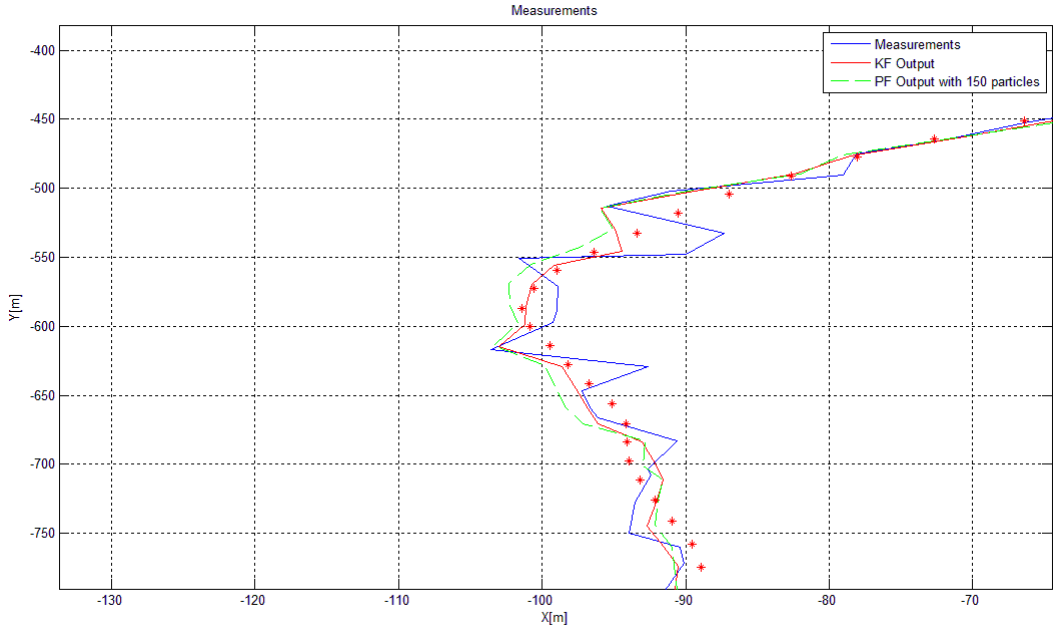

Figure 23 – Zoomed into the same blob in the middle of the sequence

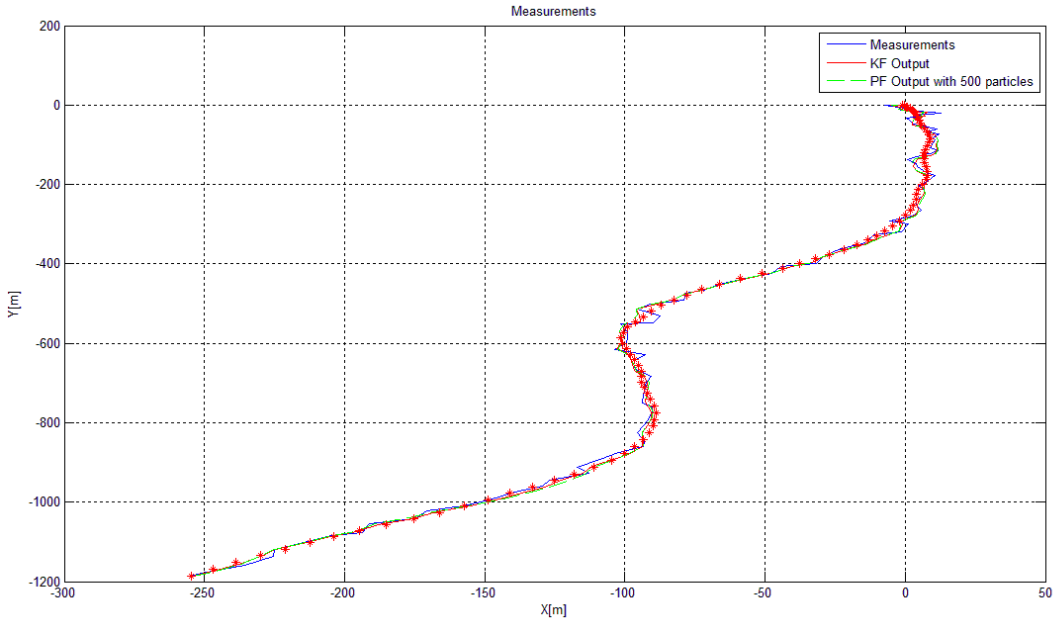

Figure 24 – The comparison between Kalman filter and particle filter with 500 particles

Before analyzing the results, it is necessary to clarify that the blue line stands for the measurements with noise, red points stand for the measurements without noise, the red line stands for the output of Kalman filter and the dashed green line stands for the output of particle filter with different number of particles. The aim of both Kalman filter and particle filter is to reach the measurements without noise sequence. Certainly, the Kalman filter gives the optimal results here and the particle filter algorithm proves to be effective, since its results are close to the optimal results provided by the Kalman filter.

The particle filter with 50 particles clearly moves away from the optimal Kalman filter result especially in the middle and at the end of the sequence. These regions are shown in Figure 20 and Figure 21. Generally, the results in Figure 19 are unsatisfactory for most cases. Therefore one can conclude that 50 particles were definitely not enough for this task.

The results are shown in Figure 22, when the number of particles increases up to 150. There are some obvious improvements in the results, specifically in the problematic regions with 50 particles. The convergence to the optimal Kalman filter

result can be easily seen from the Figure 23. In summary, the results with 150 particles are quite satisfactory.

To understand whether the observed error reaches to a steady level or not, the number of particles is increased again up to 500. If the results with 500 particles are studied carefully, one can easily say that there are minor improvements, especially in the beginning of the sequence, with respect to results with 150 particles. In some cases, even these minor improvements may be important for a particle filter. There is a trade-off between the accuracy and the computation time.

To observe the effects of the number of the particles to performance, MSE is calculated for particle filters with different number of particles. The results are given in Table 5.

Table 5 – The MSE results for Kalman filter and particle filter

MSE	Position X	Position Y
Kalman Filter	4.61	5.52
Particle Filter with 50 particles	11.09	13.86
Particle Filter with 150 particles	6.34	7.99
Particle Filter with 500 particles	5.67	6.79
Particle Filter with 1000 particles	4.99	5.86

The results show that the results of particle filter converge to the optimal Kalman filter results as the number of particles increases. Additionally, observed error reaches a steady level, after some increment in the number of particles. Finally, it is shown that the generic particle filter algorithm works well enough with acceptable number of particles.

CHAPTER 5

PROPOSED VISION ASSISTED TRACKING SYSTEM

In this chapter, details of how the methods presented in the previous chapters are combined to build an end-to-end object tracking system, and experimental results on different videos that present different type of challenges are presented.

The parts, such as data association, which is crucial to make the system perform correctly, does not belong to the optical flow step neither to the particle filtering step. Therefore, Section 5.1 gives implementation details on how the methods presented in the previous chapters are combined. Combining the information in the previous chapters is very important to make the system work. The videos selected for experiments [35], and various challenges in the videos are explained in Section 5.2. Results of the intermediate steps and the final tracking results are presented in Section 5.3.

5.1 Object Tracking System

The overall object tracking system consists of three main parts that were presented in the previous chapters, (*i*) background extraction and detection of foreground objects (*ii*) optical flow estimation and feature extraction on the foreground objects (*iii*) particle filter tracking. The following sections briefly explain how these steps are connected to each other to build the object tracking system.

5.1.1 Connecting the Background Extraction and Optical Flow

Connection of background extraction and optical flow is straightforward. Since the optical flow information is needed for Shi-Tomasi corner detection and improving

the tracking performance (by adding it as a correction term to the tracking result), only the optical flow vectors for the foreground objects are evaluated. Therefore, the input of the optical flow estimation step is the original video sequence and the binary masks for each frame obtained in the background extraction step, to determine where in each frame the optical flow should be computed.

5.1.2 Connecting Optical Flow with the Particle Filter Tracking

There are two outputs of the optical flow step: (*i*) the optical flow vectors (*ii*) features (Shi-Tomasi corner detection) of the foreground objects. Experimental results with and without using the optical flow vectors in the tracking step will be presented, however, eventually the optical flow information is not crucial for the tracking since tracking step can be executed without the optical flow vectors too. However, the features detected by Shi-Tomasi corner detection algorithm during optical flow step are crucial for the particle filter tracking step, since the feature centroids play a critical role constituting the measurements of the tracking step. One problem here is to assign the features in a given frame, to the features in the next frame to obtain the measurement trajectories of each object individually and generate the measurements for each object. In general there might be multiple objects in the video. Furthermore, objects that enter, or leave the video frame in the middle of the video sequence should also be handled.

5.1.2.1 Data Association

In videos, where multiple objects are being tracked simultaneously, it is important to avoid confusion of distinct objects and keep measurement updates of each object separately; otherwise particle filtering step would fail. This is called the data association problem. A visualization of the data association problem can be seen in Figure 25.

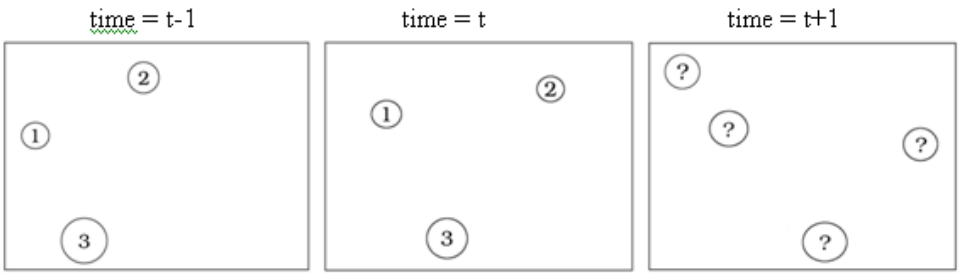

Figure 25 – Visualization of data association problem

Assume that each circle represents an object which is going to be tracked with the particle filter. When the foreground objects in the new frame are detected, these need to be assigned to the objects in the previous frames. This is not a straightforward problem because additional problems occur when (i) objects leave the frame, (ii) new objects enter the frame, (iii) some false alarms in background extraction creates non-existent foreground objects (iv) if objects temporarily disappear due to occlusion of multiple objects, misdetection in the background extraction step, or when objects move behind the foreground objects (e.g. passing behind a tree).

To solve these problems the following rules are used in this thesis:

1. **Target initiation:** When a *K* number of features appear at least *M* pixels away from all existing objects for at least *N* frames continuously, a new object is initialized. *K, M* and *N* are predefined numbers and parameters to the algorithm. Selecting these parameters is important, since if *K* or *N* is selected too low, the final results are more susceptible to misdetections in the background extraction step due to small (very low *K*) non-existent objects that are misdetections in the background extraction model and appear in the video sequence for a brief period (very low *N*).
2. **Data Association:** Measurement updates of all features in a given frame are the closest feature in the next frame in the (*X*, *Y*, *R*, *G*, *B, U, V*) space. *X* and *Y* are the positions, *R*, *G*, *B* denotes the color channels and *U*, *V* indicates the optical flow information. Therefore the feature point should be close in optical flow sense, in the spatial and in the color dimensions. There are two assumptions here:

- The objects move continuously. Consequently, *X*, *Y*, *U* and *V* for the same feature are not very different between two frames.
- The objects do not change color. Therefore, for each feature *R*, *G*, and *B* values are similar between two frames.

3. **Losing an Object:** When the objects are lost due to various reasons (e.g. last 10 previous frames all have three objects and the current frame has only one) perhaps because the object has left the frame or there is a misdetection in the object detection step, the detected objects in the current frame are assigned to the closest ones in the previous frame (as explained in the data association step) and the remaining objects are marked as lost. Losing an object more commonly occurs when an occlusion of two objects or occlusion with a background object happens.
4. **Re-finding an Object:** If the current frame has more objects than the previous frame, either there is a new object entered to the frame or an object that was lost a few frames ago has found again. First, the discovered objects in the current frame are assigned to the closest ones on the previous frame. Then, the remaining is either a new object, or a previously lost object. Therefore, before initializing a brand new object, it is checked that if this remaining object is close to a previously lost object in (*X*,*Y*,*R*,*G*,*B*,*U*,*V*) space, and if so it is assigned to the previously lost object, otherwise the target initialization logic is used as given in Step 1. An object that is lost for 10 frames is forgotten and removed from the list of comparisons, consequently if an object is lost for this many frames or more, it is assigned to a new object.

5.1.2.2 Feature Clustering

Another problem with the measurement updates is the features are not stable, e.g. the same object has 10 features, and 20 features in the next frame, and 15 features in the next frame and so on. Therefore, tracking each feature separately is quite hard, and furthermore, if the final goal is to track each object, tracking each feature separately is unnecessary.

For the results given in the following sections, a clustering of the features is performed and the cluster centers are tracked with the particle filter models and each object is represented by its feature cluster center in the (X,Y,R,G,B,U,V) space; hence, the above data association rules are applied to the feature cluster centers, not to the individual features separately. This approach is considered more successful, since the feature clustering gets rid of the different number of features per object type problems completely, and in addition it makes the assumptions in Step 2 of data association rules more reliable. The feature cluster centers are more continuous in the (X, Y, R, G, B, U, V) space than the individual features.

For clustering, the well-known mean shift algorithm is used [36]. Previously, some experiments with K-means [37], [38] are also performed, and the following problems have been examined:

- K, the number of clusters needs to be manually entered. For this, selecting K as the number of foreground objects in the image for each frame is tried. Such an approach only works well, if there is no occlusion in the frame (if two objects are very close to each other they will be assigned to a single cluster). In addition, running a connected components algorithm at each frame is computationally expensive.
- K-means needs a good initialization at each frame to perform reliably. In this work, using the cluster centers of the previous frame as the initialization is tried. Such an approach only works well, if the number of objects does not change throughout the video. Furthermore, the initialization in the beginning of the video cannot be handled with this approach.

After observing these problems it is decided to use mean-shift instead of K-means since it does not require the number of clusters as a parameter (it decides the number of clusters itself), and it does not require an initialization. It has only one parameter, the Parzen window bandwidth.

5.1.2.3 Mean Shift Clustering

Mean shift considers the feature points as sampled from an underlying probability density function and tries to model this distribution by using a smooth continuous non-parametric probability density model. The aim of mean shift is finding the peaks in this data distribution without computing the complete function. These peak points are assumed as the cluster centers. For the general framework of mean shift algorithm, mean shift defines a window around each data point and computes the mean. Then, it shifts the center of the window to the mean point and repeats the same procedure until the convergence is reached. Iterations shift the window to a denser region of the dataset until it converges to the densest region.

To estimate the density function of a random variable non-parametrically, the first step of mean shift algorithm is the kernel density estimation or Parzen window technique [39]. Kernel density estimator is stated in Equation 92.

$$f(x) = \sum_i k\left(\frac{\|x - x_i\|^2}{h^2}\right) \tag{92}$$

In this equation, x_i denotes the input samples; $k(r)$ is the kernel function or Parzen window and h is fixed kernel bandwidth. After $f(x)$ is computed its local maxima can be obtained by using gradient ascent technique. The gradient of $f(x)$ can be calculated by using (93). The terms in (93) are clarified in (94) and (95).

$$\nabla f(x) = \sum_i (x_i - x) g\left(\frac{\left\| x - x_i \right\|^2}{h^2} \right) = \left[\sum_i G(x - x_i) \right] m(x) \quad \textbf{(93)}$$

$$g(r) = -k'(r) \quad \textbf{(94)}$$

$$m(x) = \frac{\sum_i x_i G(x - x_i)}{\sum_i G(x - x_i)} - x \quad \textbf{(95)}$$

Here $m(x)$ is the mean shift vector. Finally, the estimation of the mode, which is stated in (96), can be achieved iteratively by replacing its locally weighted mean.

$$y_{k+1} = y_k + m(y_k) = \frac{\sum_i x_i G(y_k - x_i)}{\sum_i G(y_k - x_i)} \quad \textbf{(96)}$$

Mean shift algorithm is sensitive to the selection of the bandwidth *h*. A small *h* slows down; a large *h* speeds up the convergence, and also it affects the performance of the algorithm. However, as compared to the parameter selection problem in *K*-means, the parameter selection problem is considered much simpler. For *K*-means the number of clusters should be entered correctly, which is quite difficult to estimate in practice. For choosing the bandwidth of mean shift algorithm, one should consider the distance in the feature space, that two points are assumed to be similar. For example if one knows usually how big (in pixel dimensions) the objects are, one can easily select a reasonable bandwidth, which by nature is much simpler than deciding for the number of clusters as in *K*-means. In practice, *h* can be selected empirically by examining the output of the mean-shift algorithm. A smaller *h* will result in more number of smaller clusters (where *h*=0 each data point is a cluster by itself), and a bigger *h* will result in smaller number of bigger clusters (as h gets bigger, eventually all points will be a single cluster).

To show the effects of bandwidth selection to the mean shift algorithm, feature clustering is implemented with two inappropriate bandwidths. In order to observe the differences, one can compare the results shown in Figure 26.

Figure 26 – Feature clustering with small bandwidth

Figure 26 – Continued

Figure 27 – Feature clustering with high bandwidth

Figure 27 – Continued

5.1.3 Using Optical Flow Data as a Correction Term

Optical flow information can be a good guide in tracking problems. The main idea of this thesis is using the optical flow data as a correction term in particle filter tracking to improve the performance. A similar idea is used in [40]. Before particle

filter tracking, the same approach is implemented to Kalman filter tracking to see the effects of using the optical flow data, where the problem is linear.

Optical flow vector is added directly to the state equation like a noise term. However, optical flow vector is not random as a noise; it acts like a correction term, since the optical flow vector guides the correct direction. Therefore, for this method the state equation in (87) is modified to Equation 97.

$$x_k = Fx_{k-1} + w_k + u_k \quad \textbf{(97)}$$

Here u_k denotes the optical flow vector. The effects of using (97) instead of (87) will be shown in experimental results part.

5.1.4 Implementation Details of the Compared Kalman Filter Models

In order to investigate the effects of using optical flow vector to tracking performance, firstly the performance of three Kalman filter based trackers are compared. Since the problem is linear for this case, standard Kalman procedure is implied to achieve the tracking results.

The first of the Kalman filter based trackers is the one that is used in Section 4.2. The state vector of this Kalman filter is four dimensional, which contains the position and the velocity on x and y directions. And the measurements consists of only the positions of feature cluster centroids along x and y directions. This Kalman filter is referred as KF4D, the four dimensional Kalman filter. The state and measurement equations for KF4D are given in (87) and (88). The terms in these equations are also clarified in (89).

The second Kalman filter based tracker is referred as KF2D, the two dimensional Kalman filter. To compare the effects of the dimension of state vector and not tracking the velocities in each dimension, the dimension of state is reduced to two. Therefore, the state vector of KF2D contains only the positions along x and y directions. Similar to KF4D, the measurements of KF2D are the positions of feature

cluster centoids on x and y directions. The state and measurement equations are also the same, which is given in (87) and (88). However, the matrixes in these equations for KF2D are identity matrixes for this case.

The idea, which is covered in Section 5.1.3 is used in the last Kalman filter based tracker. It is two dimensional Kalman filter with optical flow correction and referred as KF2D-OF. The state and the measurement vector of KF2D-OF consist of the positions along x and y directions. The position information of feature cluster centorids is used as measurements again. Equation 97 is used as state equation and Equation 88 is used as measurement equation of KF2D-OF.

The performance results of these three Kalman filter based trackers are compared in Section 5.3. The effects of using optical flow data as a correction term can be also observed.

5.1.5 Implementation Details of the Compared Particle Filter Models

In order to investigate the effects of using optical flow information to particle filter tracking, again three particle filters are employed. In Kalman filter based tracking, the x and y positions of feature cluster centroids are used as measurements. For particle filter based tracking a distance measure is constituted by using the position information of the feature cluster centroids and the intensity information near the feature cluster centroids. Specifically, the distance measure is constituted by using five dimensional (x, y, R, G, B) space. First, N x N windows are taken near the feature cluster centroid in previous frame and near each particle in current frame. Then, the sum of square difference of the mean intensity value of each color channel between the window near the feature cluster centroid and the window near each particle are calculated. The spatial distance between the feature cluster centroid and each particle are also considered. By using this distance measure, the weights of each particle can be calculated. The weight of each particle is proportional to its distance to the feature cluster centorid. To formalize, the measurement equation, which is used in particle filter based tracking is stated in (98).

$$\begin{bmatrix} x_m \\ y_m \\ \overline{R}_m \\ \overline{G}_m \\ \overline{B}_m \end{bmatrix} = \begin{bmatrix} x_d \\ y_d \\ \overline{R}(x_d, y_d) \\ \overline{G}(x_d, y_d) \\ \overline{B}(x_d, y_d) \end{bmatrix} + v_k \quad \textbf{(98)}$$

In Equation 98, v_k denotes the measurement noise. All the three particle filter based trackers are using (98) as measurement equation.

For a comparative analysis, similar to Kalman filter based tracking, three particle filters are employed. The first particle filter is identical to the one that is expressed in Section 4.2. It has a four dimensional state vector, which contains the position and the velocity on *x* and *y* directions. Because of this reason it is referred as PF4D. The state equation of PF4D is given in Equation 87 and the measurement equation is given in Equation 98.

The second particle filter has a two dimensional state vector and it is referred as PF2D. Similar to KF2D, to compare the effects of the dimension of state vector and not tracking the velocities in each dimension, the dimension of state vector is reduced to two. It contains only the positions on *x* and *y* directions. The state and measurement equations are (87) and (98) respectively.

Optical flow data is used as a correction term in PF2D-OF, which is the two dimensional particle filter with the optical flow correction. State vector of PF2D-OF consists of the positions along *x* and *y* directions. The state equation of PF2D-OF is Equation 97 and the measurement equation of PF2D-OF is Equation 98.

The performance results of these three particle filter based trackers will be compared in the Section 5.3. The effects of using optical flow data as a correction term can be also observed.

5.2 Datasets

Datasets that are used for this thesis work are taken from an open source in [35], and are video sequences that have been used before in other publications. There are three datasets namely St. George sequence, PETS 2000 sequence and PETS 2001 sequence. Here, it is presented the results with 352 by 288 pixels versions of all video sequences. These datasets are quite challenging because they contain occlusion between the objects and the background, unstable background and excessively small objects due to the distance between the camera and the object. Further details of the dataset and challenges in each video are presented here.

5.2.1 St. George Sequence

The first dataset is St. George sequence, in which two pedestrians are walking across the crosswalk. St. George sequence, which is introduced in Figure 28, consists of 79 frames.

Figure 28 – St. George dataset sequence

The difficulties related with St. George dataset are;

1. The branches of the trees are unstable therefore it has a moving background.
2. There is an occlusion between two objects, which is a difficult situation, both moving object detection and data association.
3. There is also an occlusion between an object and background.
4. There is a small intensity difference between one of the objects and the background.

5.2.2 PETS 2000 Sequence

The second dataset is PETS 2000 sequence, which includes different objects in size and in velocity. Two cars, one pedestrian and one bird come into the scene. PETS 2000 sequence, which is illustrated in Figure 29, is made up of 520 frames.

Figure 29- PETS 2000 dataset sequence

Figure 29 – Continued

The difficulties for PETS 2000 sequence are as follows:

1. The difference between the pedestrian and the camera is large; therefore that object consists of only few pixels.
2. A bird comes into and gets out of the scene suddenly, which is challenging for object detection if it is wanted to detect as an object as well.
3. There is a small intensity difference between one of the cars and the background.

5.2.3 PETS 2001 Sequence

The last dataset, PETS 2001 sequence, consist different objects as two cars and three pedestrians. PETS 2001 sequence, which is presented in Figure 30, includes 800 frames.

Figure 30 – PETS 2001 dataset sequence

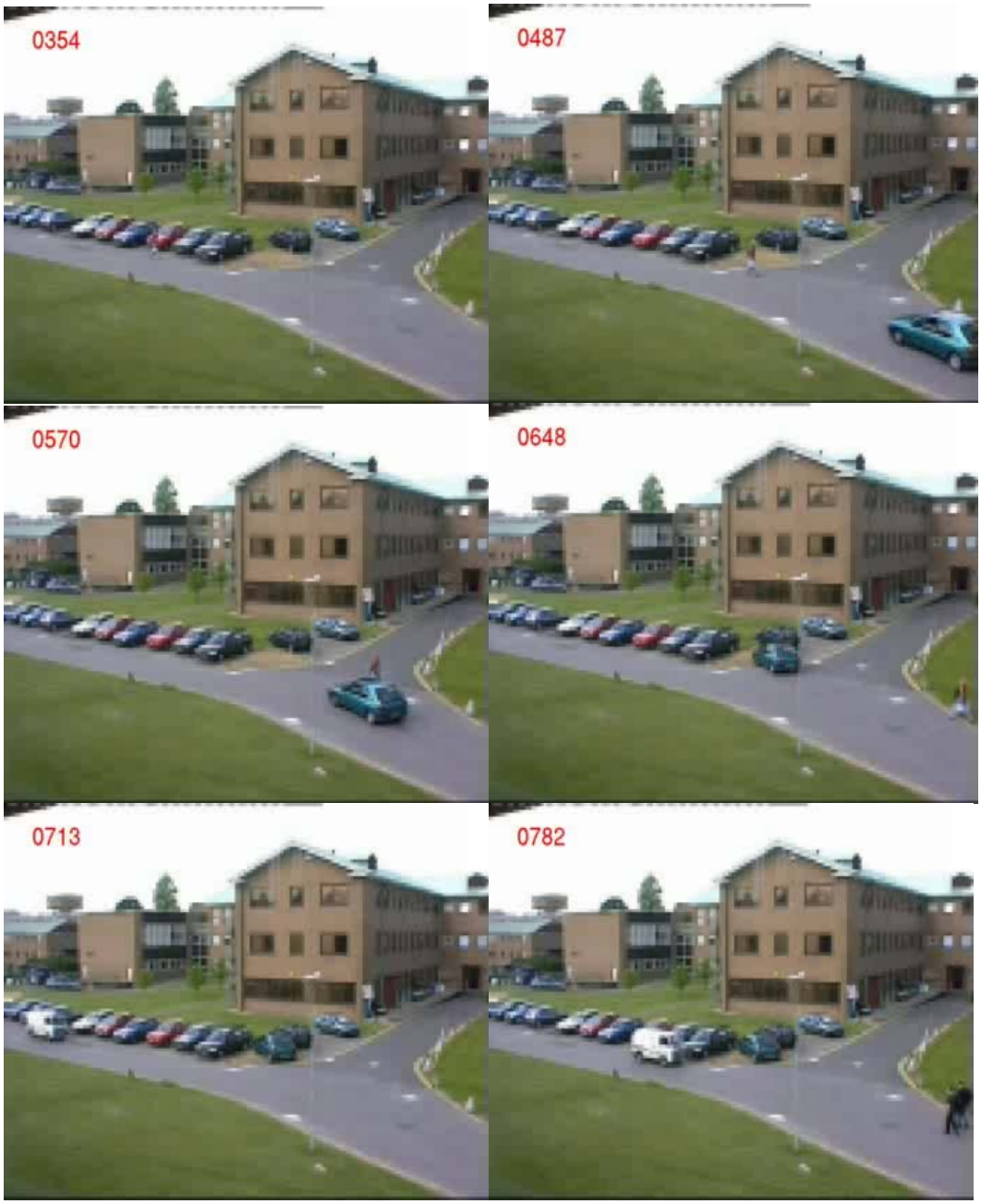

Figure 30 – Continued

The difficulties of PETS 2001 sequence can be listed as follows:

1. Similarly, the camera is far away from the scene and because of this reason the pedestrian in the beginning of the sequence hardly recognized.
2. There is an occlusion between the pedestrian and the car, which complicates moving object detection and data association.

5.3 Experimental Results

5.3.1 Results on Moving Object Detection

Moving object detection, which is expressed in Chapter 2, is implemented to these three datasets. The achieved results are stated in Figure 31 respectively.

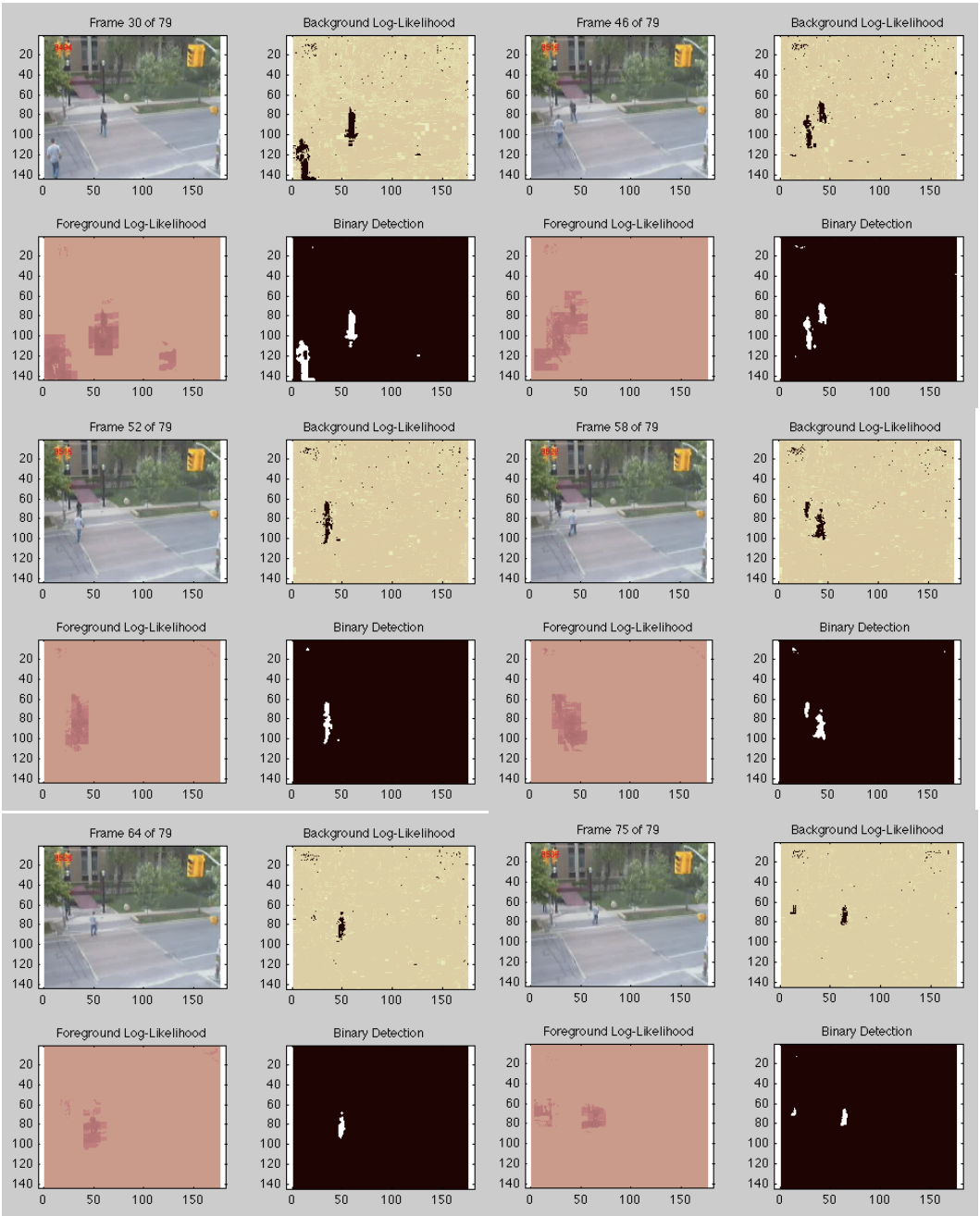

Figure 31 – Moving object detection results of St. George sequence

Moving detection results of St. George sequence is satisfactory. The first 15 frames are used for training of the background. Even there is a non-stationary background, because of the branches of the trees; the false alarm rate is very small. Except the occlusion in the end of the sequence, two pedestrians are detected successfully even they occlude each other in the middle of the sequence. Furthermore, the background detection algorithm copes well with the small contrast difference between the pedestrian and the road.

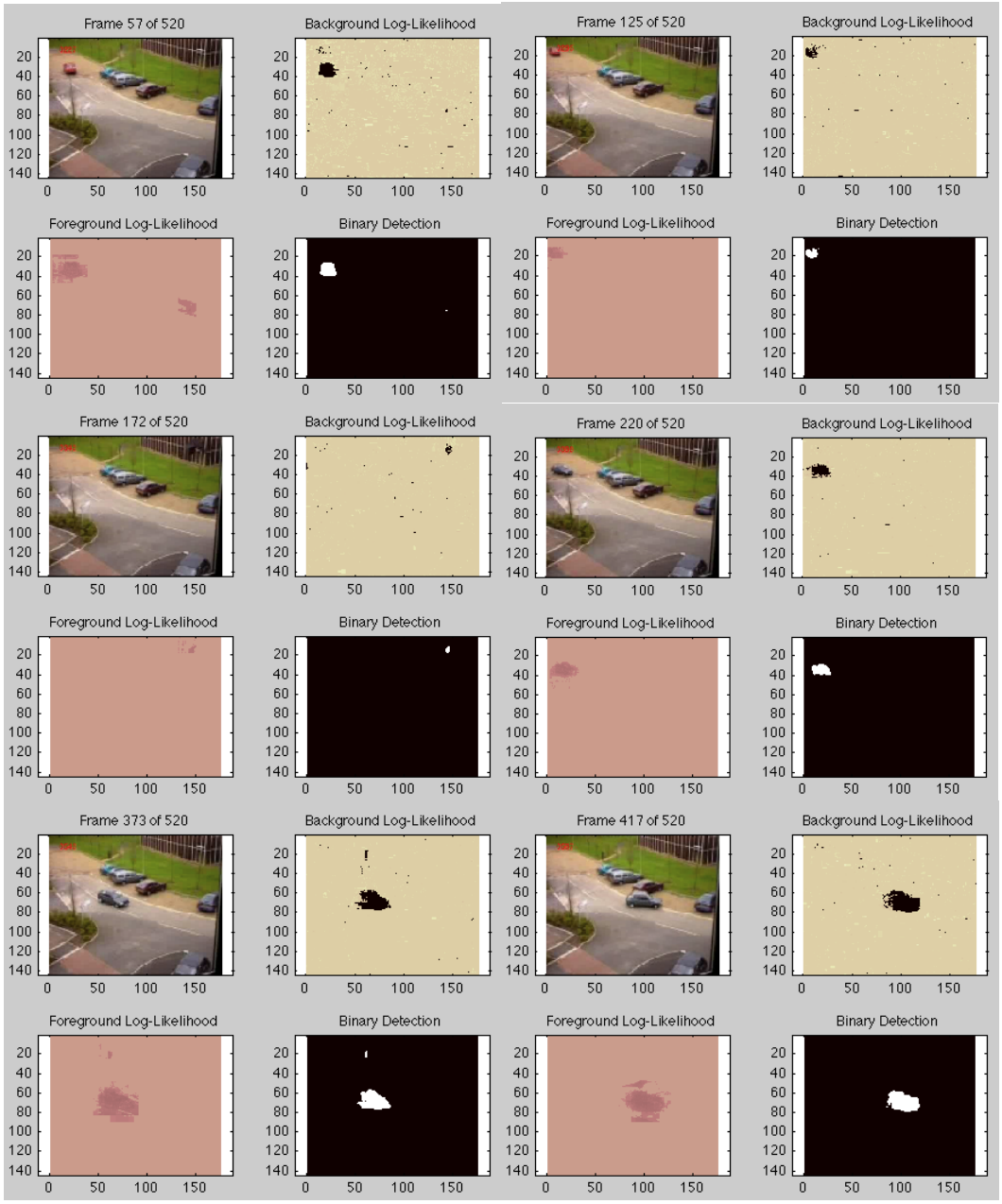

Figure 32 – Moving object detection results of PETS 2000 sequence

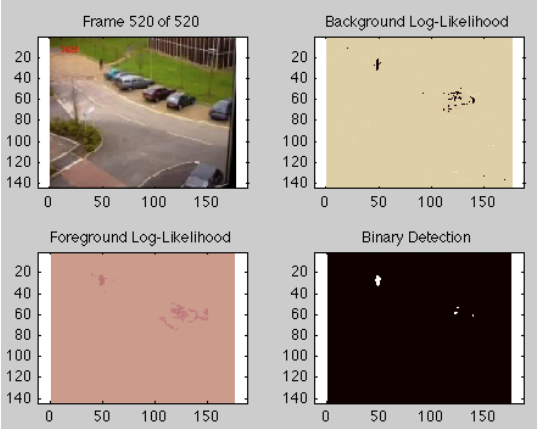

Figure 32 – Continued

For PETS 2000 sequence, again the first 15 frames are used to train the background. The false alarm is as small as a few pixels. The detection performance is well enough to detect a flying bird in the 172nd frame which enters the video frame for only a few seconds. All the moving objects are detected from beginning to end except the loss of detection of the pedestrian near 417th frame. However, in a few frames later, it is detected again. In the end of the sequence, some frames later after the car stops; it begins to disappear and starts merging into background.

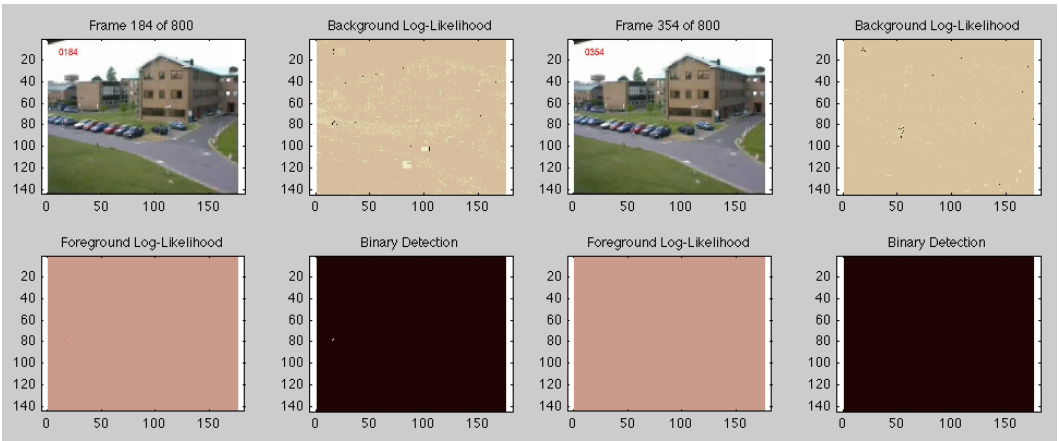

Figure 33 – Moving object detection results of PETS 2001 sequence

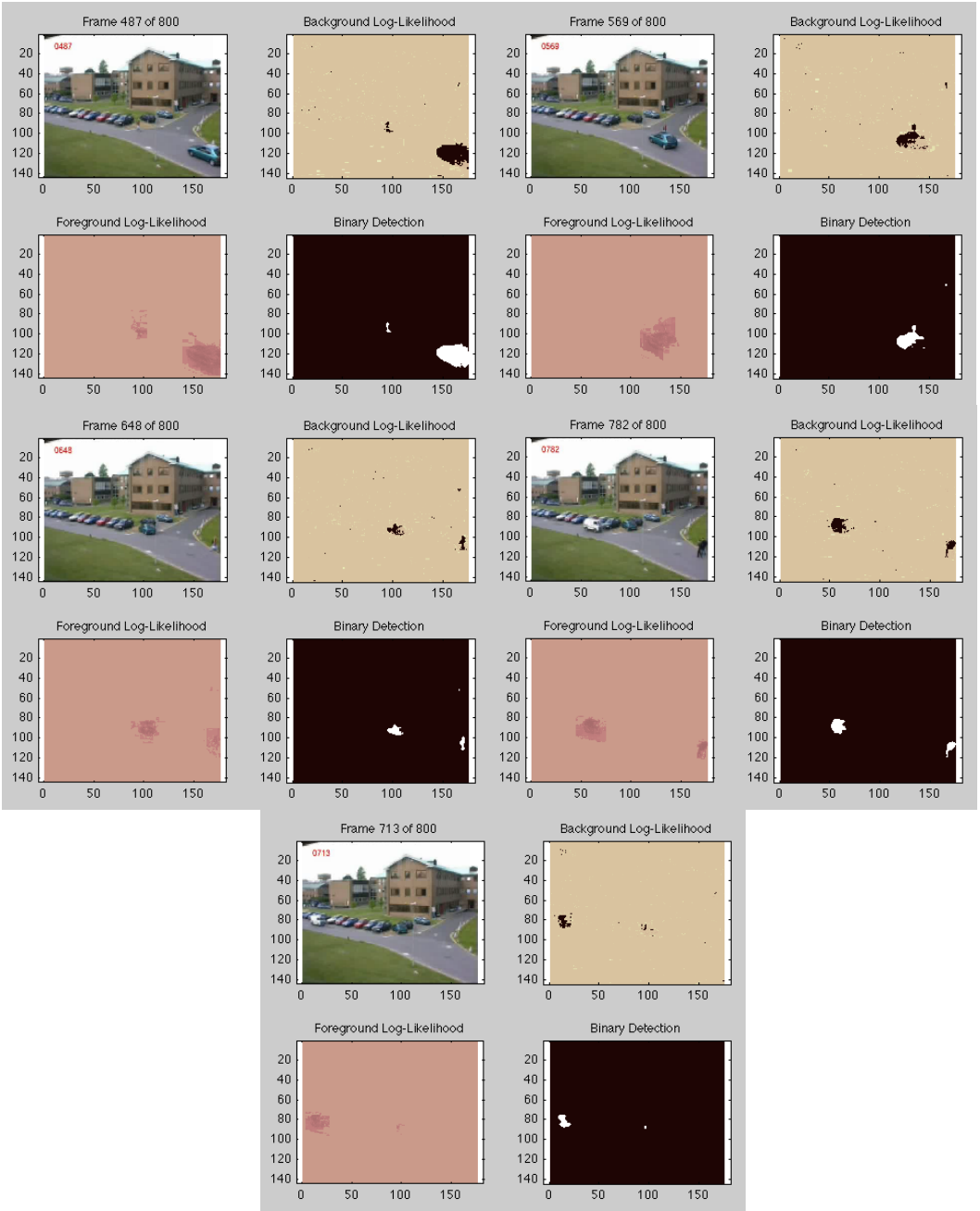

Figure 33 – Continued

PETS 2001 sequence is a challenging dataset. Since the distance between the camera and the objects is quite large, fewer pixels fall to the objects. Therefore the moving object detection performance decreases inevitably. In the beginning of the sequence, the walking pedestrian is not detected for a while because of the small contrast difference between pedestrian and the background. Except this all objects managed to detect. Similar to previous dataset, some frames later after the green car

parks, it begins to merge into the background. For this sequence 100 frames are used for training the background.

According to results, all the objects in three datasets can be detected successfully. Moving object detection algorithm suffers from occlusions, small contrast difference between the background and objects with very few number of pixels; however it is able to detect them after a while. Furthermore the false alarm rate is very small for all the three datasets.

5.3.2 Results on Feature Extraction

The feature extraction method, which is expressed in Chapter 3, is implemented to these three datasets. As expressed before, features are extracted only from the moving regions in the scene, which is shown in the previous part. The achieved results for these three sequences are stated in Figure 34, 35 and 36 respectively.

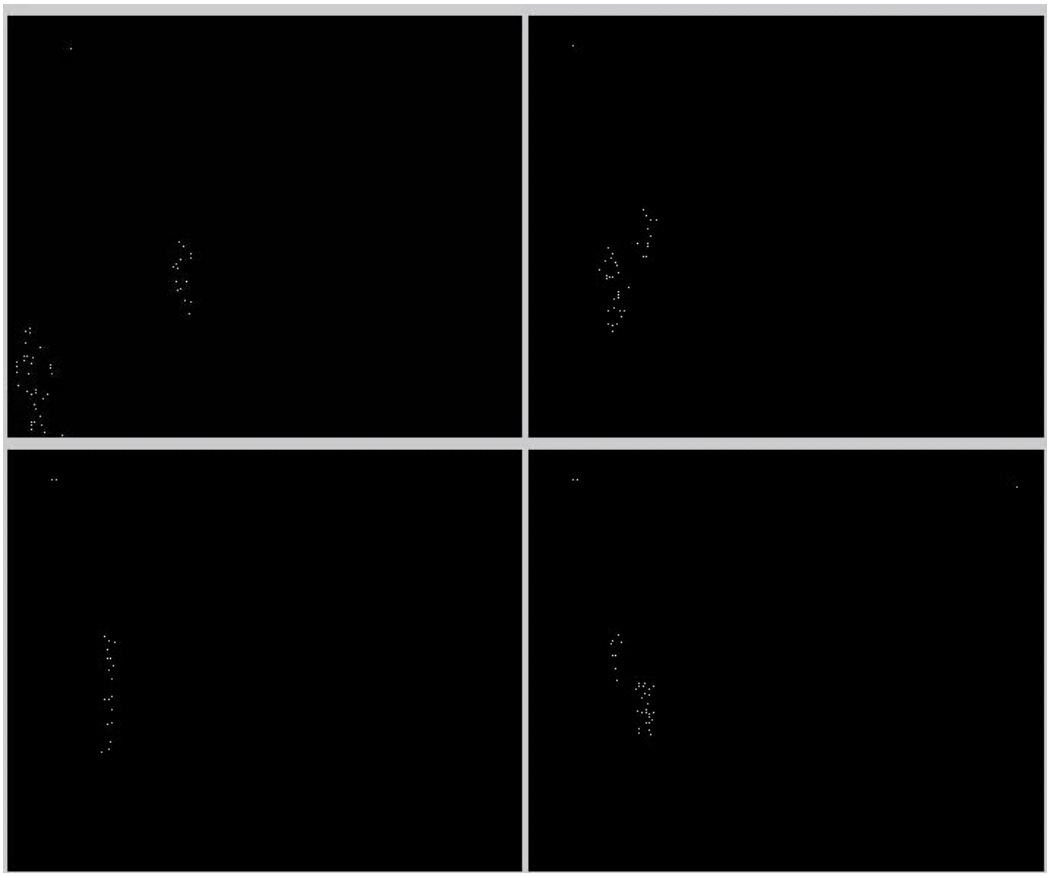

Figure 34 – Feature extraction results of St. George sequence

Figure 34 – Continued

The results show that quite sufficient number of features is extracted from moving regions. During the occlusion of the pedestrian, first the moving detection algorithm loses the object; in that case no features can be extracted naturally. However, in a few frames later moving detection algorithm catches the legs of the pedestrian and in that case several features are extracted.

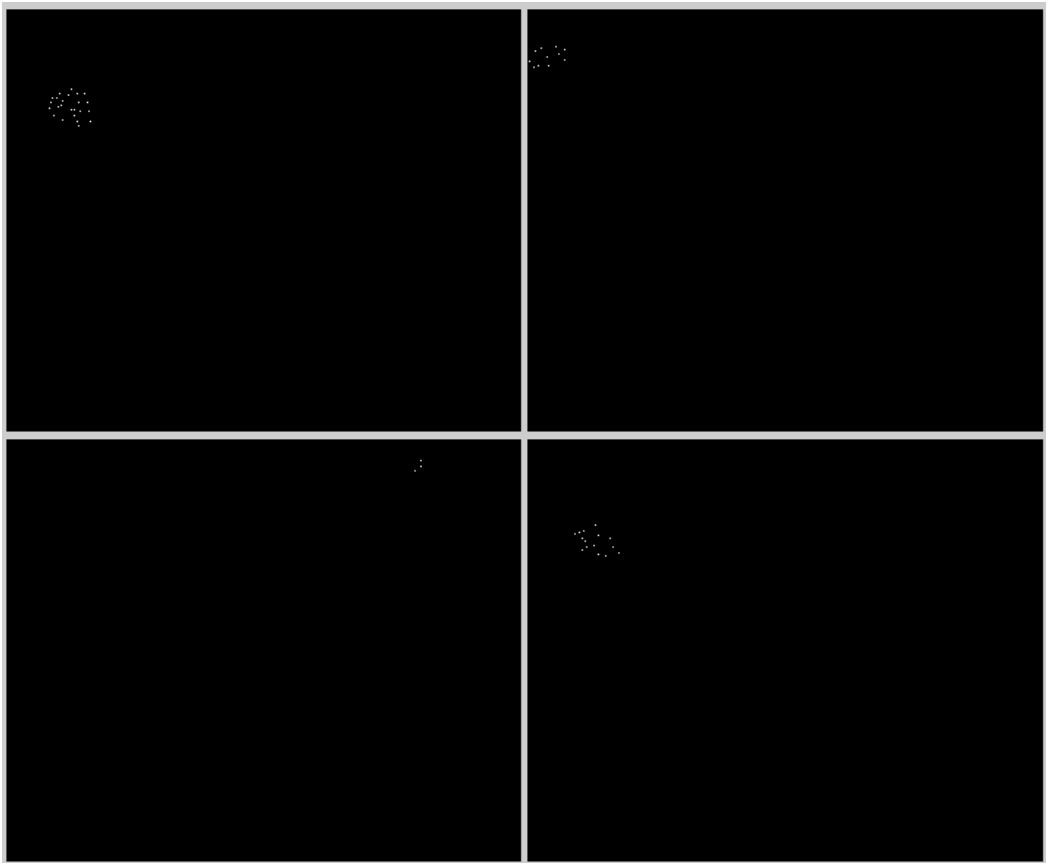

Figure 35 – Feature extraction results of PETS 2000 sequence

Figure 35 – Continued

Similarly, quite sufficient number of features is achieved for PETS 2000 sequence. At the 172nd frame, features can be extracted even from a flying bird. However, extracting features from the pedestrian, which is far away from the camera is not so easy for the algorithm.

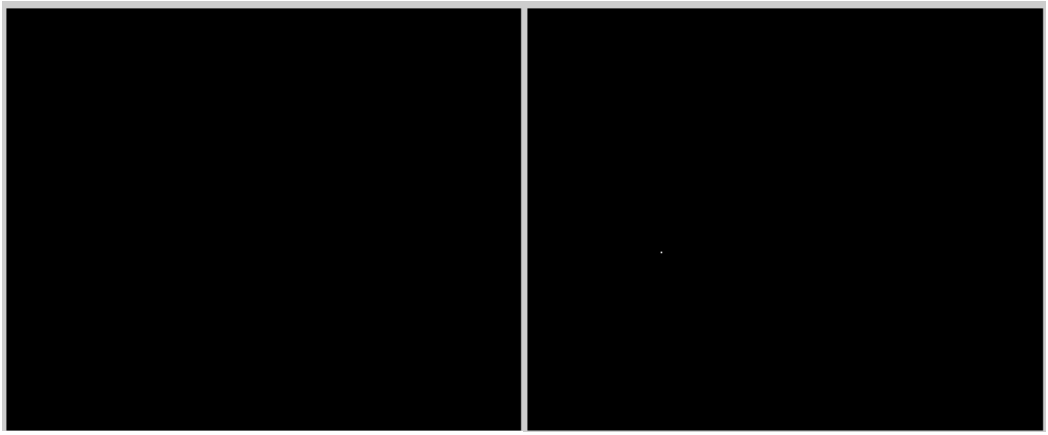

Figure 36 – Feature extraction results of PETS 2001 sequence

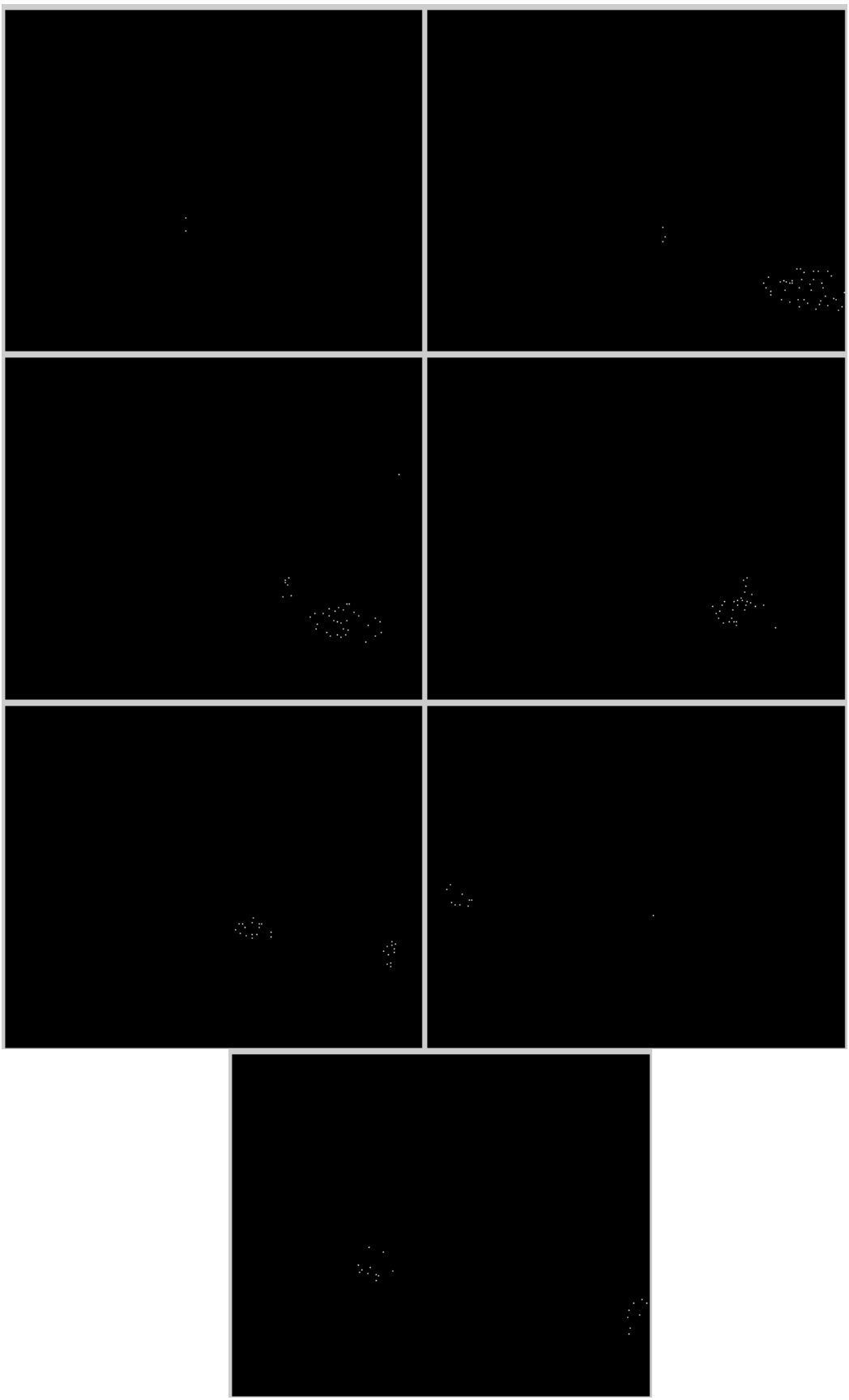

Figure 36 – Continued

Since, the walking pedestrian is not detected for a while by the object detection algorithm; features can not be extracted in the beginning of the sequence naturally. At the rest of the sequence, sufficient number of features extracted from the objects.

5.3.3 Results on Clustering

After the feature extraction step, the features should be clustered since feature cluster centers will be tracked, not the individual features all together. As explained above in Feature Clustering part, features are clustered by using mean shift algorithm. The achieved results for all the three datasets are introduced in Figure 37, 38 and 39 and the center of each cluster in each frame is marked.

Figure 37 – Feature clustering results of St. George sequence

Figure 37 – Continued

The results show that feature clustering can be problematic in case of two objects occluding each other. Additionally, if the distance between the features is large enough, the features from the same object can be clustered as two different clusters.

Figure 38 – Feature clustering results of PETS 2000 sequence

Figure 38 – Continued

The results are quite satisfactory. There are only several false alarms, mainly caused by the high distance between the features.

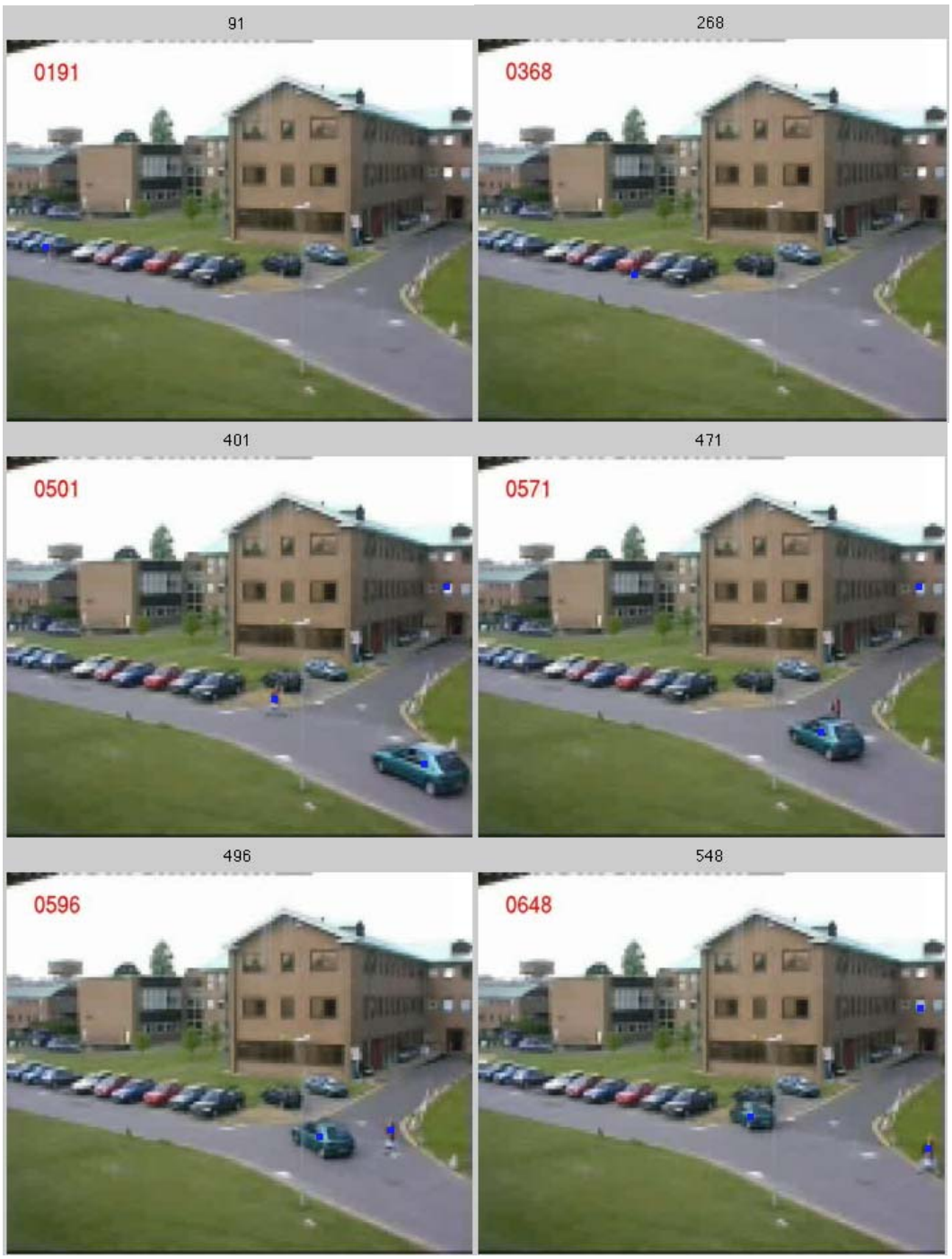

Figure 39 – Feature clustering results of PETS 2001 sequence

Figure 39 – Continued

If the results are investigated, except the occlusion moment, they are quite successful. There is one persistent false alarm in the background.

5.3.4 Results on Kalman Filter Tracking

To understand if the optical flow information would be sufficient by itself for tracking without any tracking step (Kalman or particle), the following experiment is done.

- The object is initialized to the first frame it appears in the video sequence, note that this corresponds to the measurement in the first time instance for the Kalman/particle filter implementations.
- Afterwards, instead of executing the Kalman/particle filter update equation, the measurement is updated by adding the optical flow vector at that point to the measurement.

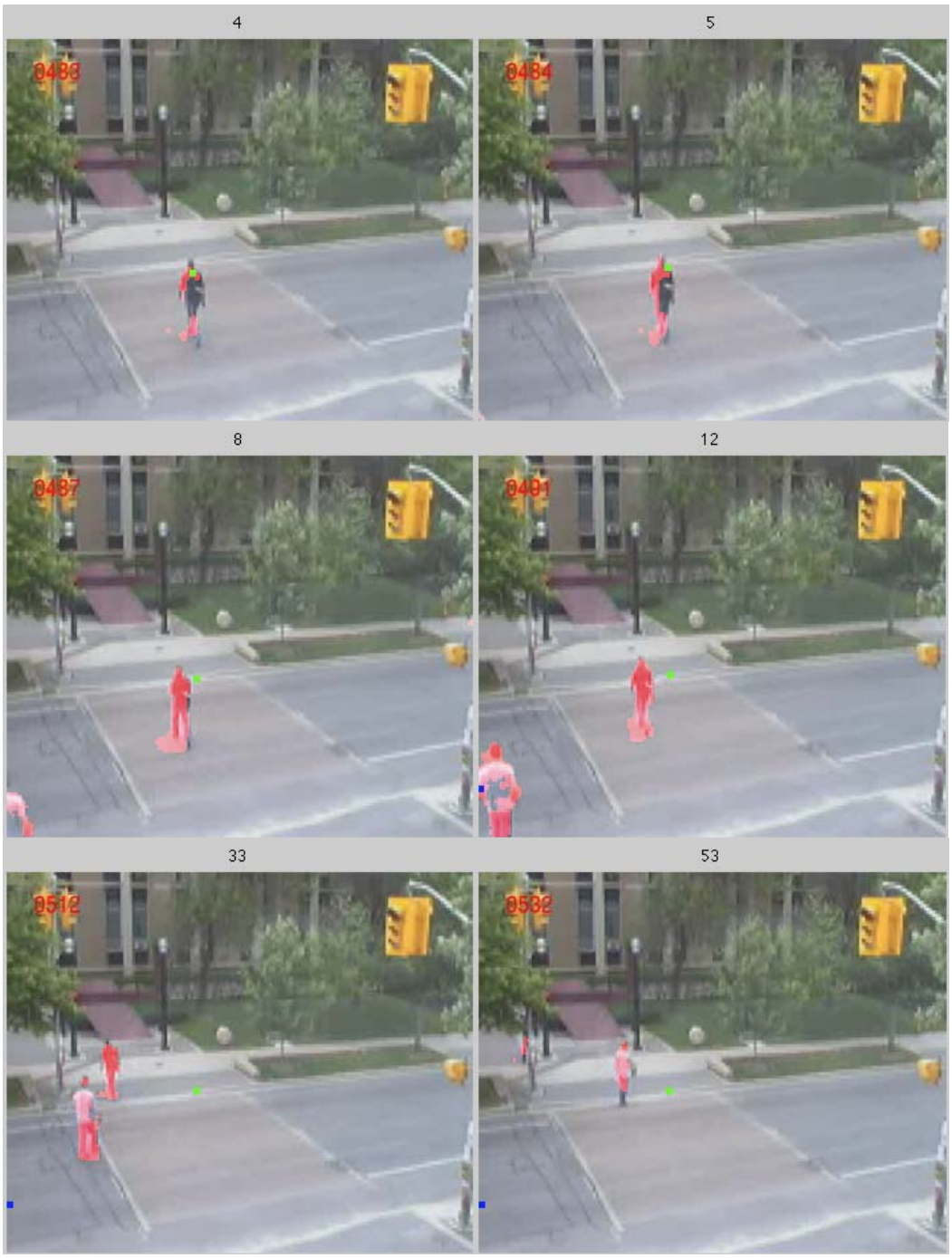

Figure 40 – Tracking results with only optical flow information

The problem with this approach is that without the Kalman/particle filter update error accumulation is unavoidable, and also once the tracker makes a single mistake and moves on the background, it cannot recover. For example, at any time instance an imperfect optical flow vector can send the tracker to a background pixel where there is almost no movement and optical flow vectors are all zero, hence the tracker

will not be able to find the object any more. The results obtained with this method are shown on the St.George dataset in Figure 40.

The results show that, in the beginning of the sequence optical flow information tries to track the objects for a few frames. However after a few frames, optical flow information sends the tracker to the background and tracker stuck into background. In summary, although it improves the tracking performance, it is obvious that the optical flow vector based tracking by itself is not a good idea.

After all of these preprocessing steps, the idea that using optical flow data as a correction term is implemented to Kalman filter tracking firstly. As mentioned before, three different Kalman filter models, namely KF2D, KF2D-OF and KF4D, are compared. Tracking process is implemented to the datasets with these three Kalman filters and the results are compared here. The tracking results for St. George sequence are stated in Figure 41, 42 and 43.

Figure 41 – Tracking results of St. George sequence with KF2D

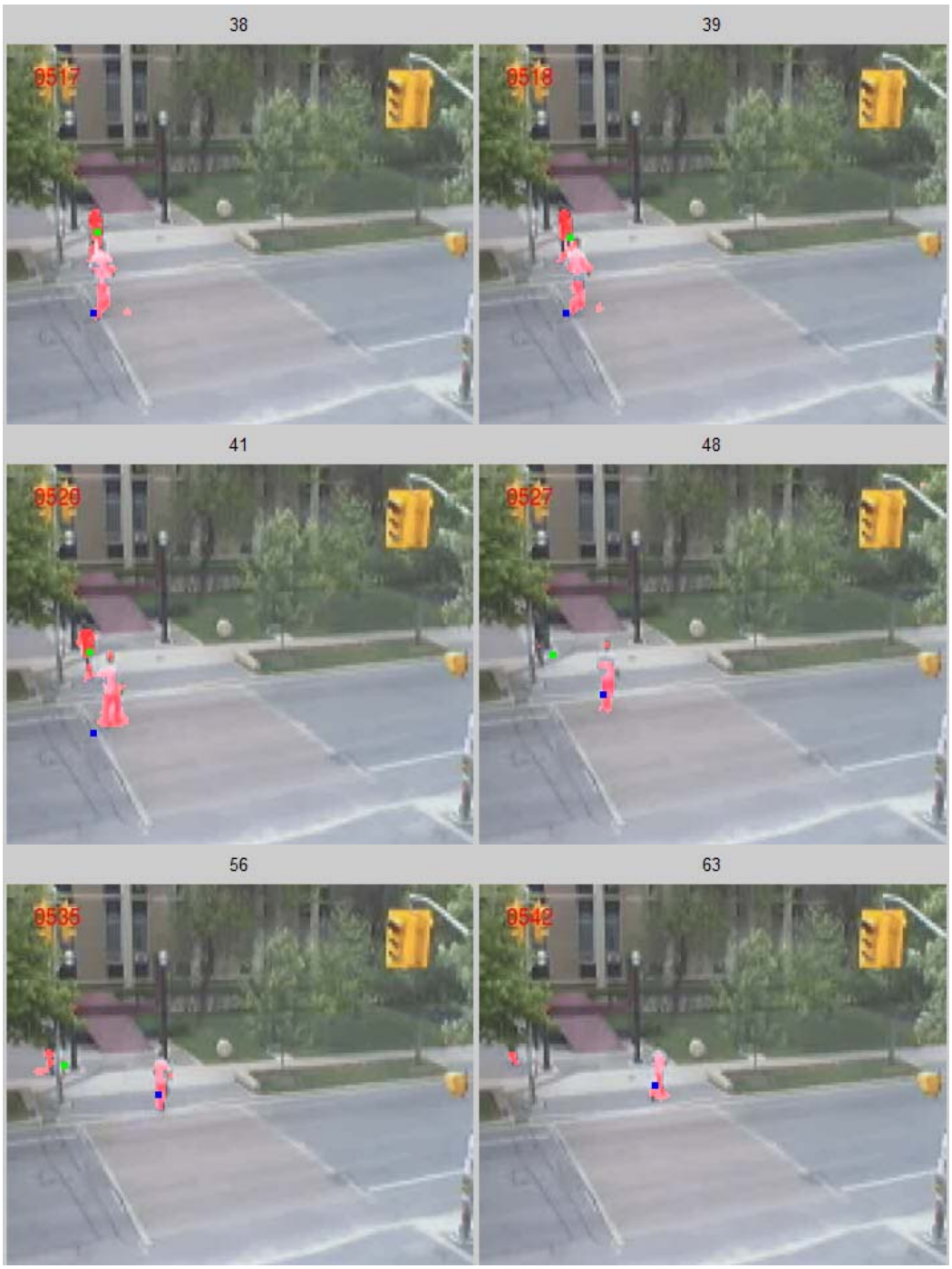

Figure 41 – Continued

Figure 42 – Tracking results of St. George sequence with KF2D-OF

Figure 42 – Continued

Figure 43 – Tracking results of St. George sequence with KF4D

Figure 43 – Continued

If the results are analyzed carefully, it can be easily observed that KF2D-OF and KF4D show a much better performance throughout the sequence comparing with KF2D. Additionally, these two filters cope with the occlusion between the object and the background better than KF2D does. Therefore, it is clear that using optical flow data as a correction term improves both the tracking performance and capability of handling the occlusion. Tracking performance of KF2D-OF and KF4D are comparable in general. However, KF4D can handle the occlusion better than the KF2D-OF does. Therefore, KF4D gives the best tracking results for St. George sequence. The tracking results for PETS 2000 sequence are introduced in Figure 44, 45 and 46.

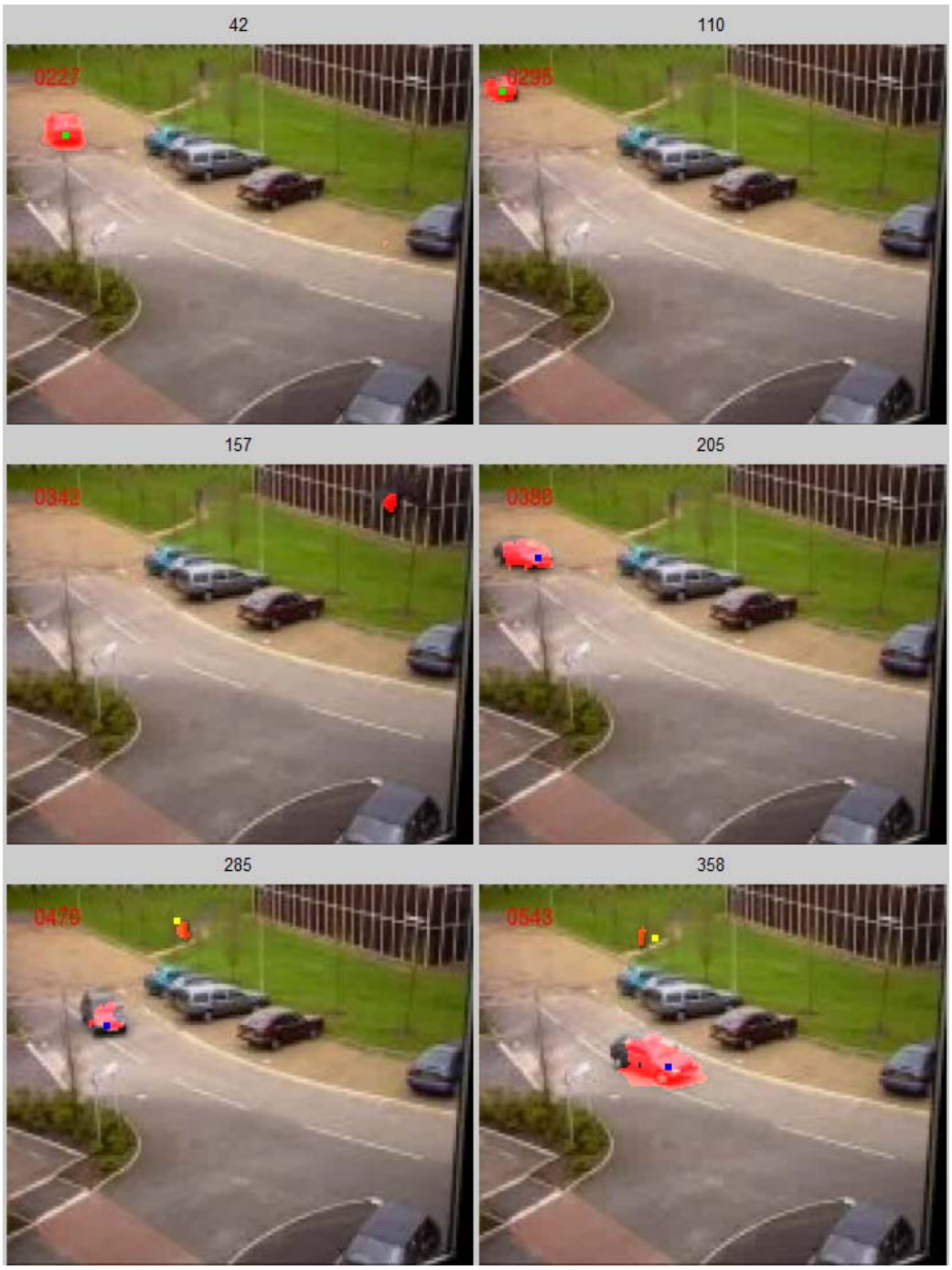

Figure 44 – Tracking results of PETS 2000 sequence with KF2D

Figure 44 – Continued

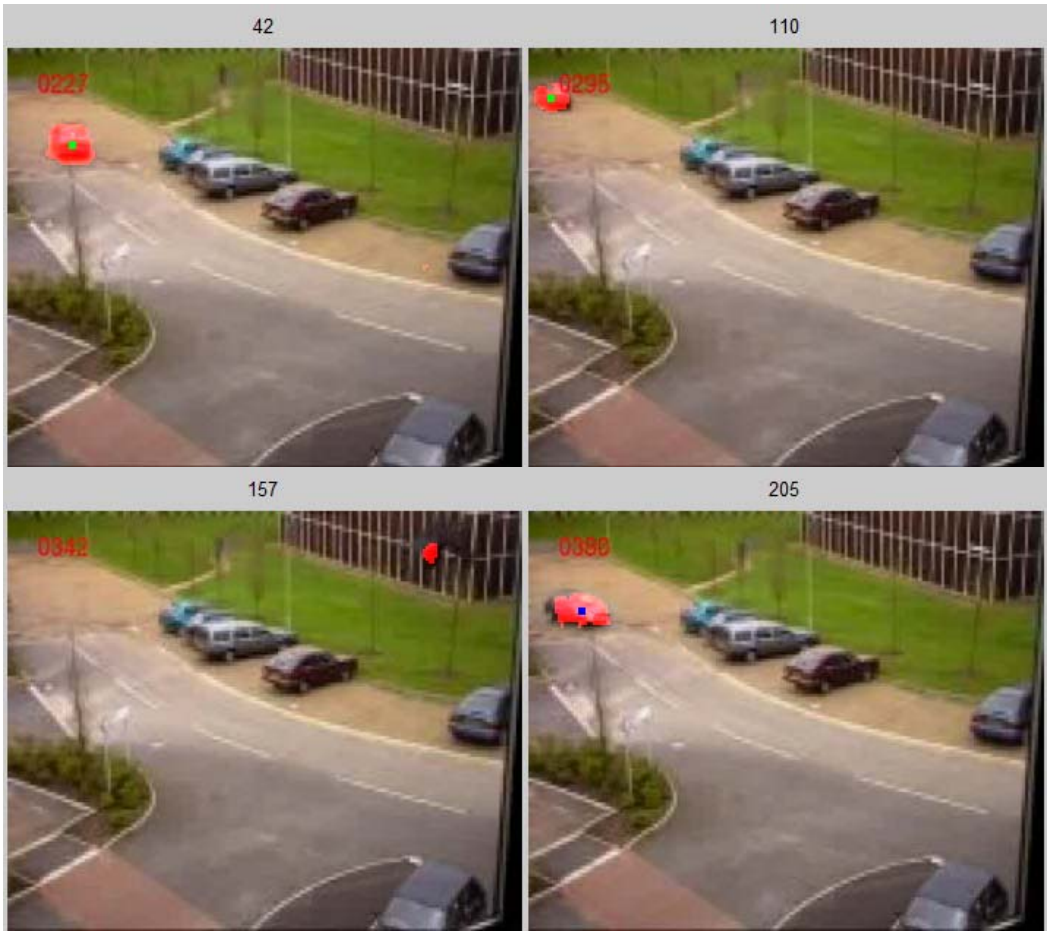

Figure 45 – Tracking results of PETS 2000 sequence with KF2D-OF

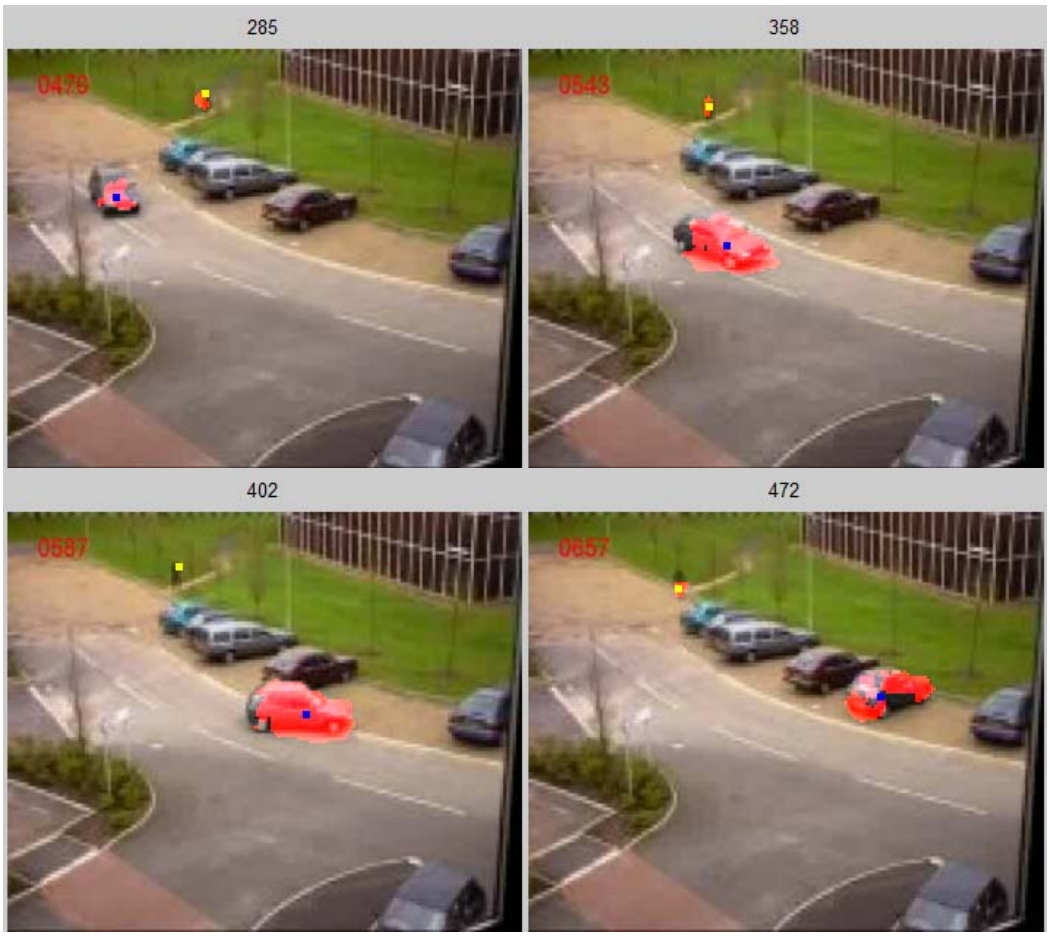

Figure 45 – Continued

Figure 46 – Tracking results of PETS 2000 sequence with KF4D

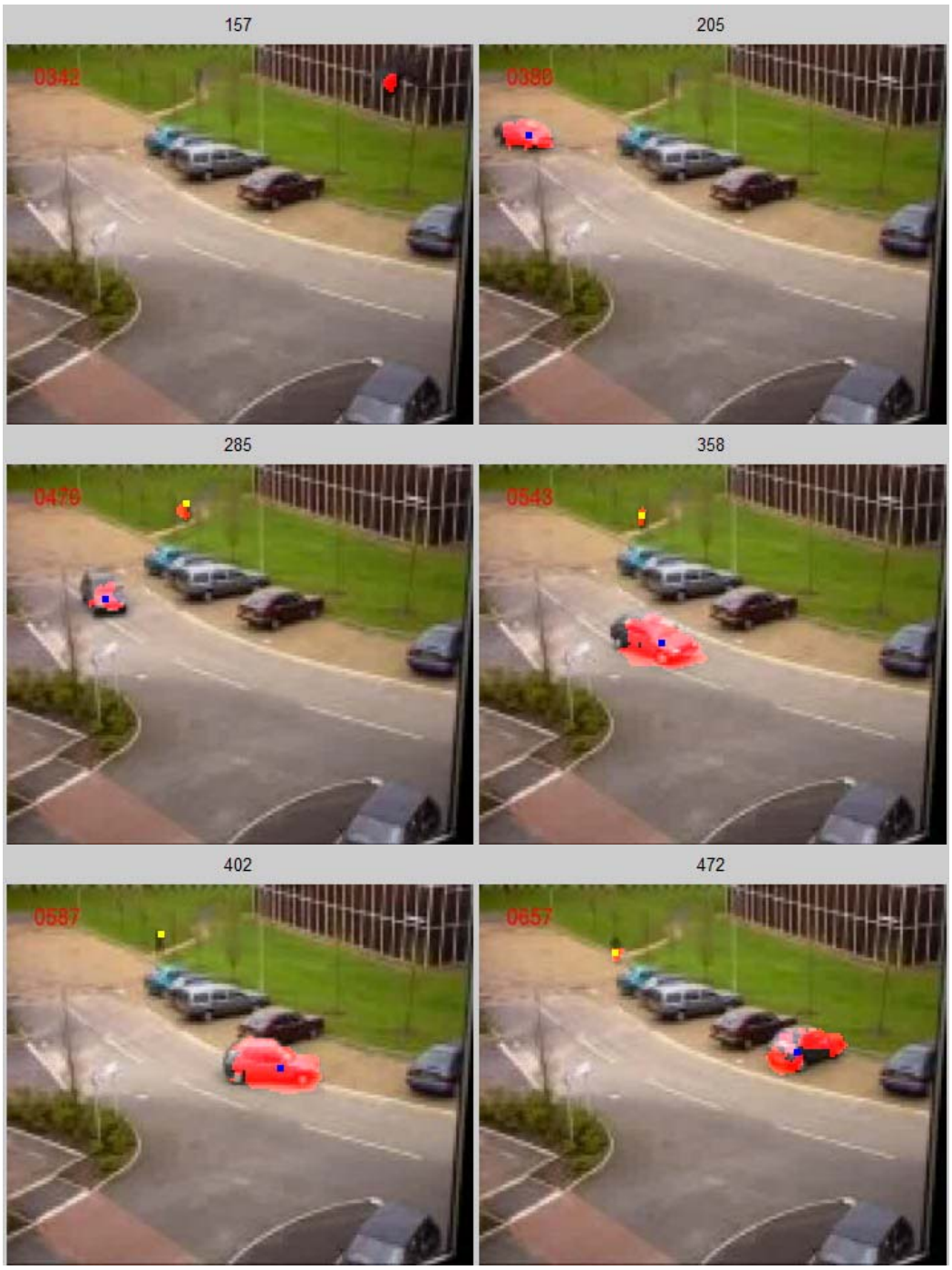

Figure 46 – Continued

If the results of KF2D and KF2D-OF are compared, it can be observed that, optical flow clearly improves the tracking results. By the help of optical flow data, the tracking results are more consistent especially for the pedestrian which walks at the background. Similar to KF2D-OF, KF4D results are better than KF2D results. KF4D can also cope with the tracking of the pedestrian, which is difficult for PETS 2000 sequence. The tracking performances of KF2D-OF and KF4D are almost

identical; the tracking performances for the moving cars are the same for these filters. However for the tracking of the pedestrian KF4D shows a slightly better performance with respect to KF2D-OF. Therefore, KF4D gives the best tracking results for PETS 2000 sequence. The tracking results for PETS 2001 sequence are presented in Figure 47, 48 and 49.

Figure 47 – Tracking results of PETS 2001 sequence with KF2D

Figure 47 – Continued

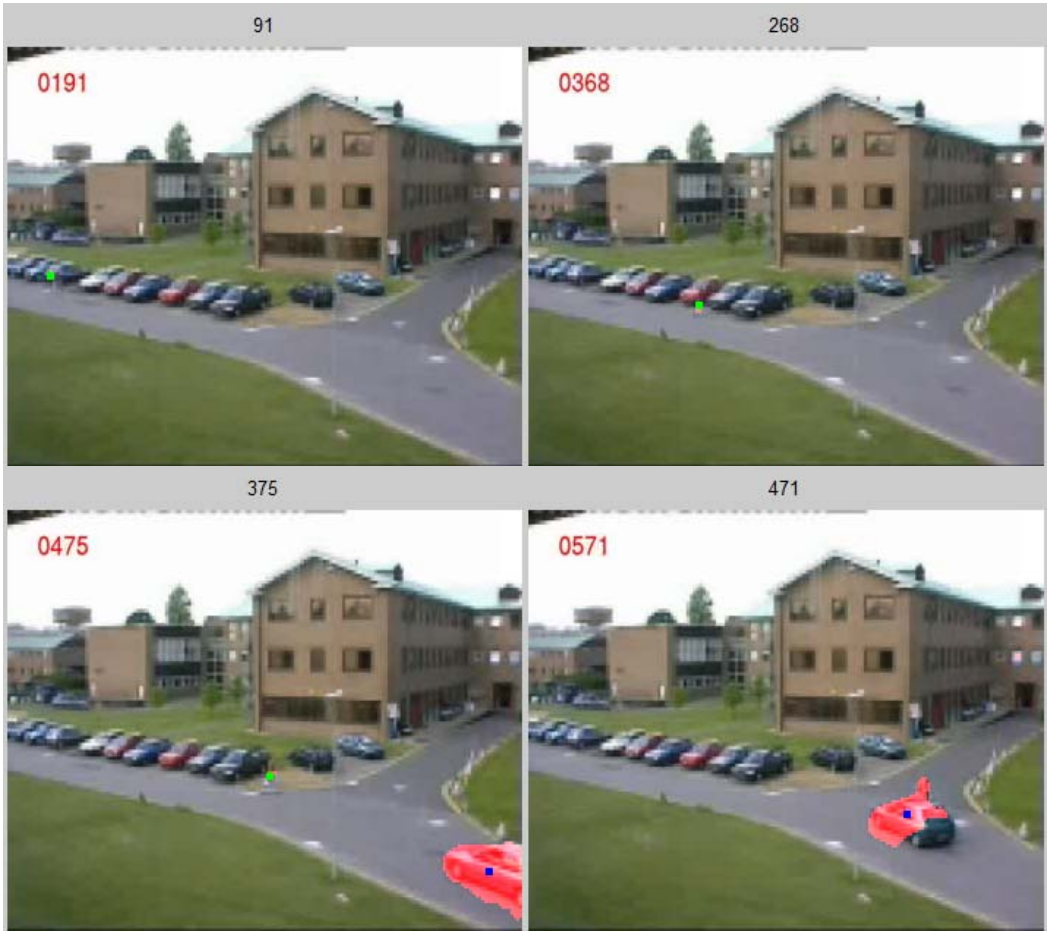

Figure 48 – Tracking results of PETS 2001 sequence with KF2D-OF

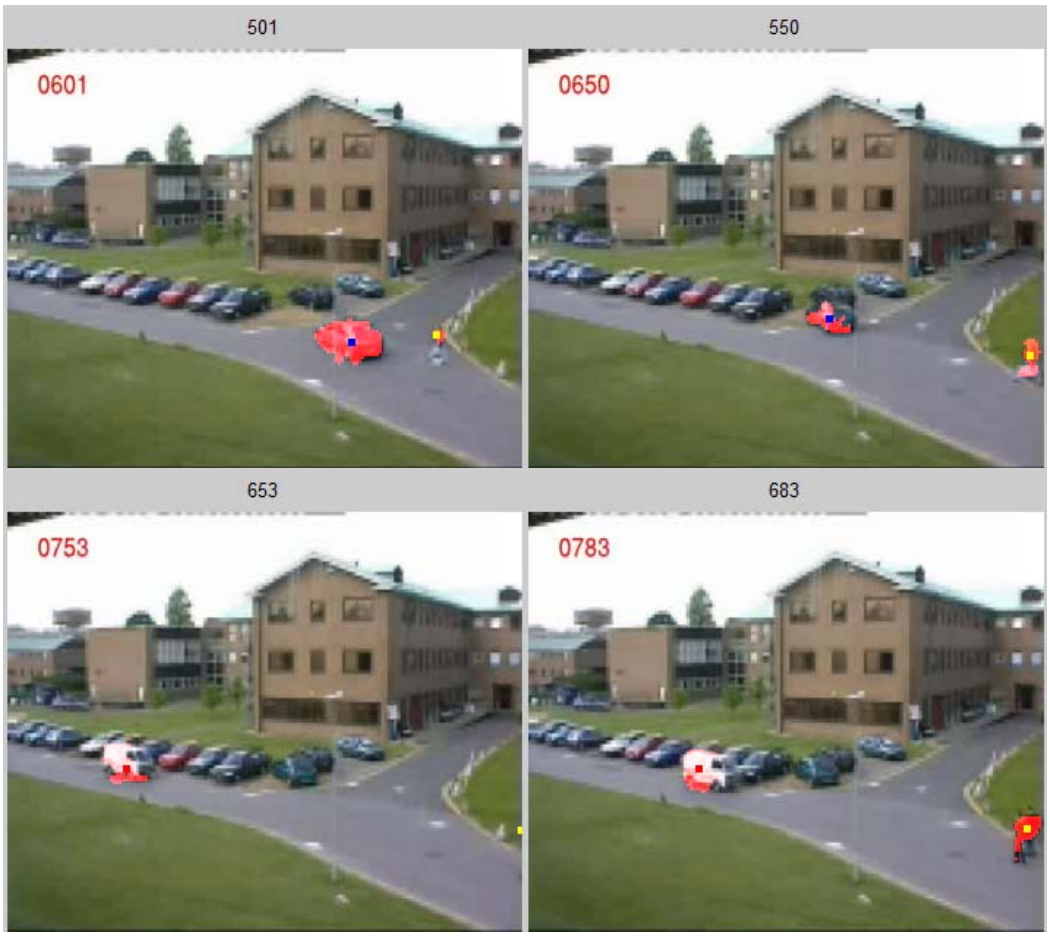

Figure 48 – Continued

Figure 49 – Tracking results of PETS 2001 sequence with KF4D

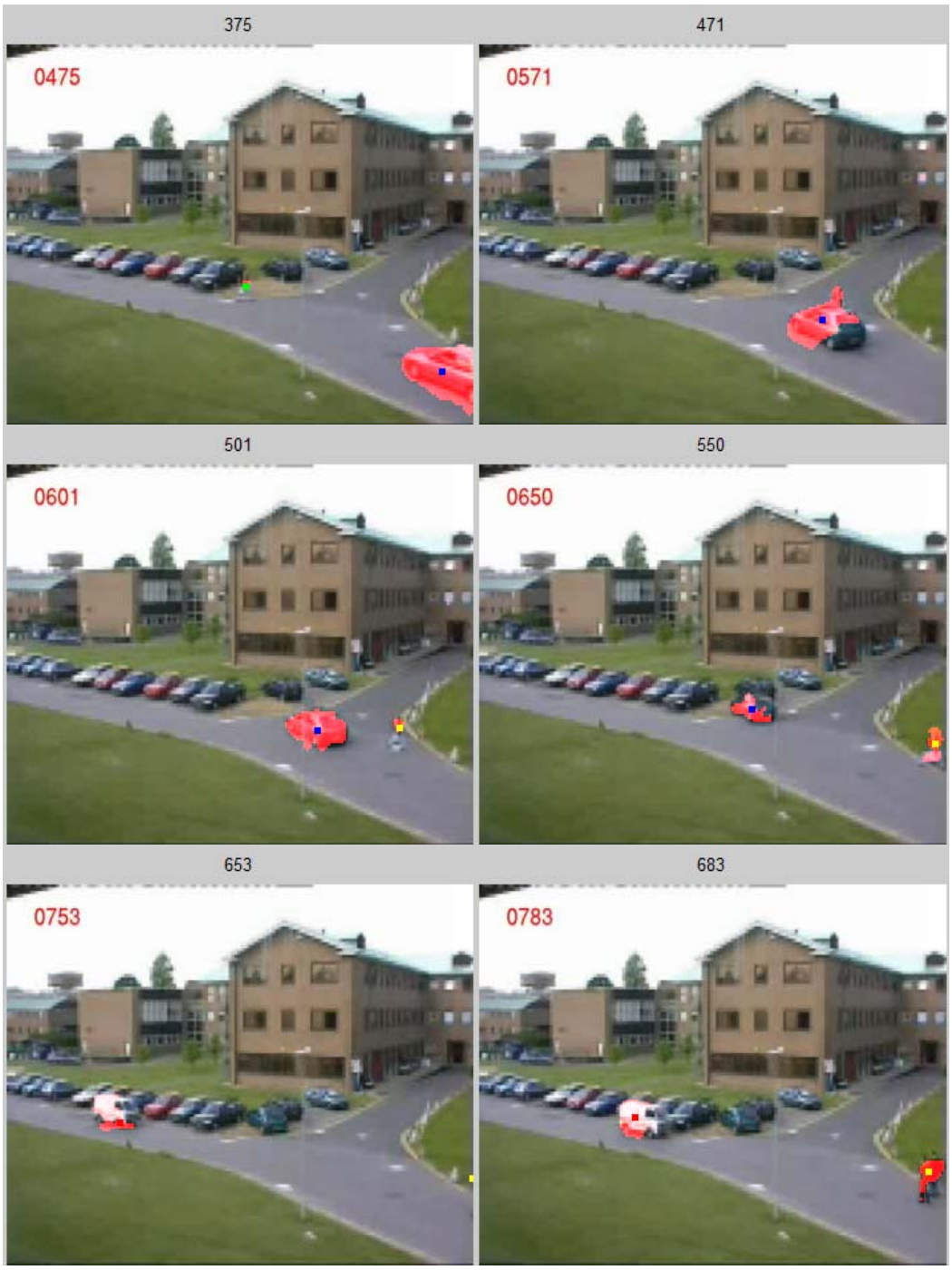

Figure 49 – Continued

As compared with KF2D, KF2D-OF shows a better tracking performance throughout the PETS 2001 sequence. Particularly, optical flow data helps the Kalman filter while tracking the pedestrian after the occlusion. In comparison with KF2D, KF4D clearly has more reasonable tracking results. The tracking results of KF2D-OF and KF4D are comparable again. There are slightly differences between the performance of two Kalman filters however if the results are carefully analyzed,

it can be observed that KF2D-OF has better tracking results especially when occlusions occur. Therefore, KF2D-OF gives the best tracking results for PETS 2001 sequence.

5.3.5 Results on Particle Filter Tracking

Finally, the idea that using optical flow data as a correction term is implemented to particle filter tracking. As mentioned before, three different particle filter models namely PF2D, PF2D-OF and PF4D, are compared. The details of the particle filter are as follows:

- 500 particles are used in all experiments
- Resampling if the effective sample size is less than half of the number of particles.
- The measurement update is done based a five dimensional feature space as given in Section 5.1.5 with 5 x 5 windows.

Tracking process is implemented to the datasets with these three particle filters and the results are compared here. To visualize the feature points, small yellow dots are used. The tracking results for St. George sequence are introduced in Figure 50, 51 and 52.

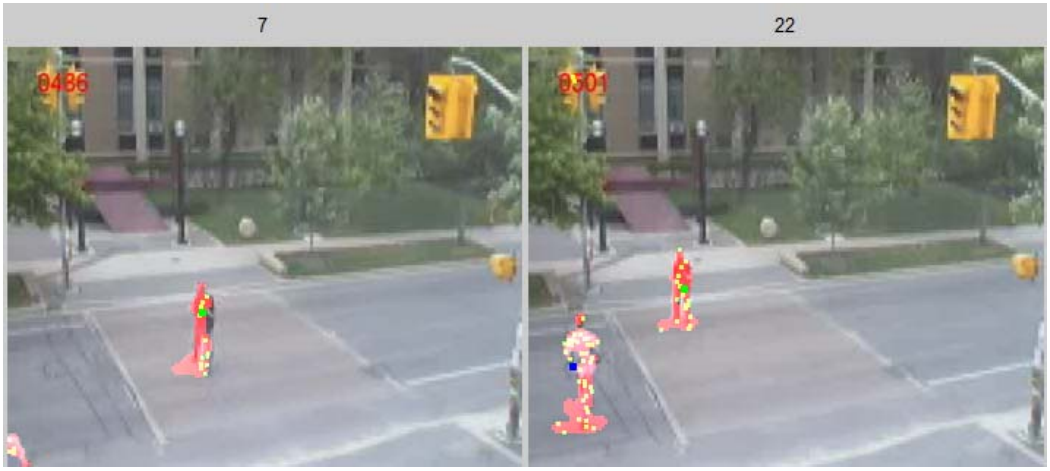

Figure 50 – Tracking results of St. George sequence with PF2D

Figure 50 – Continued

Figure 51 – Tracking results of St. George sequence with PF2D-OF

Figure 51 – Continued

Figure 52 – Tracking results of St. George sequence with PF4D

Figure 52 – Continued

The results show that, PF2D-OF and PF4D perform a better tracking performance throughout the St. George sequence. Especially, PF2D can not handle the occlusions between two pedestrians and between the pedestrian and the background. The general tracking performance of PF2D-OF and PF4D are comparable. However, with the help of optical flow information PF2D-OF copes the occlusion better than PF4D does. Therefore PF2D-OF shows the best tracking performance for St. George sequence. The tracking results for PETS 2000 sequence are presented in Figure 53, 54 and 55.

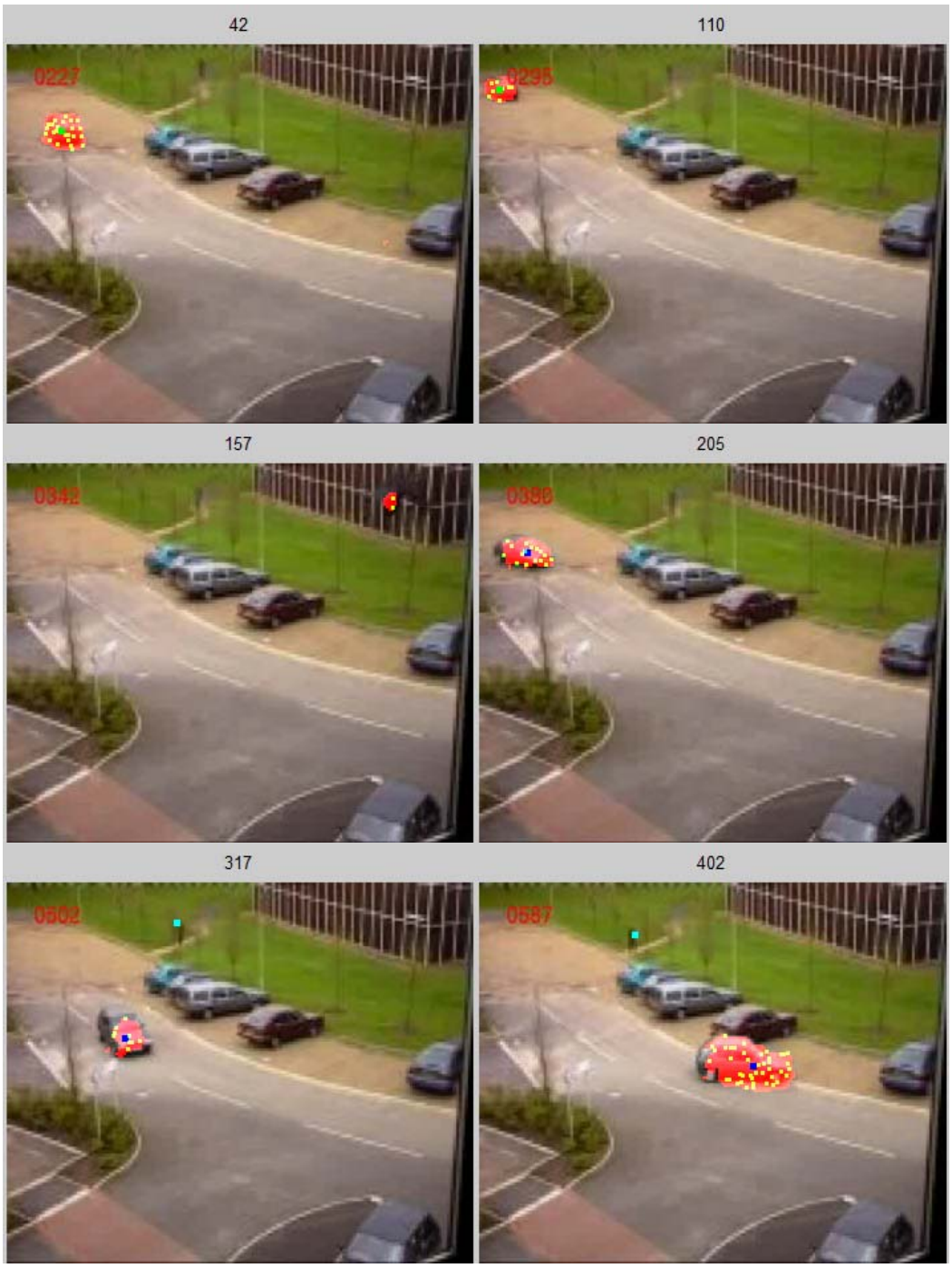

Figure 53 – Tracking results of PETS 2000 sequence with PF2D

Figure 53 – Continued

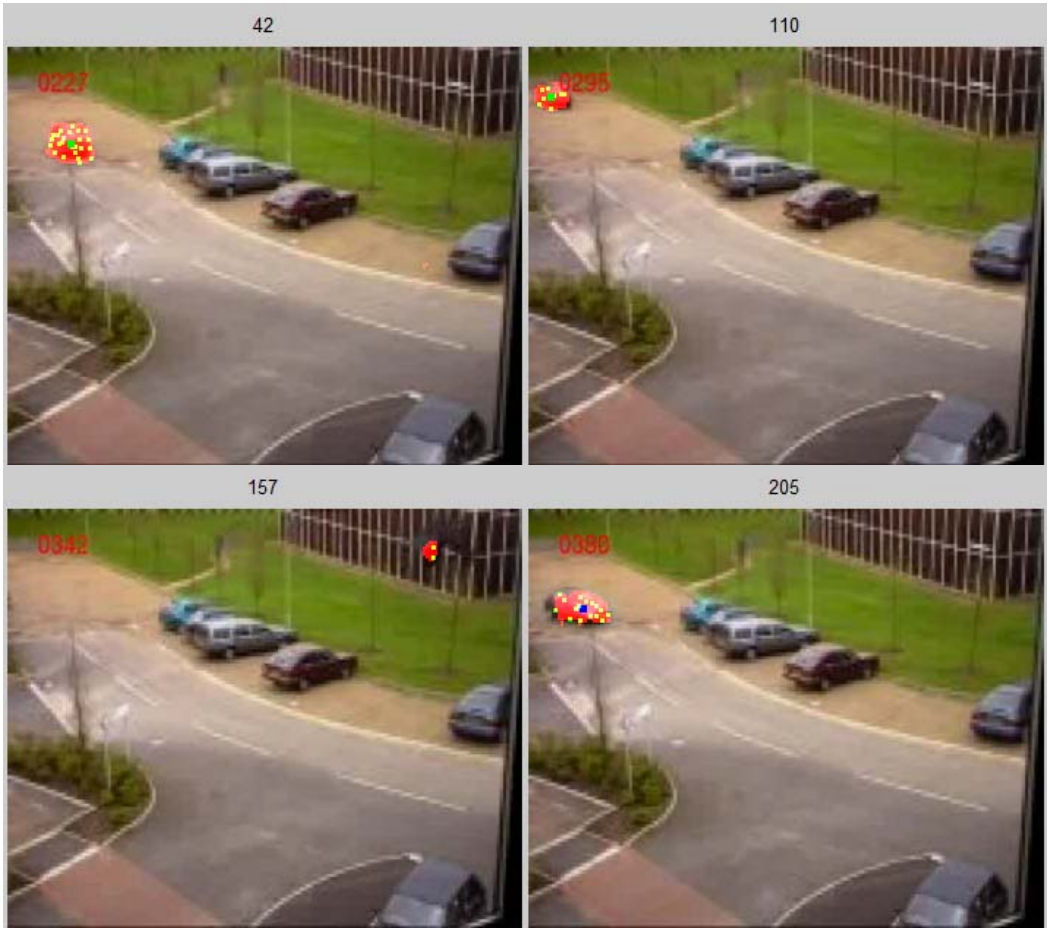

Figure 54 – Tracking results of PETS 2000 sequence with PF2D-OF

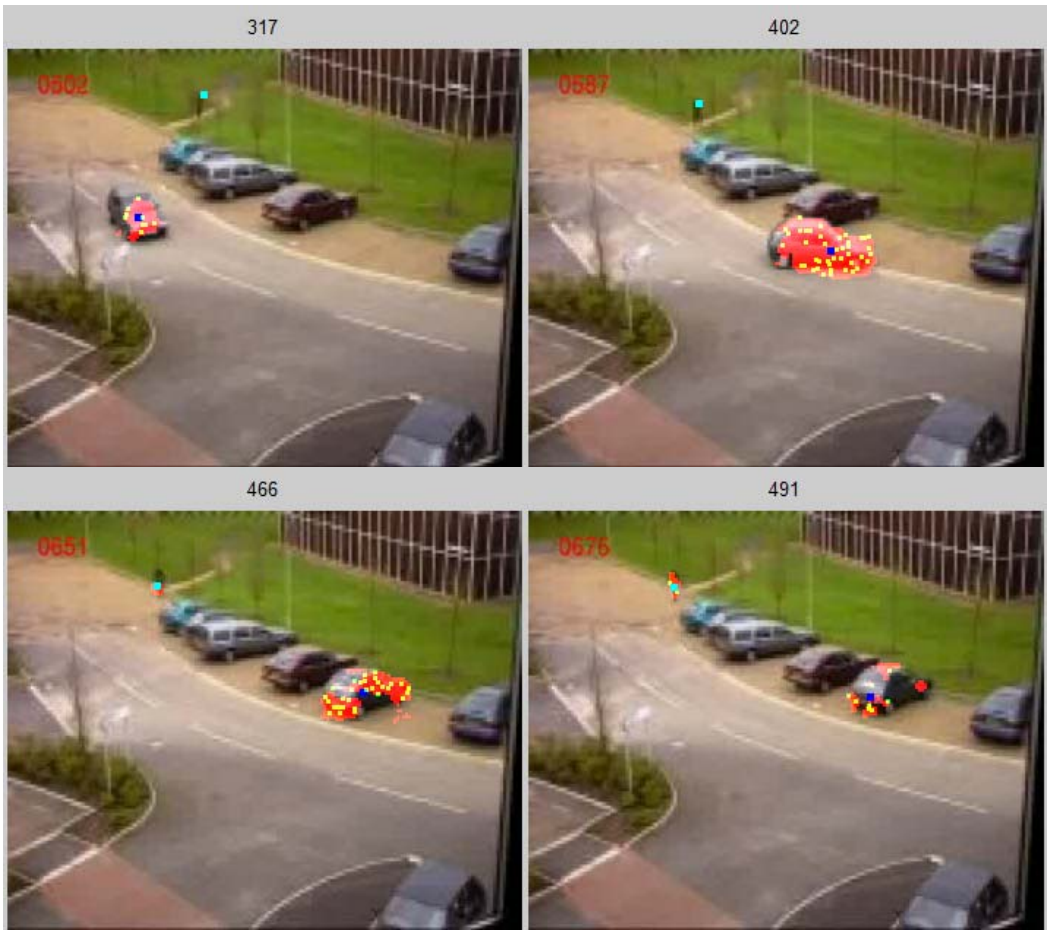

Figure 54 – Continued

Figure 55 – Tracking results of PETS 2000 sequence with PF4D

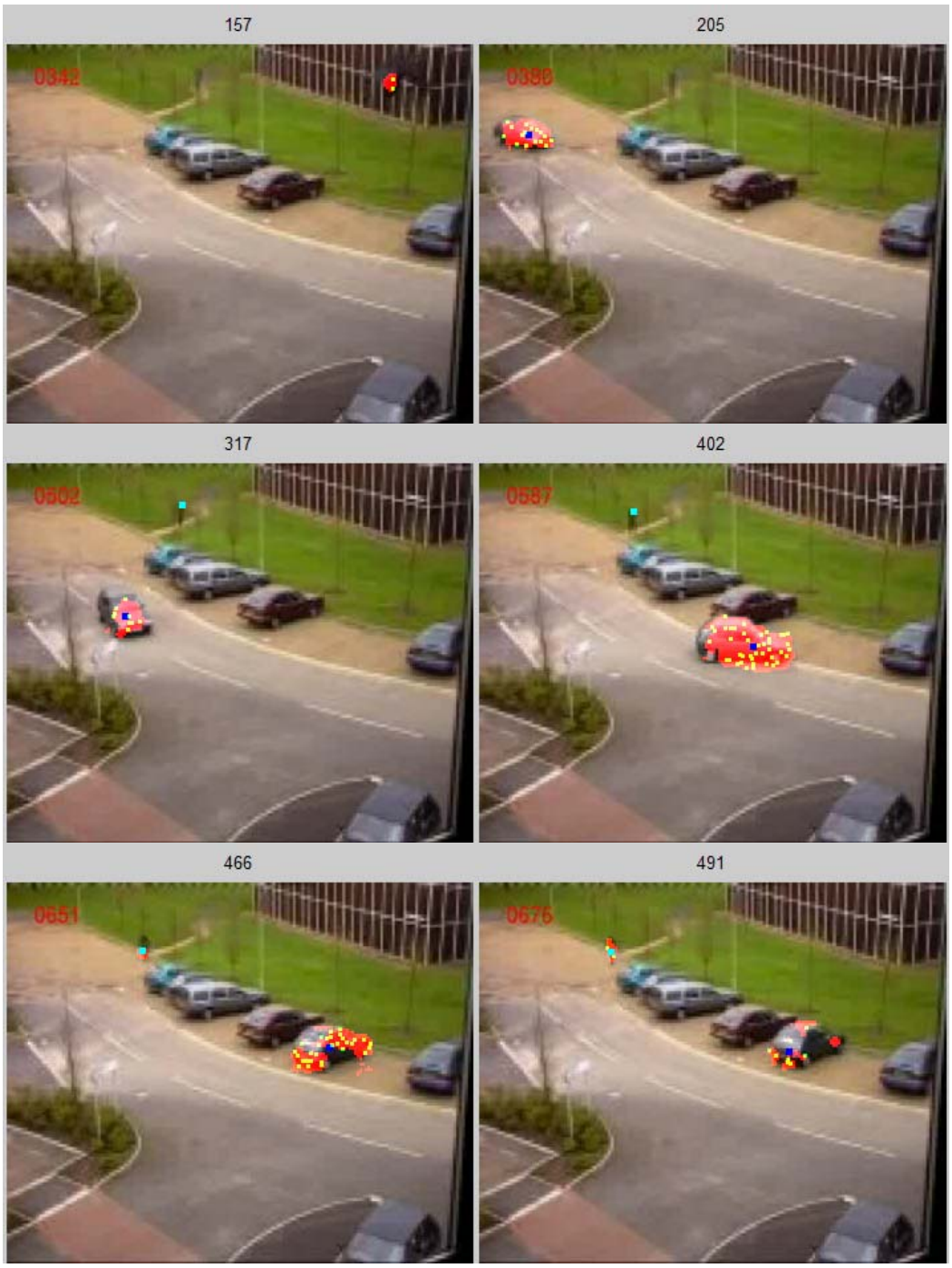

Figure 55 – Continued

For PETS 2000 sequence, all three particle filter based trackers show satisfactory results. However, PF2D slightly suffers while tracking the pedestrian. The tracking performance of PF2D-OF and PF4D are almost identical throughout the PETS 2000 sequence. However, if the results are carefully analyzed, it can be seen that PF4D can track the pedestrian better than PF2D-OF does. Therefore, PF4D gives the best

tracking results for PETS 2000 sequence. The tracking results for PETS 2001 sequence are introduced in Figure 56, 57 and 58.

Figure 56 – Tracking results of PETS 2001 sequence with PF2D

Figure 56 – Continued

Figure 57 – Tracking results of PETS 2001 sequence with PF2D-OF

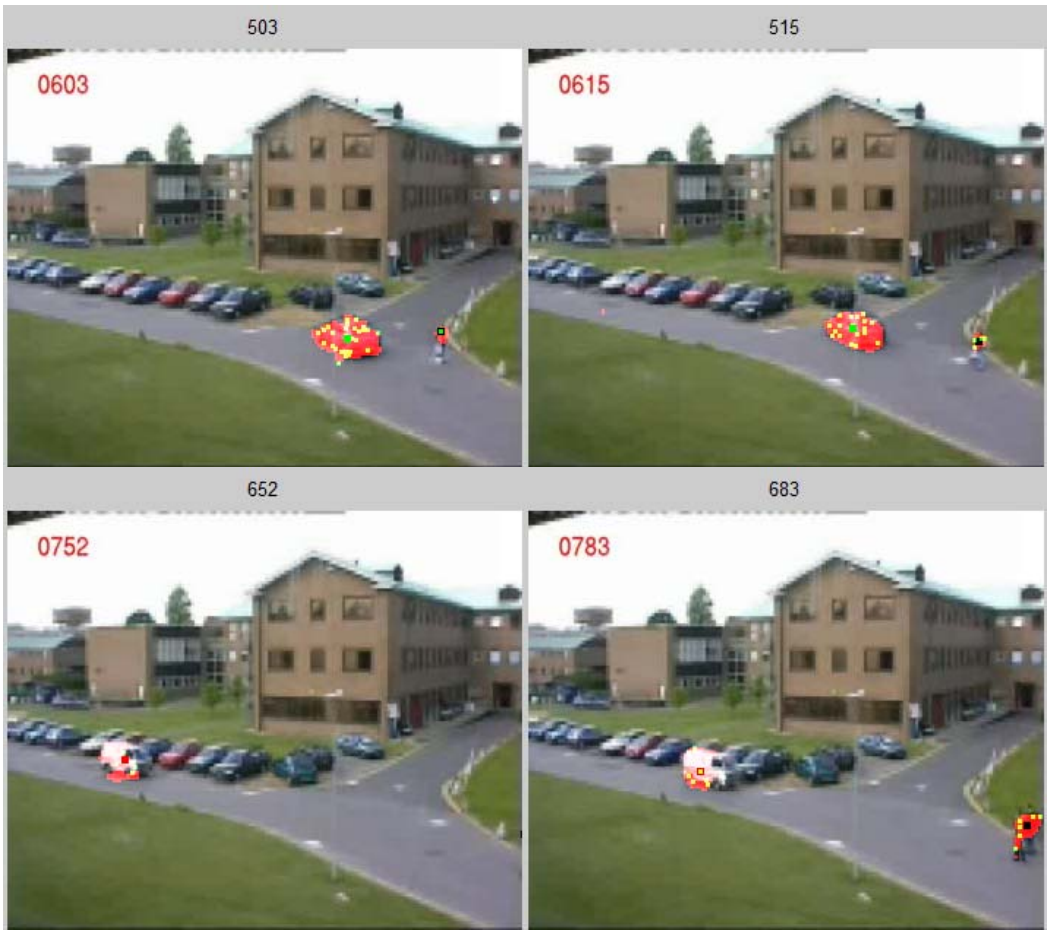

Figure 57 – Continued

Figure 58 – Tracking results of PETS 2001 sequence with PF4D

Figure 58 – Continued

If the results are analyzed carefully, it can be easily observed that PF2D-OF and PF4D show a much better performance throughout the sequence comparing with PF2D. The tracking results of PF2D-OF and PF4D are comparable again for PETS 2001 sequence. There is a minor performance difference between these two particle filters. However, PF2D-OF has slightly better tracking results with respect to PF4D. Therefore, PF2D-OF shows the best tracking performance for PETS 2001 sequence.

5.3.6 Quantitative Analysis of Trackers

To compare the performance of three Kalman filter based trackers, MSE for these Kalman filters is calculated for each dataset. Ground truth data for each dataset is constituted manually by selecting the position of each object in every four frames. The results are given in Table 6.

Table 6 – The MSE results for Kalman filter tracking

MSE	St. George	PETS 2000	PETS 2001
KF2D	424.5	266.9	380.8
KF2D-OF	388.4	128.6	288.6
KF4D	258.7	122.2	310.4

Similarly, to compare the performance of three particle filter based trackers, MSE for these particle filters is calculated for each dataset. Similarly, ground truth data for each dataset is constituted manually by selecting the position of each object in every four frames. The results are given in Table 7.

Table 7 – The MSE results for particle filter tracking

MSE	St. George	PETS 2000	PETS 2001
PF2D	793.0	126.6	431.6
PF2D-OF	662.2	119.6	303.4
PF4D	667.5	111.3	307.1

5.3.7 Summary

In this chapter, firstly the general framework of the object tracking system is expressed and the connections between the main parts of this thesis are clarified. For the feature clustering step, the differences between two approaches; *K*-means

and mean shift algorithm are stated. Then, mean shift algorithm is explained in detail and the clustering results are shown for two inappropriate bandwidths. Three datasets, which are used for this thesis work, are introduced and the difficulties of these datasets are also stated.

After that, the experimental results on moving object detection, feature extraction, clustering, Kalman filter tracking and particle filter tracking are given for each dataset. The tracking results show that two dimensional Kalman filter KF2D gives the worst tracking performance among the three Kalman filters. Two dimensional Kalman filter with optical flow data KF2D-OF and four dimensional Kalman filter KF4D show a comparable tracking performance and there are slightly differences between the results of these Kalman filters. For St. George and PETS 2000 sequences, KF4D gives the best results and for PETS 2001 sequence, KF2D-OF gives the best results.

Particle filter based tracking results are similar to Kalman filter based tracking results. The tracking results show that two dimensional particle filter PF2D gives the worst tracking performance for all datasets. Two dimensional particle filter with optical flow data PF2D-OF and four dimensional particle filter PF4D are again comparable and there are slightly differences between the results of these particle filters. For St. George and PETS 2001 sequences, PF2D-OF gives the best results and for PETS 2000 sequence, PF4D gives the best results.

To conclude, the results show that, using optical flow data as a correction term in Kalman filter and particle filter clearly improves the tracking performance of two dimensional Kalman filter or two dimensional particle filter and reaches them comparable with four dimensional Kalman or particle filter tracking performance. Therefore, the positive effects of using optical flow as a correction term in Kalman and particle filters are clearly shown in these experiments.

CHAPTER 6

CONCLUSIONS

In this thesis work, two different disciplines; computed vision based algorithms and estimation theory based algorithms, are evaluated together for tracking problems. The main objective is to investigate the effects to tracking performance in the case of using these algorithms interactively. Specifically the optical flow term is added to the particle filter tracker as a correction term and the tracking results are checked, whether there is an improvement or not. In order to constitute an object tracking system three major chapters are devoted to moving object detection, optical flow and feature extraction and particle filter. In the previous chapter, the connections between these chapters are also detailed, along with many experimental results.

6.1 Summary of the Thesis

For moving object detection, three different approaches are implemented. The conventional temporal median estimator is simply based on subtracting the current frame from the temporal median of the sequence. The second approach is background modeling using nonparametric KDE. To decide whether a pixel in the current frame belongs to the background or foreground, KDE is implemented to the sample frames. For KDE, a Gaussian kernel and an adaptive bandwidth are chosen to use. The third approach is a Bayesian approach for object detection. This model utilizes the useful correlation in the intensities of neighboring pixels and assumes that true foreground objects keep their colors and spatial positions in time. It also models the background and the foreground individually and obtains the object detection by using a likelihood ratio classifier. These three approaches are

compared by using a test video, which has nominal camera motion. According to simulation results, Bayesian approach for object detection method is certainly superior against other two methods in terms of detection and false alarm rates.

For calculating the optical flow, an improved version of KLT is employed. Unlike the conventional optical flow calculation techniques, KLT includes the image pyramid approach. Using the image pyramid approach provides the algorithm not to violate Taylor series expansion assumption, therefore KLT can also handle relatively fast objects in the scene. In addition, optical flow vectors are calculated iteratively and in sub-pixel accuracy, therefore KLT is more reliable with compared to the conventional optical flow calculation methods. For feature extraction, an algorithm called "Shi-Tomasi corner detection" is used. This algorithm uses the eigenvalues of spatial gradient matrix in order to decide the feature points. According to simulation results, optical flow calculation and feature extraction work well enough for object tracking system that is constituted in this thesis work.

The particle filter is considered for the main component of the object tracking system. For this reason, particle filter is studied in detail, the problems of particle filter and the solutions for them are clarified. A generic particle filter is implemented and its performance is tested against a Kalman filter, which gives optimal results for linear dynamic systems. With appropriate number of particles, the particle filter that is developed shows almost identical performance to optimal Kalman filter.

In order to obtain a robust object tracking system, data association and feature clustering problems should be solved. For data association problem, some rules about target initiation, data association and losing / re-finding the objects are put and followed. To solve the feature clustering problem, *K*-means and mean shift algorithms are employed. Both of these algorithms are implemented and the experiments show that mean shift algorithm is more suitable for this thesis work since it does not require an initialization and has only one parameter; Parzen window bandwidth. The importance of bandwidth selection is visualized by implementations with inappropriate bandwidths.

Optical flow calculation and feature extraction are implemented only moving regions in the image that reduces computation time significantly. Tracking process can be also done without optical flow information however the features are crucial for particle filter since they constitute the measurements. To improve the tracking performance, the idea of using the optical flow data as a correction term in particle filter is also expressed. Before particle filter based tracking, the idea is tested with Kalman filter based tracking. All the implementation details are clarified for Kalman filter tracking and particle filter based tracking.

6.2 Conclusions

Experiments are conducted by using three different challenging datasets that were used in other object tracking papers in the literature. All the studies that are mentioned in this thesis work are applied to these datasets step by step and all the results are presented. Although the scene in datasets has difficulties such as having a moving background such as tree branches, occlusion between the objects and between the object and the background, small intensity difference between the objects and the background and relatively small objects due to the distance between the objects and the camera, the results on moving object detection, feature extraction and clustering are quite satisfactory and good enough to use in particle filter tracking.

Since PF4D shows better performance for all the datasets with compared to PF2D, it is observed that including the velocity information to state vector is necessary for a robust particle filter tracker. Experimental results also show that using optical flow information clearly improves the tracking performance. As compared with PF2D, PF2D-OF gives more reliable results in general. Additionally, the optical flow data also helps the algorithm to cope with the occlusions in the scene. Although PF2D-OF does not use the velocity information, the optical flow information keeps the performance of PF2D-OF at a level that is comparable with PF4D.

The results of Kalman filter based tracking are quite similar with particle filter based tracking. KF2D performs the worst tracking results with compared to KF2D-OF and KF4D. Similar to particle filter based tracking results, the performance of KF2D-OF and KF4D are comparable again. Therefore, optical flow information improves the general tracking performance and helps the tracker to handle the occlusion.

To conclude, in this thesis work, computed vision based algorithms are utilized together with estimation theory based algorithms in order to have a robust object tracking system. The experimental results show that, using optical flow information clearly improves the tracking performance. Although computing the optical flow vectors is computationally expensive, the idea of limiting the optiacal flow computation on the foreground objects significantly decreases the computational cost. This vision-assisted object tracking idea can be used for any object tracking systems like surveillance systems.

REFERENCES

[1] A. Elgammal, D. Harwood, and L. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance," in Proceedings of IEEE, 2002.

[2] Y. Sheikh and M. Shah, "Bayesian modeling of dynamic scenes for object detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005.

[3] C. Stauffer and W. Grimson, "Learning patterns of activity using real-time tracking," in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.

[4] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of human body," IEEE Trans. Pattern Anal. Machine Intell., vol. 19, pp. 780-785, July 1997.

[5] K. P. Karmann, A. Brandt, and R. Gerl, "Using adaptive tracking to classify and monitor activities in a site," In Time Varying Image Processing and Moving Object Recognition, Elsevier Science Publishers, 1990.

[6] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "Towards robust automatic traffic scene analysis in real-time," in International Conference of Pattern Recognition, 1994.

[7] A. Yilmaz, O. Javed, M. Shah, "Object tracking: A survey," ACM Computing Surveys, vol. 38, 2006.

[8] N. Friedman and S. Russell, "Image segmentation in video sequences: A probabilistic approach," presented at the13th Conf. Uncertainty in Artificial Intelligence, Providence, RI, 1997.

[9] W. E. L. Grimson and C. Stauffer, "Adaptive background mixture models for real-time tracking," in Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol. 1, 1999, pp. 22–29.

[10] I. Haritaoglu, D. Harwood, and L. S. Davis, "W^4: Real-time surveillance of people and their activities," IEEE Trans. Pattern Anal. Machine Intell., vol. 22, pp. 809-830, Aug. 2000.

[11] J. Rittscher, J. Kato, S. Joga, and A. Blake, "A probabilistic background model for tracking," in Proc. 6^{th} Eur. Conf. Computer Vision, vol. 2, 2000, pp. 336–350.

[12] B. Stenger, V. Ramesh, N. Paragios, F. Coetzee, and J. Bouhman, "Topology free hidden markov models: Application to background modeling," in Proc. IEEE Int. Conf. Computer Vision, 2001, pp. 294–301.

[13] Y. H. Yang and M. D. Levine, "The background primal sketch: An approach for tracking moving objects," Machine Vision Appl., vol. 5, pp. 17–34, 1992.

[14] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practice of background maintenance," in IEEE Proceedings of the International Conference on Computer Vision, 1999.

[15] N. Oliver, B. Rosario, and A. Pentland, "A Bayesian computer vision system for modeling human interactions," in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.
[16] A. Monnet, A. Mittal, N. Paragios, and V. Ramesh, "Background modeling and subtraction of dynamic scenes," in IEEE Proceedings of the International Conference on Computer Vision, 2003.
[17] J. Zhong and S. Sclaroff, "Segmenting foreground objects from a dynamic textured background via a robust kalman filter," in IEEE Proceedings of the International Conference on Computer Vision, 2003.
[18] Y. Hsu, H. H. Nagel, and G. Rekers, "New likelihood test methods for change detection in image sequences," Computer Vision Image Process., vol. 26, pp. 73–106, 1984.
[19] T. Matsuyama, T. Ohya, and H. Habe, "Background subtraction for nonstationary scenes," in Proc. 4th Asian Conf. Computer Vision, 2000, pp. 662–667.
[20] E. Parzen, "On estimation of a probability density function and mode," Annals of Mathematical Statistics, vol. 33, pp. 1065-1076, 1962.
[21] D. Comaniciu, "An algorithm for data-driven bandwidth selection," IEEE Trans. Pattern Analysis Machine Intelligence, vol. 25, pp. 281-288, 2003.
[22] M. D. Levine, Vision in Man and Machine, New York: McGraw Hill, 1985.
[23] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt and J. M. Ogden, "Pyramid methods in image processing," RCA Engineer, pp. 33-41, November 1984.
[24] B. Horn, B. Schunk, "Determining optical flow," Artificial Intelligence, pp. 185-203, 1981.
[25] J. Barron and N. Thacker, "Tutorial: Computing 2D and 3D optical flow," Tina Memo, no: 2004-12.
[26] S. Birchfield, "Derivation of Kanade-Lucas-Tomasi tracking equation," January 1997.
[27] J. Bouguet, "Pyramidal implementation of the Lucas Kanade Feature Tracker description of the algorithm," Intel Corporation Microprocessor Research Labs.
[28] J. Shi, C. Tomasi, "Good features to track," IEEE Conf. Computer Vision and Pattern Recognition, June 1994.
[29] M. A. Hall, "Correlation-based feature selection for machine learning," PhD thesis for the University of Waikato, April 1999.
[30] M. S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, "A tutorial on particle filters for online nonlinear / non-Gaussian Bayesian tracking," IEEE Transactions on Signal Processing, vol. 50, no. 2, February 2002.
[31] M. Johannes, N. Polson, "Particle filtering," October 2006.
[32] B. Ristic, S. Arulampalam, N. Gordon, "Beyond the Kalman filter: Particle filters for tracking applications," Artech House, January 2004.
[33] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson and P. Nordlund, "Particle filters for positioning, navigation and tracking," IEEE Transactions on Signal Processing, 2002.
[34] D. Salmond and N. Gordon, "An introduction to particle filters," September 2005.

[35] http://www.cs.berkeley.edu/~flw/tracker (Last access date: 27.01.2012)
[36] R.Szeliski, "Computer Vision Algorithms and Applications," Springer Verlag, 2011.
[37] J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observation," Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281-297, 1967.
[38] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu, "An efficient k-means Clustering Algorithm: Analysis and Implementation," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, no: 7, July 2002.
[39] R. O. Duda, P. E. Hart, D. G. Stork, "Pattern Classification," A Wiley Interscience, 2001.
[40] S. K. Zhou, R. Chellappa and B. Moghaddam, "Visual tracking and recognition using appearance – adaptive models in particle filters," IEEE Transactions on Image Processing, Vol. 13, no: 11, November 2004.
[41] A. Mittal and N. Paragios, "Motion-based background subtraction using Adaptive Kernel Density Estimation," 2003.
[42] D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," Proceedings of Imaging Understanding Workshop, pp. 121-130, 1981.

APPENDIX A

MAIN ALGORITHM OF KLT

The method of using KLT algoritms in this study is given in Chapter 3. In Table 8 detailed information of KLT algorithm is provided.

Table 8 –Detailed explanation of KLT algorithm [27]

Assume that u be a point on image I. Find its corresponding location v on image J;

Build pyramid representations of I and J: $\{I^L\}_{L=0,\ldots,L_m}$ *and* $\{J^L\}_{L=0,\ldots,L_m}$

Initialize the pyramidal guess: $g^{L_m} = \left[g_x^{L_m}\ g_y^{L_m}\right]^T = [0\ 0]^T$

FOR $L = L_m$ down to 0 with the step of -1

Location of point u on image I^L: $u^L = \left[p_x\ p_y\right]^T = u/2^L$

Derivative of I^L with respect to x: $I_x(x,y) = \dfrac{I^L(x+1,y) - I^L(x-1,y)}{2}$

Derivative of I^L with respect to y: $I_y(x,y) = \dfrac{I^L(x,y+1) - I^L(x,y-1)}{2}$

Spatial gradient matrix: $G = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)\,I_y(x,y) \\ I_x(x,y)\,I_y(x,y) & I_y^2(x,y) \end{bmatrix}$

FOR k = 1 to K with the step of 1 (or until a threshold level)

Image difference: $\delta I_k(x,y) = I^L(x,y) - J^L(x + g_x^L + v_x^{k-1}, y + g_y^L + v_y^{k-1})$

Image mismatch vector: $\overline{b_k} = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \left(\begin{bmatrix} \delta I_k(x,y) I_x(x,y) \\ \delta I_k(x,y) I_y(x,y) \end{bmatrix} \right)$

Residual pixel motion: $\eta^{-k} = G^{-1}\overline{b_k}$

Guess for next iteration: $\overline{v}^{-k} = \overline{v}^{-k-1} + \overline{\eta}^{k}$

END of the for loop on *k*

Final optical flow at level *L*: $d^{L} = \overline{v}^{K}$

Guess for next level *L*-1: $g^{L-1} = \left[g_x^{L-1}\ g_y^{L-1}\right]^T = 2(g^{L} + d^{L})$

END of the for loop on *L*

Final optical flow vector: $d = g^{0} + d^{0}$

Location of point on *J*: $v = u + d$