



UNIVERSITEIT • STELLENBOSCH • UNIVERSITY

An Adaptive Feature-based Tracking System

by

Eugene Pretorius

*Thesis presented at the University of Stellenbosch
in partial fulfilment of the requirements for the
degree of*

Master of Science in Applied Mathematics

Department of Mathematical Sciences
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Supervisors:

Prof B.M. Herbst Dr K.M Hunter

2008

Copyright © 2008 University of Stellenbosch
All rights reserved.

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature:

E. Pretorius

Date:

Abstract

An Adaptive Feature-based Tracking System

E. Pretorius

Department of Mathematical Sciences

University of Stellenbosch

Private Bag X1, 7602 Matieland, South Africa

Thesis: MSc (Applied Mathematics)

2008

In this paper, tracking tools are developed based on object features to robustly track the object using particle filtering. Automatic on-line initialisation techniques use motion detection and dynamic background modelling to extract features of moving objects. Automatically adapting the feature models during tracking is implemented and tested.

Opsomming

In hierdie thesis word video volgings gereedskap ontwikkel en getoets. Deur gebruik te maak van 'n voorwerp se kenmerke is dit moontlik om sodoende 'n voorwerp robuust te kan volg deur "Particle Filtering" tegnieke. Die stelsel word automaties geïnisialiseer met beweging deteksie en agtergrond modellering om voorwerpe se kenmerke te identifiseer en te ontrek. Automatiese opdatering van die kenmerk modelle gedurende video volging word geïmplementeer en getoets.

Acknowledgements

Special thanks to my supervisors, Professor Ben Herbst and Dr. Karin Hunter for their insights and guidance. And to my family, friends and girl friend for their support and understanding.

Contents

Declaration	ii
Abstract	iii
Opsomming	iv
Acknowledgements	v
Contents	vi
List of Figures	ix
List of Abbreviations	x
List of Symbols	xi
1 Introduction	1
1.1 Problem statement: Tracking	1
1.2 Literature study	2
1.3 Objective of the study	4
1.4 Dissertation structure	5
2 Particle filter theory	7
2.1 Introduction to Bayesian estimation	7
2.2 Dynamic system representation	8
2.3 Bayesian filter	9
2.4 Monte Carlo (MC) integration	10

2.5	Particle filter algorithm	12
2.6	Summary	15
3	Feature vectors	18
3.1	Particles and Features	18
3.2	Colour-based feature	19
3.3	Histogram of oriented gradients feature	22
3.4	Motion model	30
3.5	Feature adaptivity	32
3.6	Finding an object and detecting a loss	34
3.7	Perspective adaption	35
3.8	Algorithm	36
3.9	Feature tracking experiment	37
3.10	Summary	41
4	System implementation	42
4.1	System design and goals	42
4.2	Background modelling	44
4.3	Motion tracking	48
4.4	User defined parameters	53
4.5	Object detection experimental results	54
4.6	Particle filter tracking	57
4.7	Automatic initialisation tracking	60
4.8	Summary and conclusion	63
5	Conclusion	68
5.1	Future challenges	69
5.2	Restrictions	69
5.3	Goals achieved	70
5.4	Recommendations	70
A	Project files	72
A.1	Required software	72
A.2	Installing tracking tools	73

<i>Contents</i>	viii
A.3 Running an example	73
A.4 Directory structure and file description	73
A.5 Hardware configuration used	76
References	77

List of Figures

2.1	Resampling	16
2.2	Particle filter iteration	17
3.1	Colour weighting function	20
3.2	Colour feature example	21
3.3	Example HOG descriptor	25
3.4	Example HOG images	26
3.5	HOG feature vectors at different cell sizes	27
3.6	Model images used to obtain similarity results of Figure 3.7. . .	30
3.7	HOG features comparison	31
3.8	Simulated tracking example using colour and HOG features . .	38
3.9	Simulated tracking example: HOG adaption	40
4.1	Feature-based tracking modules	43
4.2	System module design	44
4.3	Motion tracking	52
4.4	Background modelling	55
4.5	Colour feature tracking of soccer players	58
4.6	Face image used to initialise tracking	59
4.7	HOG feature tracking of face	65
4.8	Tracking face using combined features	66
4.9	Tracking snooker balls	67
A.1	GUI main window	74
A.2	Project files	74

List of Abbreviations

EM	Expectation Maximisation
FGD	Foreground object detection
HOG	Histogram of gradients
HSV	Hue Saturation Value
KLT	Kanada-Lucas-Tomasi
MC	Monte Carlo
MOG	Mixture of Gaussian's
pdf	probability distribution function
RGB	Red Green Blue
SIR	Sequential importance resampling
SIS	Sequential importance sampling
SMC	Sequential Monte Carlo

List of Symbols

ρ	Bhattacharyya coefficient
\mathbf{f}	Transition function
\mathbf{h}	Observation function
\mathbf{I}	Image
k	Time
\mathbf{M}	Motion Image
\mathbf{R}	Image region
\mathbf{s}	Particle samples
\mathbf{S}_σ	Samples set deviation
T	Threshold
\mathbf{v}	Speed
\mathbf{w}	Weight
\mathbf{x}	State
\mathbf{z}	Observation

Chapter 1

Introduction

Suppose a robot is given a video stream, and through it, interacts with the world around it. In this scenario, computer vision techniques would have to be programmed so that the robot can track, and maybe even recognise certain objects.

Object tracking comes naturally to humans, since it is instinctive to observe the world around us. Computers, however, need sophisticated techniques in order to mimic our tracking ability and to automate tedious tasks. These techniques attempt to solve object tracking through cluttered scenes with noisy measurements. Specific algorithms each have inherent shortcomings due to the nature of the problem, while successful approaches use object appearances that indeed mimic the way humans would track objects.

1.1 Problem statement: Tracking

Tracking of objects in a video sequence is one of the most fundamental problems in Computer Vision. It forms the basis of applications as diverse as surveillance, traffic monitoring, gesture recognition and sport analysis such as soccer.

Some of the most widely used tracking algorithms include the Kanada-Lucas-Tomasi (KLT) feature tracker which is an optical flow method. These

algorithms track objects by comparing consecutive pairs of frames, no dynamic information about the moving object is used. The problem is that as soon as the motion of the object itself is used, some information about the object is required. On the other hand using dynamic information leads to more robust tracking algorithms, allowing, for example, tracking through occlusions. It turns out that it is not hard to incorporate dynamic information into the tracking algorithms by using a particle filter. That leaves finding information about the object itself that is to be tracked.

There are several choices to obtain information about the object. Isard and Blake [13] track the shape of an object, allowing, but also restricting shape deformations. This is known as active contours. Another choice is to track features that describe the colour or texture of the objects. Then it is possible to combine all these which resulted in the so-called Active Appearance Models (AAM) [3]. Although robust, AAM's are computationally expensive algorithms. A simple fact is that clients requiring systems based on computer vision techniques, such as surveillance, often cannot afford the necessary CPU power. It is therefore of considerable interest to explore light-weight alternatives. That takes us back to colour and texture tracking using a particle filter.

1.2 Literature study

A study of the most recent developments in tracking has shown that colour-based particle filtering is used successfully to track non-rigid objects [14]. A colour distribution model is built in RGB space and a similarity measure is employed for the object model. The authors compare this technique with the mean-shift algorithm which tries to minimise the distance between the theoretical mean and the observed ones. It is shown that the mean-shift algorithm fails when the object's position in successive frames does not overlap, where the particle filter has no such problems. The colour-based particle filter would however fail if lighting conditions change to an

extent where the similarity between the object model and measurements is indistinguishable from the background.

A self-adapting histogram model is used in [6] to adjust to lighting changes. This is possible since the colour model uses the HSV instead of the RGB colour space. The adaptivity allows for small illumination changes as well as partial rotation in 3D-space. A confidence measure is calculated from the probability distribution that describes how well objects are being tracked. Adapting is done using this confidence measure to only adapt to the actual target when confidence is high. Also, the implementation is done on a smart camera (camera with CPU) and runs real-time. Since all the processing is done on the camera itself, no images need to be sent over the network. This is a very important property in security applications where client privacy might be an issue.

Blob tracking [7] is also a successful feature-based tracker. A multi-resolution graph for tracked regions is built from connected components (blobs). The point is made by the authors that robust tracking cannot be handled by only one algorithm. Modules need to be built up that solve problems robustly at each step of a semantic ladder. The first step being segmentation, and the next step tracking. The algorithm handles larger slow moving blobs that are easy to track, and fast moving, small blobs that are much more difficult to track equally well. In cases where the algorithm failed the segmentation step produced unsatisfactory results. Either a region of interest is not segmented, two separate regions merge and form one blob or the relationship between a blob in consecutive frames has a low likelihood at a low resolution in the multi-resolution scale. This is handled at the next semantic step.

Multi-camera systems are implemented in [21] and [20] for tracking football players and surveillance purpose, respectively. Cameras with overlapping fields of view are used. In the case of the football players, each camera's processing is done separately and then combined. A Kalman fil-

ter is used for each camera to track players. Measurement data is used whenever available to minimize estimation errors. For the surveillance application the Kanada-Lucas-Tomasi (KLT)[18] feature tracking algorithm is used. KLT tries to estimate the motion at every pixel position using concurrent available frames.

Contour features are used in [13] implementing the condensation (particle filter) algorithm. Spline curves are fitted to an object's shape and high contrast features are extracted at intervals along the curve. Object contours (splines) are descriptive features and are successfully used to track curves through clutter. This is known as contour-based tracking. An impressive experiment is done tracking a falling leaf against a background filled with similar leaves. Contour-based tracking has the disadvantage of being computationally expensive.

In [4] edges and the pixel gradients are considered as feature models. Images are broken into cells, each a histogram of oriented gradients (HOG). Combined these cells represent the feature model. This approach has been successfully implemented in object recognition type problems. This technique suffers from expensive calculations and slow execution on less simplistic scene composition.

Some of the most popular tracking algorithms where shown here. An important factor for each of these algorithms is their computational cost. For any tracker to be useful it should be robust and light-weight and should be cheap to build and use.

1.3 Objective of the study

In this thesis several light-weight trackers are studied. More specifically implementing of a colour-based tracker and a texture-based feature using an adapted HOG descriptor is developed. The tracking "engine", a particle filter, is implemented. The project objectives are

- Design a tracking implementation to solve problem statement
- Build a particle filter for the design
- Create a colour-based feature model
- Investigate and implement other type feature models such as texture
- Test the feature's effectiveness and robustness
- Make improvements to the original implementation based on learnt shortcomings
- Automate the tracking process after initialisation
- Automatic initialisation of the tracker
- Feature adaption when object's appearance change

HOG was developed to detect objects in a scene at different image scales. Its success, when used as an object detector, sparked our interest for use in a tracking context. An adaption to the HOG texture feature is developed and combined with the colour feature to improve tracking robustness. The HOG feature is used to find a similarity measure between the target object and samples. Failure as a robust tracking feature is discovered and adjustments to the HOG construction are developed. The developed feature descriptor can successfully track objects using only texture information, (when texture is available) and tracking improves when combined with a colour feature.

1.4 Dissertation structure

This thesis builds on the theory in Chapter 2 to a working implementation in Chapter 4.

After first covering the basic particle filter concepts, object features are investigated in Chapter 3. Features are adapted through new observations and the process is automated to adapt independent of user interaction. Features with complementary characteristics, that contain sufficient information, are investigated.

A motion tracker is built to help with automating a tracking system at initialisation. Background modelling techniques are also investigated as part of the tracker initialisation and feature extraction. Each system module is discussed as implemented and results are shown.

Chapter 2

Particle filter theory

A particle filter is a non-linear sub-optimal model estimation technique based on simulation. It is an implementation of the formal recursive Bayesian filter that performs sequential Monte Carlo (SMC) estimation based on a weighted representation of probability densities [16]. Random sampled approximations of the probability density function (pdf) are called the weighted particles. In general more particles lead to a better approximations of the pdf. Particle filters propagate a finite number of these samples according to the dynamics of the system and update the pdf using the observed measurements [1].

2.1 Introduction to Bayesian estimation

The Bayesian approach aims to construct the posterior pdf based on all available previous information and current measurements. In such a case where the pdf is constructed from all available information the solution is complete and an optimal estimate (in a minimising-of-a-cost-function sense) of the state is possible. A recursive approach is considered that allows for a new estimate whenever new measurements are obtained. Recursively, predictions and updates form the two main steps for most Bayesian estimators. The prediction step propagates the state pdf forward according to a dynamic system model. The update step, using Bayes' theorem, uses the

latest measurements to calculate the prediction pdf. The recursive Bayesian estimation or filter therefore provides a formal mechanism for propagating and updating the posterior pdf as new information is received [17].

The following sections develop the background theory of particle filtering. Firstly, a dynamic system is represented by a dynamics model and a measurement model in a probabilistic form so that a Bayesian approach may be adopted. Then the recursive estimation in Bayesian filtering [5], prediction and update steps, fits this dynamic representation. Integration difficulties in Bayesian filtering are handled by Monte Carlo (MC) estimation [16] presented in Section 2.4. Finally, the implemented particle filtering algorithm is discussed.

2.2 Dynamic system representation

A sequence of evolving probability distributions $\pi(\mathbf{x}_k)$, indexed by discrete time $k = 0, 1, 2, \dots$, is called a probabilistic dynamic system [12]. A dynamic system is generally represented by a state space \mathbf{x}_k . Two models are required for analysis in a dynamic system: a dynamic model and a measurement model.

Firstly, a dynamic model describing system evolution, the change in the state over time, is defined. The state sequence is a Markov random process and the state equation is written as

$$\mathbf{x}_k = f_{k-1}(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}), \quad (2.2.1)$$

where \mathbf{x}_k is the state vector at time step k , f_{k-1} is the (possibly non-linear) state transition function that propagates the system from time step $k-1$ to time step k . Process noise is modelled by \mathbf{v}_k and the pdf is assumed known. Secondly, a measurement model where noisy measurements are related to

the state is needed. The observation equation is of the form

$$\mathbf{z}_k = h_k(\mathbf{x}_k, \mathbf{w}_k), \quad (2.2.2)$$

where \mathbf{z}_k is the observation vector at time step k , h_k is the observation function that relates the state space to the observations and the observation noise, \mathbf{w}_k , which has a known pdf.

The state and observation equations can also be represented by probability densities. Note that (2.2.1) is a first order Markov process and that the state equation is equivalent to $p(\mathbf{x}_k|\mathbf{x}_{k-1})$, also known as the transition density. Similarly, the observation equation (2.2.2) is equivalent to $p(\mathbf{z}_k|\mathbf{x}_k)$.

In summary, the probabilistic description of a dynamical system formulated in a probabilistic way fits the Bayesian estimation approach, as described in the next section.

2.3 Bayesian filter

Bayesian filtering attempts to construct the posterior pdf from all available information. The state vector, \mathbf{x}_k , contains information describing the system. This true state, \mathbf{x}_k , is assumed to be a Markov process which cannot be observed directly, and the measurements \mathbf{z}_i , where the set $\mathbf{Z}_k = \{\mathbf{z}_i, i = 1, \dots, k\}$ are the observations of the state. A Markov assumption is made about the state space, that assumes the current state is only dependent on the immediately preceding state,

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}) = p(\mathbf{x}_k|\mathbf{x}_0, \dots, \mathbf{x}_{k-1}). \quad (2.3.1)$$

Similarly, the measurement at the k -th time step depend only on the current state and is independent of all other states given the current state

$$p(\mathbf{z}_k|\mathbf{x}_k) = p(\mathbf{z}_k|\mathbf{x}_0, \dots, \mathbf{x}_{k-1}). \quad (2.3.2)$$

From the Markov assumption made, the formulation of (2.3.1), (2.3.2) is equivalent to the dynamic system state representation in Section 2.2.

Given the posterior pdf at time $k - 1$, $p(\mathbf{x}_{k-1}|\mathbf{Z}_{k-1})$, the idea is to find $p(\mathbf{x}_k|\mathbf{Z}_k)$. This is achieved by means of a prediction and an update step. First, $p(\mathbf{x}_k|\mathbf{Z}_{k-1})$, the prior pdf, is obtained using the transition density $p(\mathbf{x}_k|\mathbf{x}_{k-1})$

$$p(\mathbf{x}_k|\mathbf{Z}_{k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{Z}_{k-1})d\mathbf{x}_{k-1}. \quad (2.3.3)$$

Now the new observation is obtained, this is used to update the posterior pdf using Bayes' rule by including the observation \mathbf{z}_k ,

$$p(\mathbf{x}_k|\mathbf{Z}_k) = p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{Z}_{k-1})/p(\mathbf{z}_k|\mathbf{Z}_{k-1}). \quad (2.3.4)$$

The normalization factor is given as usual by

$$p(\mathbf{z}_k|\mathbf{Z}_{k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{Z}_{k-1})d\mathbf{x}_k.$$

Bayesian filtering is defined by the prediction step in (2.3.3) and the update step in (2.3.4) with initial condition $p(\mathbf{x}_0|\mathbf{z}_0) = p(\mathbf{x}_0)$ obtained from assumed or given data.

Analytical evaluation of the pdf in (2.3.3) and (2.3.4) is impossible except in cases such as the Kalman filter and hidden finite-state space Markov chains where linearisation (Gaussian pdf's) simplifies the equations. Monte Carlo (MC) integration, on the other hand, is not limited by linear-Gaussian assumptions and will be described in the following section.

2.4 Monte Carlo (MC) integration

Monte Carlo (MC) integration methods use pseudo-random numbers to numerically approximate multi-dimensional, definite integrals and form the

basis of sequential monte carlo (SMC) methods. Pseudo-random numbers are generally used for computational convenience. By the Law of large numbers¹ if $N \rightarrow \infty$ then MC integration approaches the exact solution. MC integration is used to evaluate the integral (2.3.3) of the optimal Bayesian filter.

Consider a multi-dimensional, definite integral $g(\mathbf{x})$. Writing $g(\mathbf{x}) = f(\mathbf{x})\pi(\mathbf{x})$ its integral becomes

$$I = \int g(\mathbf{x})d\mathbf{x} = \int f(\mathbf{x})\pi(\mathbf{x})d\mathbf{x}. \quad (2.4.1)$$

The integral $g(\mathbf{x})$ is factorised such that $\pi(\mathbf{x})$ is a density. Since $\pi(\mathbf{x})$ is a density I is interpreted as the mean of $f(\mathbf{x})$. In a Bayesian context $\pi(\mathbf{x})$ is realised as the posterior pdf. Where $\{\mathbf{x}_i; i = 1, \dots, N\}$ are the samples drawn from $\pi(\mathbf{x})$. The MC estimate of I is the sample mean

$$I_N = \frac{1}{N} \sum_{i=1}^N f(\mathbf{x}^i) \quad (2.4.2)$$

and converges to I if N is chosen large enough. Unfortunately, effective sampling from $\pi(\mathbf{x})$ is not possible due to the distribution being multi-variate, non-Gaussian and only known up to a proportional constant. Importance sampling rather samples from a known density distribution $q(\mathbf{x})$ that approaches $\pi(\mathbf{x})$ when N is increased. This proposed pdf $q(\mathbf{x})$ is referred to as the importance or proposal pdf. Since $q(\mathbf{x})$ is a weighted density of the sample set, MC estimation is possible. The integral (2.4.1) is written as

$$I = \int f(\mathbf{x})\pi(\mathbf{x})d\mathbf{x} = \int f(\mathbf{x})\frac{\pi(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})d\mathbf{x}, \quad (2.4.3)$$

¹Jacob Bernoulli first described the law of large numbers as so simple that even the stupidest man instinctively knows it is true. -[http : //en.wikipedia.org/wiki/Law_of_large_numbers#note - 0](http://en.wikipedia.org/wiki/Law_of_large_numbers#note-0)

and the MC estimation is calculated, by drawing $N \gg 1$ samples, as

$$I_N = \frac{1}{N} \sum_{i=1}^N f(\mathbf{x}^i) w(\mathbf{x}^i) \quad (2.4.4)$$

where

$$w(\mathbf{x}) \propto \frac{\pi(\mathbf{x})}{q(\mathbf{x})} \quad (2.4.5)$$

are the normalised importance weights so that $\sum_{i=1}^N w^i = 1$. Therefore, from (2.4.5) samples drawn from the known importance density $q(\mathbf{x})$ have weights

$$w(\mathbf{x}_k) \propto \frac{p(\mathbf{x}_k | \mathbf{Z}_k)}{q(\mathbf{x}_k)}. \quad (2.4.6)$$

The choice of the importance density $q(\mathbf{x})$ is crucial when designing the SMC. In this case a suboptimal choice is made to approximate $q(\mathbf{x})$. Choosing the transitional prior, $q(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{z}_k) = p(\mathbf{x}_k, \mathbf{x}_{k-1})$, the weights are updated by

$$\mathbf{w}(\mathbf{x}_k^i) \propto \mathbf{w}(\mathbf{x}_k^i) p(\mathbf{z}_k | \mathbf{x}_k^i).$$

This indicates that the weight at time k can only be computed after the observation and the particles have been propagated at time k .

SMC is also known as particle filtering. Other names by which SMC is known include bootstrap filtering, the condensation algorithm, interacting particle approximation and survival of the fittest. The SMC method implements a recursive Bayesian filter of Section 2.3 using the MC integration method (sub-optimal) to evaluate the integrals.

2.5 Particle filter algorithm

The particle filter algorithm is a direct implementation of the recursive Bayesian filter using MC methods. The basic particle filter, *sequential im-*

portance sampling (SIS) algorithm is described in this section as well as the *sampling importance resampling* (SIR) algorithm.

Given the posterior $p(\mathbf{x}_{k-1}|\mathbf{Z}_{k-1})$ and that N samples are randomly drawn $\mathbf{x}_{k-1}^i, i = [1\dots N]$. Then in the **prediction phase** samples are passed from time step $k-1$ and propagated using a dynamic model to generate the prior sample set at time step k . These prior samples $\mathbf{x}_k^i, i = [1\dots N]$ produced by the dynamics model are samples from the prior pdf $p(\mathbf{x}_k|\mathbf{Z}_{k-1})$.

In the **update** step, a new measurement \mathbf{z}_k is obtained. The measurement is used to update the prior according to the particle's weight w_k^i . The new weight value is calculated as the measurement likelihood evaluated at the prior sample: $w_k^i = p(\mathbf{z}_k|\mathbf{x}_k^i)$. The weights must sum to one after normalisation. In the SIR algorithm a further step is added to resample these normalised weights. The resampling algorithm chooses particles from the prior set with a probability equal to its weight. This new set of particles is considered to be samples from the required pdf $p(\mathbf{x}_k|\mathbf{Z}_k)$.

The particle filter algorithm repeats the prediction and update phases at each time step to obtain the posterior pdf at the next time step [17].

2.5.1 Sequential importance sampling (SIS) algorithm

This is the most basic implementation of the particle filter. Sampling is done from the prior pdf and weights are assigned to the particles. The pdf is recursively updated or propagated using measurements at each time step (sequentially). A serious problem arises when applying the SIS algorithm. After a few iterations the pdf collapses around a single particle and all other particles have negligible weight. This phenomenon is called degeneracy. Also, propagating these particles is computationally costly and fails to represent the true pdf accurately. A possible solution requires resampling of the particles.

The SIS algorithm is shown in Algorithm 1. The algorithm notation used in this section is similar to [1]. The state space samples \mathbf{x}_{k-1} have corresponding weights w_{k-1} at time $k - 1$. New observations at time k is described by z_k .

```

input :  $\mathbf{x}_{k-1}, \mathbf{w}_{k-1}, \mathbf{z}_k$ 
output:  $\mathbf{x}_k, \mathbf{w}_k$ 
1 for  $i \leftarrow 1$  to  $N$  do
2   draw  $x_k^i \sim q(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{z}_k)$  generated samples;
3   assign particle weight,  $w_k^i = p(\mathbf{z}_k | \mathbf{x}_k^i)$  ;
4 end

```

Algorithm 1: SIS Algorithm

2.5.2 Resampling algorithm

The SIR algorithm is an extension of the SIS algorithm by incorporating the resampling step described here.

Degeneracy occurs when only a few particles have a large weight and the rest of the particles have weights that are almost zero. In such a situation the prior pdf is not an accurate representation. To reduce the effects of degeneracy on the particle filter a resampling step is added. Resampling is done by choosing particles with larger weights more frequently than those with smaller weights. Different methods of resampling exist such as multinomial, residual, stratified and systematic [2]. A systematic resampling scheme is considered here with complexity of $O(N)$, where N is the number of particles.

Resampling steps The resampling process is shown in Algorithm 2 and the steps are explained as follows. Figure 2.1 illustrates the resampling algorithm.

Step 1 computes the cumulative sum of N particle weights, $\mathbf{C}^i = \sum_{j=0}^i w^j$, $i = [1 \dots N]$. Note that the weights w represent a pdf and that $\mathbf{C}^N = 1$. \mathbf{C} is an index of the cumulative weights and it is divided into equally spaced intervals of $\frac{1}{N}$.

Step 2 sets where the index should start, namely at the first particle's weight index.

In **Step 3** a random offset value $\lambda \in [0, \frac{1}{N}]$, is generated from a uniform distribution.

Step 4 insures that all the particles' weights are considered whilst moving up the index.

Step 5, starting at the offset value, moves up along the index values.

step 6 draws samples by comparing the value λ to \mathbf{C}^i .

In **step 7**, if $\lambda > \mathbf{C}^i$, the particle weight of w^i is small and not sampled by increasing i . This effectively skips past a few particles with small weights. Otherwise, the weight at index i is sampled repeatedly in **step 9** until condition $\lambda > \mathbf{C}^i$ is not true. It is clear that larger weights are sampled more often whilst moving along the indexed values of \mathbf{C} and smaller weights are ignored.

2.6 Summary

The basic particle filter has an elegant and simple algorithm that can be applied in general to most non-linear estimation problems. It is important to realise the dangers, such as the choice of dynamic model, sample set size and impoverishment of the sample set.

```

input :  $\mathbf{x}_k, \mathbf{w}_k$ 
output:  $\mathbf{x}_k^*, \mathbf{w}_k^*$ 

 $C^i = \sum_{j=0}^i w^j$  //Create the cumulative values index;
1
2  $i = 0$  //start offset index;
3  $\lambda_1 \leftarrow \text{random}[0, \frac{1}{N}]$ ;
4 for  $j \leftarrow 1$  to  $N$  do
5    $\lambda_j = \lambda_{j-1} + \frac{1}{N}$  //moving up  $C$ ;
6   while  $\lambda_j > C^i$  do
7      $i = i + 1$ ;
8   end
9    $x_k^{j*} = x_k^i$ ;
10   $w_k^{j*} = \frac{1}{N}$ ;
11 end

```

Algorithm 2: Resampling algorithm

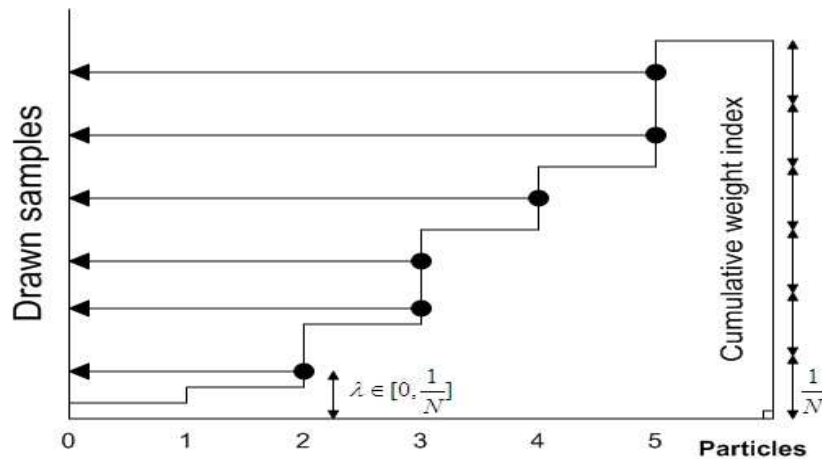


Figure 2.1: Resampling of 6 sample weights.

In Figure 2.2 a possible iteration of the particle filter algorithm is shown at particle level. The graphical representation visually summarises the main idea behind particle filtering. The three main sections as described in this chapter are shown, namely, selection and prediction of the particles, the resampling step and the observational update of the pdf.

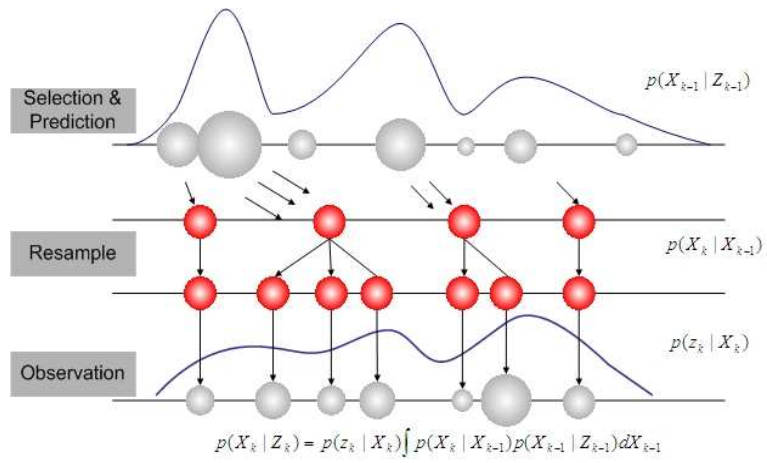


Figure 2.2: Particle filter iteration

Chapter 3

Feature vectors

Feature vectors, such as colour, contour, texture, edge and intensity, describe an object's appearance. Features are collected in a state, and the state is represented by a pdf. The pdf is known through its samples as described in the previous chapter. Sampling measurements represented by these feature are needed to update the particle filter's posterior pdf. Samples are compared and weighted according to these appearance similarities.

This chapter first describes the implementation of a colour- and texture-based feature vector, while in Sections 3.5 and 3.6 describe improvements for a robust tracker. The last section illustrates the feature-based algorithm.

3.1 Particles and Features

Particles \mathbf{s} are vectors $\mathbf{s}^i = [x_i, y_i, dx_i, dy_i, \mathbf{F}]$, $i = [1, \dots, N]$, where (x, y) is the particle's position, (dx, dy) are the velocity components, \mathbf{F} the set of one or more features and N is the number of samples. A particle's features are obtained (a sample) at (x, y) . The feature is extracted from a smaller region in the image which may contain the object. If the target feature is known, the samples are compared and a weight, directly proportional to their similarity, can be assigned to a particle. Particles can in general contain any number of features.

3.2 Colour-based feature

Good features are essential if an object is to be tracked successfully. Colour histograms model an object's colour distribution. These colour histograms have the advantage that objects can have non-rigid shapes or rotate in an environment and still be detectable provided the colour distribution describing the object remains the same.

3.2.1 Colour model

Colour image samples are obtained in a red-green-blue (RGB) representation and converted to a hue-saturation-value (HSV) colour space. A HSV histogram model allows that the intensity, V , can be handled separately. The advantage is that reflections and shadows, mostly present in V space, can be handled more robustly. A 2D Hue-Saturation (HS) and a 1D intensity (V) histogram represent the object's colour feature.

Weighted histogram

Non-rigid objects rarely have a rectangular shape. A kernel function is used to weigh specific positions in an image region differently. Defining a kernel function for example as,

$$k(\mu) = \begin{cases} 1 - \mu^2 & \text{if } \mu < 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.2.1)$$

where μ is a normalised distance of a pixel to its region's center, weighs the colour distribution of pixels on the edges less than in the center. Kernels such as epanechnikov, quartic (biweight), tricube (triweight) or Gaussian could also be employed. In Figure 3.1, the change in radius μ , illustrates how the kernel (3.2.1) weighs the image regions. Assuming that the most important information is contained around the center of an object this function will be adequate and allows for partial occlusion at the edges. An image region, R^i , has a user defined height and width, respectively, H_x and H_y .

The image region, R^i is centred at $(x^i + \frac{H_x}{2}, y^i + \frac{H_y}{2})$. Note that if the entire image is used as a region H_x and H_y then describes the entire image.

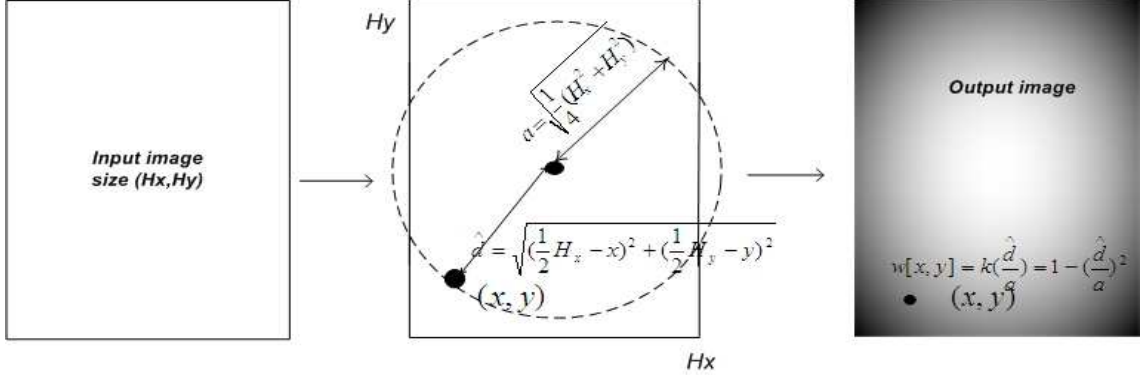


Figure 3.1: Description of weighting function calculation: (left) input image mask, (middle) distances from center, (right) output weighting function.

An image histogram is built using the image patch weighted pixel values of (3.2.1). Every pixel $\mathbf{r} = (x, y)$ in an image region R^i is binned in the histogram

$$p^i[b] = f \sum_{\mathbf{r} \in R^i} k \left(\frac{\|\mathbf{r} - \hat{d}^i\|}{a} \right) \delta [I(\mathbf{r}) - b]. \quad (3.2.2)$$

The distance from pixel (x, y) to region R^i center is $\hat{d}^i = \sqrt{(\frac{1}{2}H_x - x)^2 + (\frac{1}{2}H_y - y)^2}$. Scaling of \hat{d}^i by the region circular radius $a = \sqrt{\frac{1}{4}(H_x^2 + H_y^2)}$ ensures that the kernel function assigns the largest weights to pixels at the region's center. Image I represents the weighted HS- and V-components. The δ function bins the pixels for intensities in image I , into bins b .

The 2D HS-histogram is represented as an image as illustrated in Figure 3.2. The HS-histogram image is divided into rectangles of equal size that represent the histogram bins. A high bin value in the image is proportional to a high colour intensity (white) and a low bin value is black. Representing the colour model in histogram space using (3.2.2), it is possible to compare

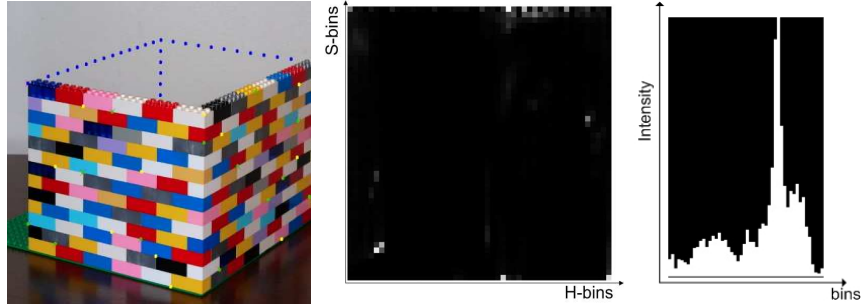


Figure 3.2: (left) input image, (middle) 2D Hue-Saturation histogram image using 50x50 bins, (right) V histogram image 50 bins

feature samples while tracking. To update the recursive nature of the algorithm when new observations are introduced a similarity measure is needed in the tracking estimate, as explained below.

Particle weight update

Given an object's colour feature histogram, the target appearance is known. The target appearance needs to be compared with particle samples that represent the observations \mathbf{z}_k to update $p(\mathbf{x}_k|\mathbf{Z}_k)$. To compare the target model and the samples, the Bhattacharyya similarity that measures the similarity of two discrete probability distributions, is used. Both the 2D HS-histogram and 1D V-histogram similarity values, ρ_{hs} and ρ_v respectively, are obtained when a sample's histogram is compared with the target's histogram model using the discrete Bhattacharyya coefficient

$$\rho[\mathbf{p}^i, \mathbf{q}] = \sum_{b=1}^B \sqrt{p^i[b] q[b]}, \quad (3.2.3)$$

where \mathbf{p}^i are the sample histograms and \mathbf{q} is the model histogram, and B the number of bins. Note that for the colour feature vector $\mathbf{F}_i = \mathbf{p}^i$. Both \mathbf{p}^i and \mathbf{q} are seen as a pdf and normalised to sum to unity. When \mathbf{p}^i and \mathbf{q} are exact, the similarity is maximized, $\rho = 1$. The more similar the appearance between the target and model histograms, the higher the similarity measure ρ . The ρ_{hs} and ρ_v similarity values are combined using

alpha blending to weigh the histograms according to their importance,

$$\rho = \alpha \times \rho_{hs} + (1 - \alpha) \times \rho_v.$$

To minimize lighting changes the V-histogram is weighted less in the experiments and $\alpha = 0.7$ is used. This value was suggested by [6] and tested by trail and error. A smaller value for alpha usually reduces accuracy while tracking and a $\alpha = 0.5$ usually fails to track an object successfully.

The Bhattacharyya distance is calculated using

$$d^i = \sqrt{1 - \rho[\mathbf{p}^i, \mathbf{q}]}. \quad (3.2.4)$$

This distance is used when calculating the particle weights, \mathbf{w} using a Gaussian. The weights of sample set \mathbf{s} then is

$$\mathbf{w} = \frac{1}{\sqrt{2\pi\sigma}} e^{(-\frac{d^2}{2\sigma^2})}. \quad (3.2.5)$$

where the variance σ is a user-defined variable. When the variance is low, the choice of particles with high ρ are favoured when propagated to the next step. The result of choosing σ too small results in a degeneracy of the pdf.

3.3 Histogram of oriented gradients feature

Histogram of oriented gradients (HOG) was developed as a successful human detector [4]. The idea is that gradients of an object contain shape and texture information that can be used to distinguish it from other objects. In this way, HOG captures an object's structure and texture into a feature vector that can be used to detect humans in a scene. The goal is to be able to use HOG features to track an object by comparing sample HOG features with a target appearance.

3.3.1 HOG model description

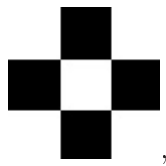
An image \mathbf{I} is divided into uniformly spaced cell regions \mathbf{I}_c . Cells may overlap and have a user defined size C_x, C_y . Calculating the number of non-overlapping cells in an image region is then $L = \frac{H_x}{C_x} \times \frac{H_y}{C_y}$, where H_x and H_y are the dimension of image I . For each cell $\mathbf{I}_c^i, i = [0, \dots, L]$ a histogram of gradients is calculated. Gradients are detected by convolving with a filter mask $[-101]$. When dealing with colour images the gradients are calculated for each colour plane. The gradients are reduced to a single plane by selecting the pixel gradient value with the largest magnitude from each plane. Each cell bins the gradient values weighted according to their magnitude. Combined, these cells form the HOG model's feature vector. The process is illustrated for a single cell in the following example and the results are shown in Figures 3.4 and 3.5 for an entire image.

3.3.2 HOG illustrative example

A checkered board matrix function

$$f(x, y) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad (3.3.1)$$

represented by image



is constructed. Differentiating $f(x, y)$ is done in practise by convolution with the kernel functions,

$$K_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad (3.3.2)$$

and

$$K_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}. \quad (3.3.3)$$

For clarity, in this example, we differentiate $f(x, y)$ using,

$$\frac{\partial f(x, y)}{\partial x} \approx \frac{f(x+1, y) - f(x-1, y)}{2}.$$

and

$$\frac{\partial f(x, y)}{\partial y} \approx \frac{f(x, y+1) - f(x, y-1)}{2}.$$

Boundary cases are handled by padding the edges with the boundary values. Applying the filter above to $f(x, y)$ we respectively obtain

$$\begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}$$

in the x-direction and in the y-direction

$$\begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

Viewing the components $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$ in polar coordinates a magnitude shown in Table 3.1

$$|\nabla f| = \sqrt{\frac{\partial f^2}{\partial x} + \frac{\partial f^2}{\partial y}}$$

and angle shown in Table 3.2

$$\theta = \arctan \frac{\nabla f_y}{\nabla f_x}$$

is calculated, shown here in degrees. A histogram of these calculated gradients, weighted by their magnitude, is constructed as shown in Figure 3.3. Each of these bins can also be represented as a vector with angle equal

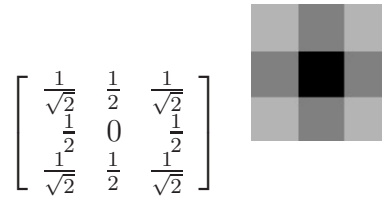


Table 3.1: Magnitude values and corresponding image representation

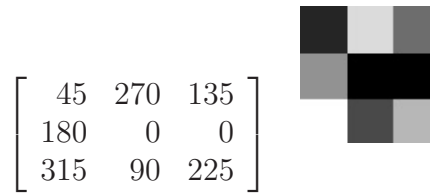


Table 3.2: Angle values and corresponding image representation

to the bin index and magnitude directly related to its bin value. Interesting results are observed when calculating a HOG for shapes with uniform colour and no texture, such as a filled rectangle or circle. Gradient information available only on the edges of these shapes creates a double edge (two neighbouring pixels contain gradient and magnitude information) image that results from the convolution using (3.3.2) and (3.3.3).

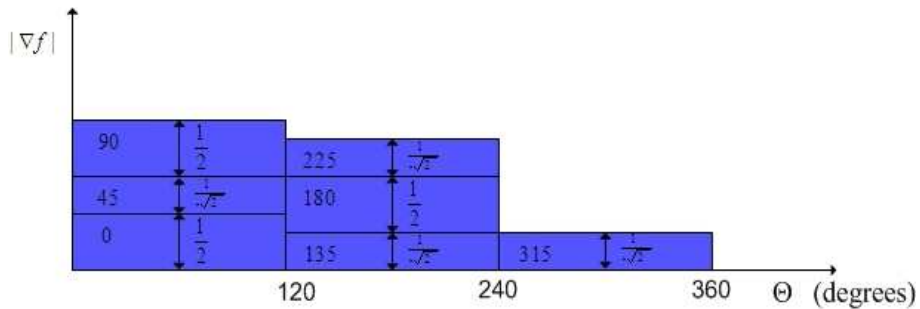


Figure 3.3: Example HOG descriptor for image $f(x, y)$ using 3 bins. Each magnitude and corresponding angle is shown in every bin.

Figure 3.5 shows the HOG descriptor in vector form, for the input image in Figure 3.4 at different cell size selections. When the cell size is small, 2×2 pixels, detail is high and the edge information is clearly noticeable on cells

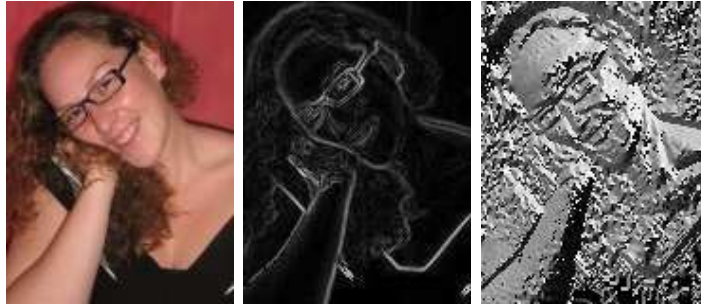


Figure 3.4: Example HOG feature steps for entire input image on left, (middle) magnitude image, (right) angle image

with dense gradient information. Note that by selecting a small cell size such as 2×2 allows that the feature can be compared at different scales by combining neighbouring cells into larger cells. For example, 10×10 cells can be combined from 5 groups of 2×2 cells without recalculation from the source image. Histogram bins can also be reduced by summing neighbouring bins. This less accurate representation might be necessary to calculate a feature more quickly to maintain real-time speeds. It is also important to note that each cell vector is associated with a position in the image.

Note that the normalised HOG descriptor can be interpreted as a pdf with the following useful properties. In this normalised form the HOG is scale invariant and less dependent on the magnitude of the gradients. Special care should be taken to normalise the pdf for a uniform region where no gradients are present in the region (all histogram bins equal zero). This is more likely to happen with smaller cell sizes. This situation is handled separately to ensure that similarity comparison between such features is zero.

Also, note that the HOG descriptor as described is not rotationally invariant. This is explained by the fact that a rotated object's edge gradient values are binned into different histogram bins and usually not in the same cell. Note that the HOG cells prevent that rotation can be detected by

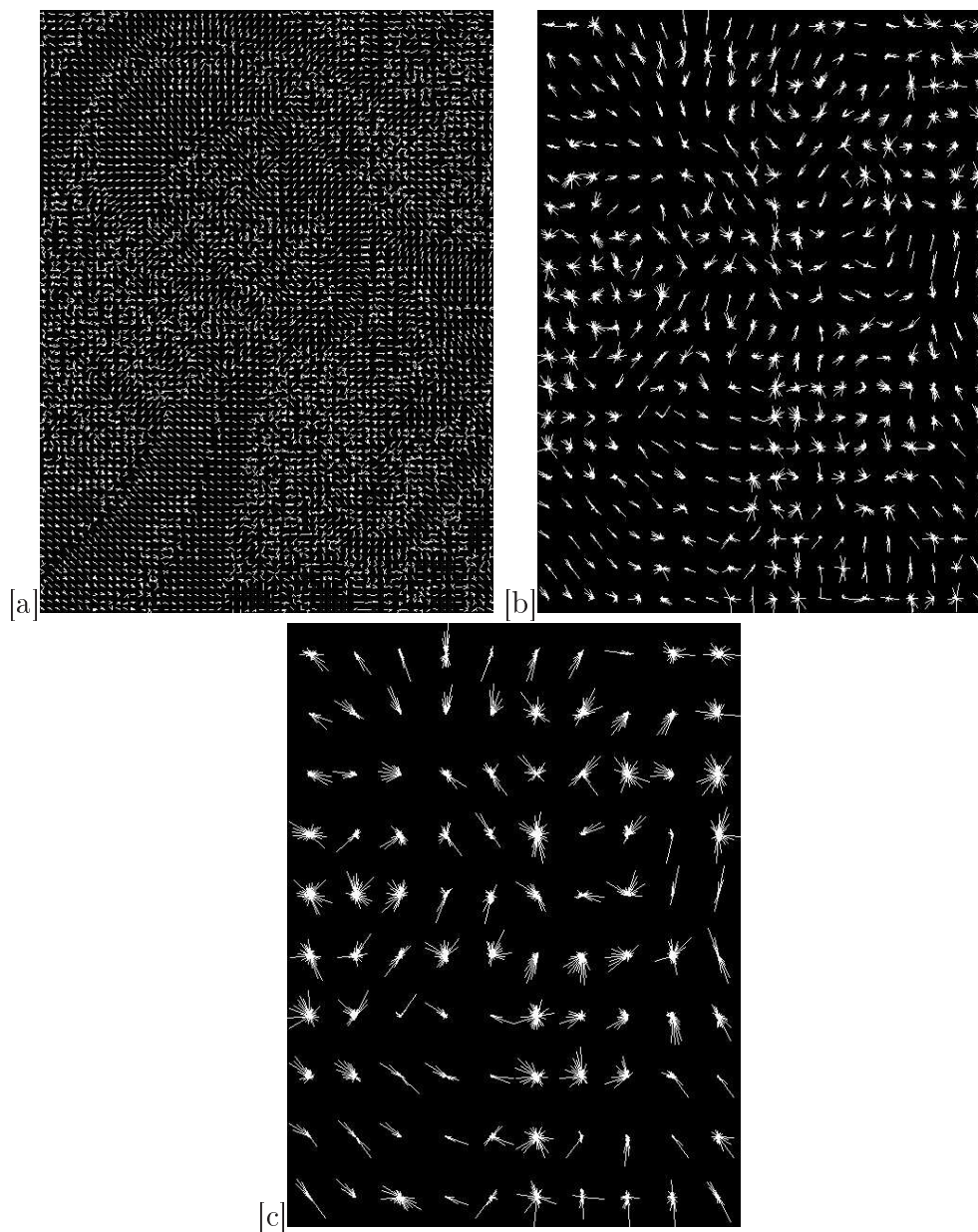


Figure 3.5: HOG features at different cell sizes, using 36 bin histograms. The images show each cell's HOG as vectors graphically (a) cell sizes at 2x2, number of histograms 60x80, (b) cell sizes at 6x8, number of histograms 20x20, (c) cell sizes at 12x16, number of histograms 10x10

a linear shift of each histogram. Comparing two HOG descriptors quickly becomes a challenging problem. This observation is explained in the next section.

3.3.3 Similarity measure between HOG features

Comparison between two HOG vectors $\mathbf{v}_1, \mathbf{v}_2$ is done in a similar way to that of the colour-based feature vector. Representing each HOG vector cell \mathbf{I}_c^i as a probability distribution, the Bhattacharyya similarity measure can be used. The cell similarity measures are combined in a single similarity value by taking the average over the similarity measures,

$$\rho = \frac{1}{L} \sum_{i=1}^L \sum_{b=1}^B \sqrt{v_1^i[b]v_2^i[b]}, \quad (3.3.4)$$

where b is the gradient histogram bins, L is the number of cells and B is the number of bins. A comparison between the trained HOG model and particle sample HOG's allows the use of the similarity value, ρ to update the particle weights using (3.2.4) and (3.2.5). In practice, however, this approach fails to be an accurate measure to track an object and is discussed in the next section.

3.3.3.1 HOG similarity used in tracking

Dividing an image into cells allows that changes in small parts of the image do not effect the entire feature. This is an advantage when using HOG for detection as presented in [4]. When viewed in a particle filter tracking context, two challenging problems arise. Firstly, sampling at predicted locations does not, in general, sample at exactly the correct position. Consider the situation where a sample is taken just left of the actual object location. Then each of the cell histograms contain gradient information that are unaligned to the right of the model histogram, resulting in a low similarity. Secondly, histograms are binned using an image's gradient angles. Cell histograms are not rotational invariant in such a situation. Again, samples

might be mis-aligned due to object rotation. This is not easy to deal with if multiple cells are used. Thus, instead of using multiple cells, we use a single cell for each region.

Using the tracker predictions of possible object location the single cell HOG is used to find a similarity value. The advantage of this is three fold. Firstly, samples at non-exact predicted locations that contains only part of the target might still have a large similarity. Secondly, object rotation can easily be handled by a linear shift. And thirdly, a much faster implementation is possible assuming that object rotation between frames is small. Then correlation reduces to shifting the histogram bins one bin position left or right respectively. For example, using 36 bins, an object can rotate 10 degrees without affecting the similarity value.

3.3.3.2 HOG similarity comparison

An experiment is done to determine how similar objects appear using the single histogram HOG and the general HOG with different cell sizes. A subset of the ETH-80 dataset is used [10] to test how well HOG descriptors compare objects at different bin and cell sizes. Figures 3.7 shows the effects of bin and cell selection when comparing objects in Figures 3.6 centred in an image.

A pear image is chosen as a model in Figures 3.7 (a) and compared with other pear images. Each of the pear images is then compared with a tomato image and the results are shown in Figure 3.7 (b). Again the test is repeated where a cup image is compared with each pear image. The similarity results for a range of different bin sizes are shown in Figure 3.7 (c).

For each of these tests the single vector (1 cell) HOG results are shown in Figure 3.7 (d). These results show a 5% better similarity when comparing pears with pears than comparing tomatoes and pears. And a 10% better similarity is obtained when comparing a cup with pears. Also, note that changing the number of bins does not effect these results.

The results show that similarity measurements suffer greatly when the number of cells is increased. A single cell representation achieves best results for all bin sizes tested. Tracking using a single cell HOG vector improves performance and an experiment is done in Section 3.9. In the experiment it is shown that a single cell HOG feature is more robust, allowing for small translation and rotation errors from tracker predictions.



Figure 3.6: Model images used to obtain similarity results of Figure 3.7.

3.4 Motion model

Particles are propagated to the next step according to a dynamic motion model. A constant velocity model is used without acceleration. Acceleration, handled by noise, is not considered since the state space becomes too high dimensional and requires far too many samples, which is computationally expensive. Particles are propagated using

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + \mathbf{w}_{k-1} \quad (3.4.1)$$

where A defines deterministic parameters, \mathbf{w}_{k-1} the stochastic and k the time. We remind the reader that \mathbf{x}_k is the state space representing the dynamics. Using (3.4.1) to propagate a particle using the motion model its

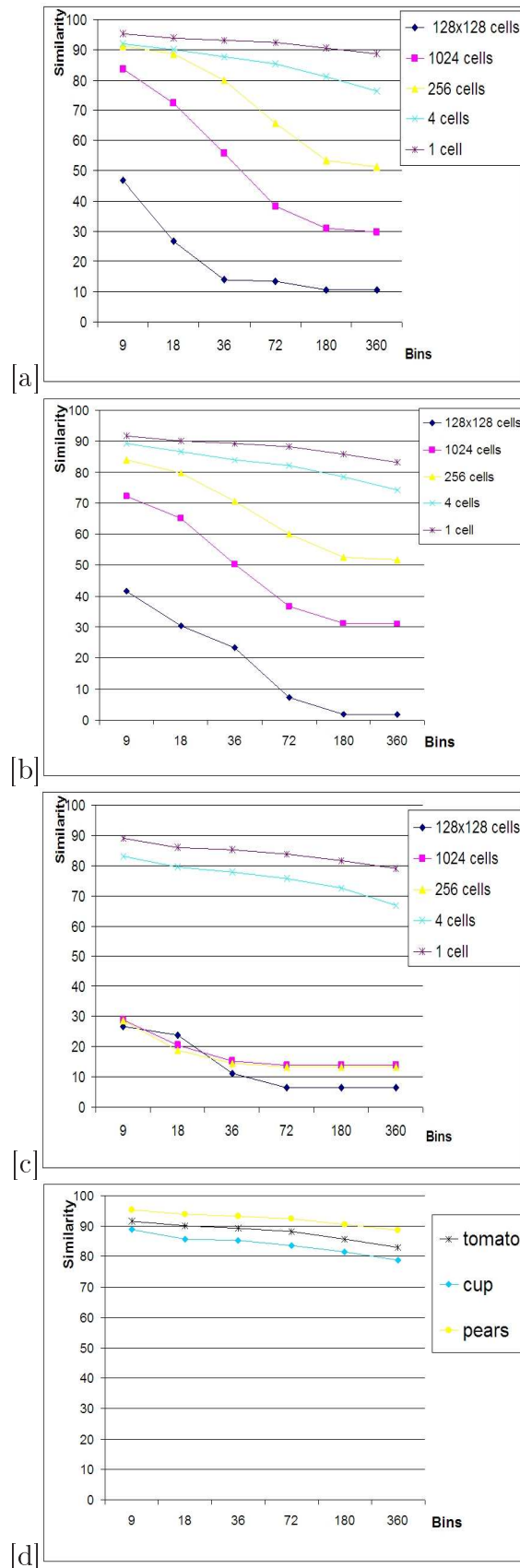


Figure 3.7: HOG features comparison at different bin and cell sizes. (a) Comparison results of image set pears. (b) Comparison results of comparing a tomato with pears. (c) Comparison results of a cup with pears. (d) Comparing results of cup, tomato and pears using only one cell.

(x, y) coordinate is updated using velocity $\mathbf{v} + k$,

$$\begin{bmatrix} \mathbf{x}_k \\ \mathbf{v}_k \end{bmatrix} = A \begin{bmatrix} \mathbf{x}_{k-1} \\ \mathbf{v}_{k-1} \end{bmatrix} + \mathbf{w}_{k-1}. \quad (3.4.2)$$

The value of A is user defined to be either a random position model,

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad (3.4.3)$$

or a constant velocity model, also used in [6]

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}. \quad (3.4.4)$$

3.5 Feature adaptivity

Changes in lighting and shape of the object, result in a bad representation of the histogram describing the object. Appearance changes of an object can be handled by adapting the model to increase tracking robustness. Adaptivity as implemented and tested here is presented in [6], [8], [15]. When tracking an object in real-time, adapting the target model needs to be done automatically.

The model q_k is adapted using,

$$q_k = \alpha \times s_k^j + (1 - \alpha) \times q_{k-1} \quad (3.5.1)$$

where s_k^j is the most likely object position at time k . Each target bin is blended mixing $\alpha \in \{0, 1\}$ with sample j , having the highest appearance similarity of all the samples. This is done for both the HS- and V-histogram colour feature and HOG feature. The choice of α is directly related to the confidence measure described in the following sections.

Selective adaption is necessary to avoid adapting the target feature models in cases where the tracked object is lost. If the loss is undetected, the models will be incorrectly updated and become corrupted. This is clearly an undesirable effect. Automatic adaption is possible using a confidence measure to only adapt if the system has a high confidence that the object is being tracked. A slow adaption rate handles occlusion better since the target model changes less over time. Fast appearance changes are handled when the rate of adaption is quick. Note that the rate at which adaption is applied affects the situations that can be handled by the tracker. Consider the situation when the tracked object moves behind a structure and the adaption is fast. While the object is lost from view the target model is adapted incorrectly using the best predicted location. When the object reappears it might not be tracked correctly due to a bad representation of the target model.

In cases where the object is being tracked with high precision, the tracking pdf has a high peak and most samples are grouped together. On the other hand a low certainty of object position is shown by a uniform pdf. In [6] the confidence is measured directly from the tracking pdf. The confidence measure is described by the degree of unimodality of the resulting pdf $p(\mathbf{x} | \mathbf{Z})$. A low confidence is measured when the pdf has a very uniform distribution. The particle weights approximate the confidence of the tracked pdf. This computationally simple confidence measure works well. However, failure can occur. When the background region's colours or textures are similar to the target model or the size of the particle's image patches are considerably smaller than the region being tracked, which might have a uniform colour, confidence is low. In both cases the pdf becomes more uniform, and the confidence measure incorrectly results in a tracking loss.

Experiments using different values for σ in (3.2.5), illustrated that the confidence measure is related to the choice of σ . Note that σ determines the variance in the position of particles. A confidence value is obtained from a

threshold defined by

$$S_\sigma < K\left(\frac{1}{\sigma}\right),$$

where σ is the user defined value from (3.2.5), K a normalisation and S_σ the standard deviation of the tracking pdf. As mentioned previously the choice of alpha is directly related to the confidence measure. Since the confidence value can be calculated during run-time it is used as the value for α in (3.5.1). It is now clear that the target model is only adapted when the confidence is high.

The next section describes how to recover from tracking failure. Both methods described above are used to calculate a confidence when testing whether to adapt the histogram model. These methods can also be used to determine whether an object is being tracked correctly.

3.6 Finding an object and detecting a loss

Assume that the object that will be tracked is known. Then its features, available as a pdf, are also known. Finding the object's position in an image is then possible.

Using the prior knowledge of the target histogram, a search for the object in the first frame can be done. The Bhattacharyya similarity measure (3.2.3) is used to compare the target model at every image region. These regions will have a low similarity when the object is not present and a high similarity when the object appears in the frames. A mean value μ and a standard deviation σ of the similarities in all the regions are calculated

$$\mu = \frac{1}{M} \sum_{i=0}^M \rho[\mathbf{s}_{R_i}, q], \quad (3.6.1)$$

$$\sigma^2 = \frac{1}{M} \sum_{i=0}^M (\rho[\mathbf{s}_{R_i}, q] - \mu)^2, \quad (3.6.2)$$

where \mathbf{s}_{R_i} are samples calculated at each of the M regions in the image. Assuming a Gaussian distribution¹ an appearance threshold is defined, in [14],

$$\rho[\mathbf{s}_{R_i}, q] > \mu + 2\sigma. \quad (3.6.3)$$

The appearance threshold indicates a 95 % confidence that the region R_i is not part of the background. The particle filter is initialised in the region if more than a user-defined fraction of the sample set \mathbf{s} meets the appearance threshold. The same rule is applied to detect when the tracker loses the object. When the object leaves the frame or becomes occluded for a couple of frames, condition (3.6.3) fails and the initialisation phase is entered again.

3.7 Perspective adaption

Adjusting the region size according to the object's perceived size is necessary to robustly track objects in a 3D environment. An object moving away from the camera, changes size relative to the camera's perspective. Detecting whether an object is moving closer or away from the camera is done by sampling at different region sizes. The region's size is sampled at $\pm 2\%$ of the region size (H_x, H_y) at the current best predicted location and compared to the target model. If a comparison is found to have a higher similarity to the feature models, the region size is adjusted. Note that features such as colour and HOG are scale invariant so an adjustment of the region size does not affect the features. Also, it is useful to only adapt when there is a significant difference in the similarity value to minimise computation.

This adaption is not directly related to the confidence measure, but results in a higher confidence if the object's size is sampled at correct region sizes to avoid including background which leads to bad feature model representations.

¹The empirical rule states that for a normal distribution assumption, about 68% of the values are within 1 standard deviation of the mean, about 95% of the values are within two standard deviations and about 99.7% lie within 3 standard deviations.

3.8 Algorithm

Implementation of Algorithm 3 follows the same steps as the basic particle filter from Algorithm 1 using the resampling step described in Section 2.5.2. The specialisation of the feature-based steps are described using the models and rules described throughout this chapter.

```

1  #Initialization step;
2   $\mathbf{q}_k =$  get observation model at time,  $k=0$ ;
3  while true do
4       $\mu, \sigma$  from eq 3.6.1 and 3.6.2;
5       $f = \sum_{i=0}^N \rho[\mathbf{p}_k^i, \mathbf{q}_k] > \mu + 2\sigma$ ;
6      if  $f > (0.1)N$  then
7          objectfound = true;
8      end
9      #Measurement step;
10     for  $i \leftarrow 1$  to  $N$  do
11          $s_{k-1}^i \leftarrow$  get particle samples, eq. 3.2.2;
12          $\pi_{k-1}^i \leftarrow$  assign particle weight, eq. 3.2.5;
13     end
14     normalize  $\pi_{k-1}$ ;
15     # Robustness improvements;
16     //confidence measure;
17     if objectfoundandconfidence  $>$  Threshold then
18         Adapt_sample_size();
19          $\mathbf{q}_{k+1} =$  adapt histogram( $\mathbf{p}_{k-1}, \mathbf{q}_k$ ) , eq. 3.5.1;
20     end
21      $\pi_k \leftarrow$  resample pdf  $\pi_{k-1}$  using algorithm2;
22     # Prediction step;
23      $\mathbf{s}_k \leftarrow$  apply motion model, eq 3.4.1;
24 end

```

Algorithm 3: Feature-based particle filter algorithm

3.9 Feature tracking experiment

Implementation of Algorithm 3 is tested using each of the feature types; colour, texture and a combination of both. In the latter case, a user defined weighting value is used to combine the features using alpha blending. For generality, an arbitrary number of features can be handled in this manner.

Experiment 1 A simulated test is done to accomplish the following;

- Colour object tracking using a HS-,V- histogram descriptor
- Texture object tracking using a HOG-histogram descriptor
- Combined feature tracking
- Tracking through clutter/noisy background
- Correct tracking with partial occlusion
- Correct tracking with full occlusion

As shown in Figure 3.8 the simulated test places four simple rigid shapes, two triangles and two rectangles, each following a circular path shown in Figure 3.8. All objects have constant movement and maintain their circular motion in their own direction. Each of the rectangle and triangle colour shapes intersect and overlap with other shapes with the same colour.

The white rectangle is the object being tracked. Figure 3.8 shows the rectangles at interesting positions as well as the particles X,Y movement along the circular path. The sequence runs for 620 frames. The particle's position and weight are shown as red circles on the image where the size of the circle is directly related to the weight.

Colour object tracking It is clear that the colour-based tracking is likely to fail at some point due to background colours and other shapes with the same colour. The particle movement in the Y-direction Figure 3.8 (c) of this sequence shows that at frame 80 the direction changes as the tracker

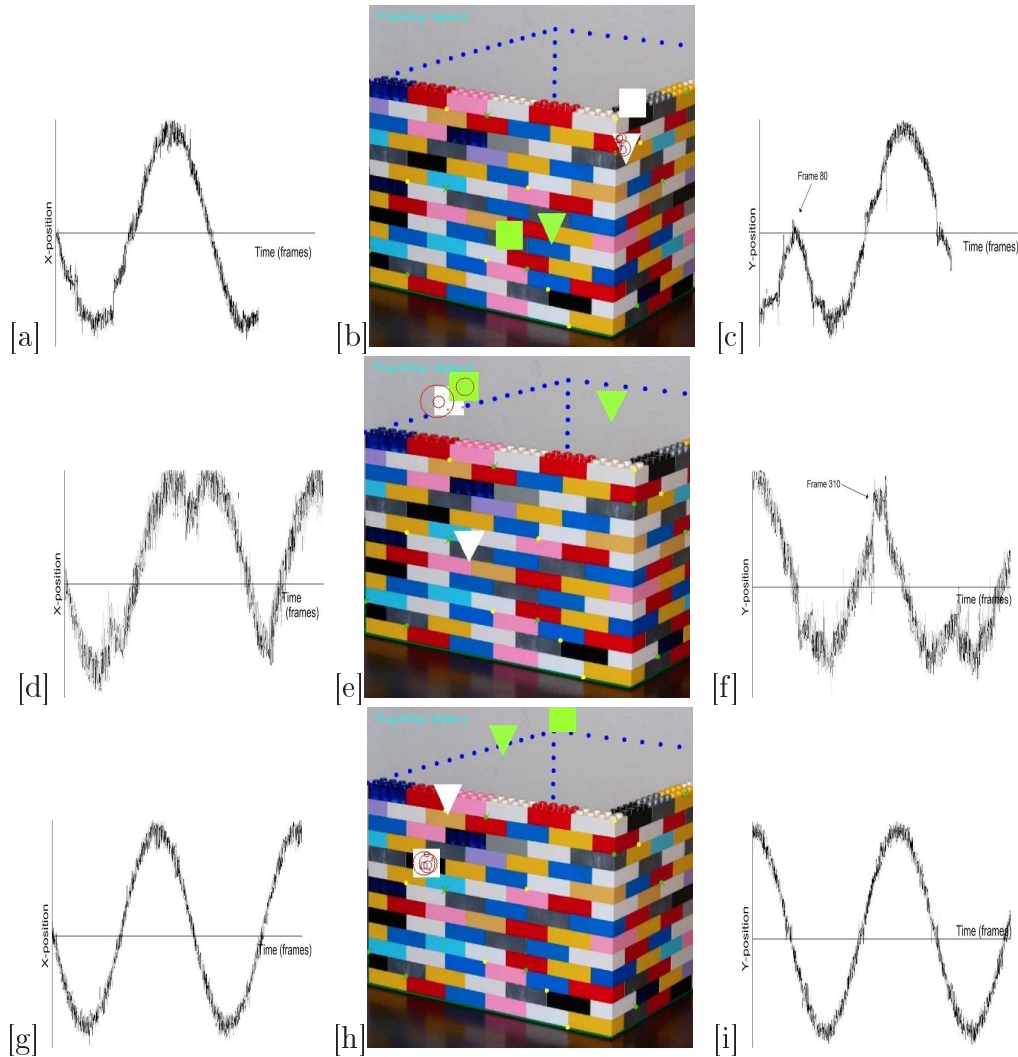


Figure 3.8: (a,b,c) Colour tracking frame 80, (d,e,f) HOG tracking frame 310, (g,h,i) feature combined frame 250. (a,d,g) Tracked particles X-movement, (c,f,i) tracked particles Y-movement. 50 particles are used with a zero velocity motion model.

confuses objects moving in opposite directions. The failure is the result of a higher similarity value for a white triangle than the white square when the two objects cross (a partial occlusion).

HOG object tracking The HOG tracker fails at frame 310 when the two rectangle shapes with colours green and white overlap (full occlusion). From Figures 3.8 (*d, f*) we see the sudden change of direction in particle movement at frame 310. From Figure 3.8 (*e*) it can be seen that the particles are distributed across both rectangles each having a high similarity value as they move past each other.

Combined object tracking When features are combined, it is clear from the X-and Y-direction graphs, Figures 3.8 (*g, i*), that the particles track the correct object throughout the sequence successfully through partial occlusion (rectangle moves under triangle) and full occlusion (white rectangle moves under green rectangle).

Experiment 2 The goal of this experiment is to illustrate HOG feature adaption to accurately track an object rotating as described in Section 3.5. As described previously, a single cell HOG feature is not rotationally invariant. However, object rotation can be detected by a linear shift of the histogram bins. This experiment tracks a rotating square object. The object is rotating around its center while it is following a circular path. The HOG feature is represented as a vector where the angle describes the histogram bin and the bin value the vector's magnitude. In Figure 3.9 the HOG feature is shown in each of the frames at the bottom right corner.

Adapting the feature model is only done when the tracking confidence is high. A low tracking confidence is measured when only HOG is used. The reason for this is that the noisy background is similar to the object. The result is that adaption to the rotating object is not done. Tracking fails when the object has rotated more than the linear shift allows. To increase the confidence the colour feature is also used. From the particles X and

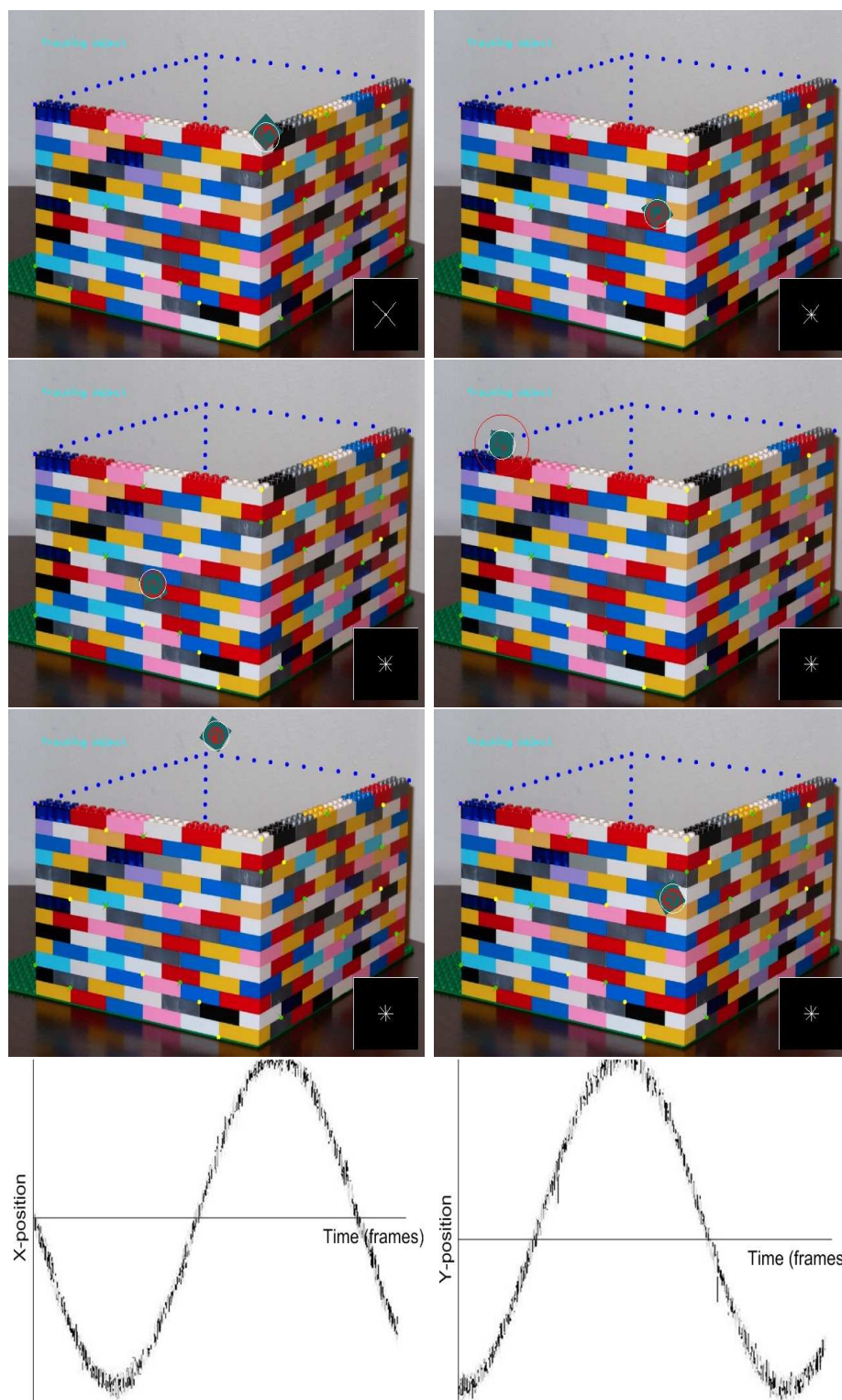


Figure 3.9: Tracking experiment 2: HOG adaption. Tracking a square over 400 frames. Selected frames 1,50,100,200,300,400, shown. (Bottom left) X-position of particles.(Bottom right) Y-position of particles

Y positions it is clear that tracking using the HOG adaption is successful when the colour feature is used to increase confidence. When the confidence is high, notice how the noisy edges in the background also become part of the model, seen as vectors at right angles. Feature adaption is successful, and the rotating object is tracked accurately.

3.10 Summary

Both colour and texture information are modelled as features that can be used to describe an object. The histogram methods used allow that adapting to objects undergoing small shape, rotation, size or colour changes can be handled effectively. These feature are clearly useful for tracking purposes. Also, combining features significantly improve results. It is also important to note that the process runs real-time when the region sizes are small. Performance is mostly affected by the number of particles and the image region sizes that need to be processed to extract features.

Chapter 4

System implementation

Robustly tracking objects relies heavily on accurate features. The feature-based particle filter is most effective when the object is rigid, can only rotate in a 2D-plane and has a constant colour and texture histograms. The self-adapting histogram components and confidence measure are added to handle realistic tracking scenarios more effectively. The next challenge is to obtain prior knowledge of the object's features and dynamic information about its movement. This chapter describes these challenges and presents an approach to integrate automatic feature extraction of moving objects and feature-based particle filtering inside a system.

4.1 System design and goals

Automatic object tracking relies heavily on robust object detection and, in our case, initialisation of motion and features. These different challenges are implemented in self-contained modules that need to be integrated in a system. A modular approach described in [7] is used where a semantic ladder, built from feature extraction to action recognition, describes the challenges as well as the system implementation. This intuitive design approach allows for models that can be created to handle a specific problem where each step up the ladder relies on the previous step. In this way the system is dynamic in that all components can be replaced as improvements

to technology and algorithms become available. Using this high level design methodology an automatic tracking system has been developed.

Each of the following sections are modules that, when combined, handle the system from initialisation to the tracking of an object. The design and implementation of feature models from Chapter 3 are shown in Figure 4.1. Note that at each level the design is modular and easily extendable.

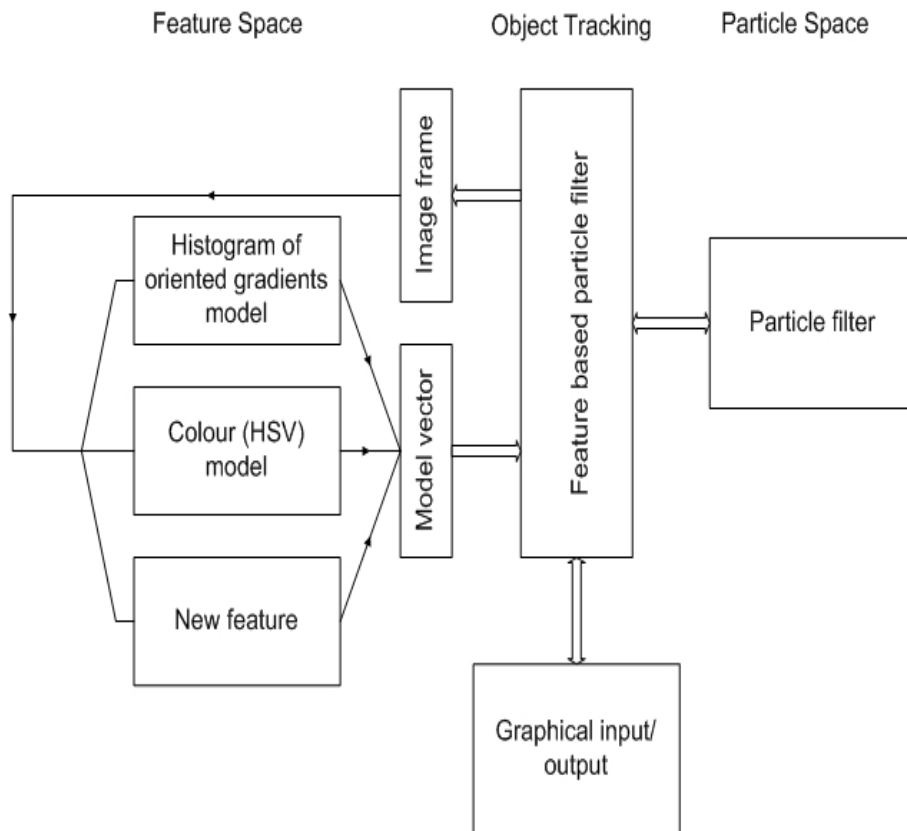


Figure 4.1: Feature-based tracking modules

An overview of the system modules is shown in Figure 4.2 and described in the following sections. Note that the focus of these sections are to detect an object of interest in a scene. Detection can also be used to track objects by means of repeated detection in every frame. This is very time consuming and it is computationally much quicker to track an object using prediction methods such as the particle filter. Real-time tracking is considered to

be at least 5 frames per second (fps). Although, speeds of more than 10 fps is achieved when using a single module. Most web cameras can perform theoretically up to 30 fps, but realistically speeds of 5 to 20 frames is normal.

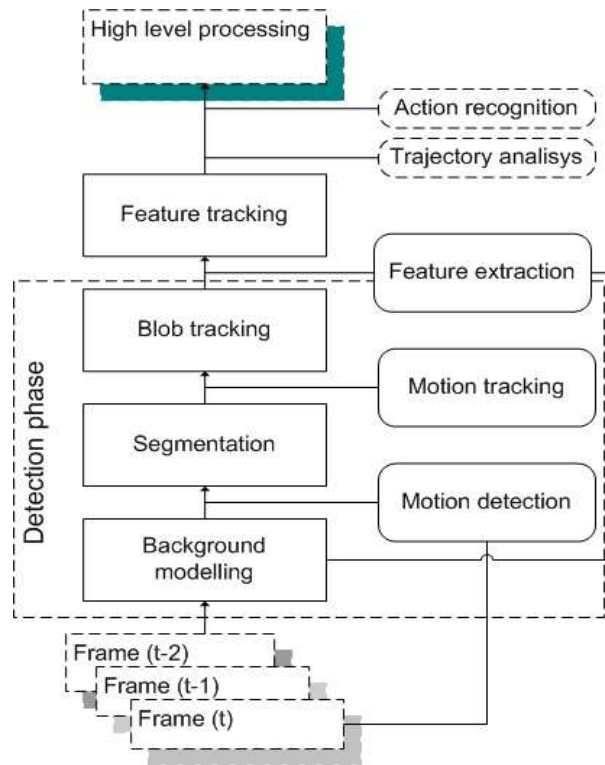


Figure 4.2: System module design

4.2 Background modelling

A background region contains objects that stay in the same place or bounded region over time, while foreground regions or regions of interest move around more freely. Information about a scene's background is useful to minimise noise during tracking or when extracting features. A background model can be defined as a reference structure that describes the background of a scene. The simplest structure being a time-averaged reference image where

concurrent frames are subtracted. Concurrent subtraction of frames results in very noisy images that need to be cleaned, usually using thresholds. Obtaining an accurate background model using these computationally simple algorithms requires a training period within a controlled environment absent from movement, foreground objects or illumination changes. Any changes to the scene requires a re-estimation of the background. This type of solution consequently requires that the background be updated constantly.

Background modelling is a separate field of research and two popular types of background modelling techniques are investigated in this section and compared with a time-averaging method. A foreground object detector [11] and an adaptive background mixture model [9] are investigated. These methods are chosen based on their ability to dynamically model complex scenes and real-time execution. Both FGD and the background mixture model algorithms are implemented in the OpenCV library.

4.2.1 Foreground Object Detection (FGD)

In a complex scene, possibly containing dynamic moving objects such as trees, background pixels can have multiple values. FGD integrates multiple features where most other background modelling techniques only use one type of feature to model static and dynamic parts. The FGD's focus is to model different parts of the background using different types of features. Feature models for both static and dynamic background pixels are used. Extracting foreground objects from a complex scene is done using a Bayes decision rule which has been extended to deal with general features. It is then possible to classify both background and foreground pixels using multiple features.

Classification rule A classification rule is formulated in general to classify a pixel as foreground or background. Following the notation in [11], let \mathbf{v}_k be a feature at time k located at position $\mathbf{r} = (x, y)$ where \mathbf{r} is possibly

a background or a foreground pixel. Using Bayes theorem, the posterior probability of \mathbf{v}_k of a background pixel b or foreground pixel f is

$$P(C | \mathbf{v}_k, \mathbf{r}) = \frac{P(\mathbf{v}_k | C, \mathbf{r})P(C | \mathbf{r})}{P(\mathbf{v}_k | \mathbf{r})}, \quad (4.2.1)$$

where $C = f$ or b . Classification of a pixel as foreground using Bayes rule is given by

$$P(f | \mathbf{v}_k, \mathbf{r}) > P(b | \mathbf{v}_k, \mathbf{r}). \quad (4.2.2)$$

To classify a pixel at run-time as part of the foreground or background the probabilities, $P(b | \mathbf{v}_k, \mathbf{r})$, $P(\mathbf{v}_k | \mathbf{r})$ and $P(\mathbf{v}_k | b, \mathbf{r})$ need to be trained. A table structure is used to store these statistics for every pixel in the image.

Table of feature statistics In [11] a histogram of feature vectors is used to approximate $P(\mathbf{v}_k | \mathbf{r})$ and $P(\mathbf{v}_k | b, \mathbf{r})$ which is not known in general.

A background pixel only has a limited number of values, they are considered to only be concentrated in a small subspace of the feature histogram. This indicates that with a good feature selection a background pixel can effectively be covered by a small number of histogram bins. On the other hand, foreground pixel values will not be as concentrated in these histogram bins and will in general be spread more widely. Then we let $P(\mathbf{v}_k^i | \mathbf{r})$, $i = 1, \dots, N$ be the first N bins from the feature histogram describing the multiple background values.

A table of feature statistics is created to store the different feature histograms. The table $\mathbf{S}_{\mathbf{v}_k}^{\mathbf{r},k}$ of feature statistics maintains three components for every pixel in an image,

$$\mathbf{S}_{\mathbf{v}_k}^{\mathbf{r},k,i} = \begin{cases} p_v^{k,i} = P(\mathbf{v}_k^i | \mathbf{r}) \\ p_{v,b}^{k,i} = P(\mathbf{v}_k^i | b, \mathbf{r}) \\ \mathbf{v}_k^i = [a_1^i, \dots, a_n^i]^T \end{cases},$$

where a_j^i are the different states that a feature can have. For a feature \mathbf{v}_k the table maintains the most significant portion, where there is the

highest concentration of pixel values, of the feature histogram. The table is maintained by each update of the background model. The probability to classify a pixel as foreground, $P(b \mid \mathbf{v}_k, \mathbf{r})$, $P(\mathbf{v}_k \mid \mathbf{r})$ and $P(\mathbf{v}_k \mid b, \mathbf{r})$, are known for each pixel from the feature table statistics.

Feature vectors For a pixel classified as part of a static background the colour is chosen as a feature to be stored in the table and, \mathbf{v}_k is substituted in (4.2.1) by $\mathbf{c}_k = [r_k \ g_k \ b_k]^T$. Static backgrounds where pixel values do not change over time is simple to handle. The feature \mathbf{c}_k is chosen if the first N entries in the feature table do not vary.

A moving background's pixel values change between frames. The colour co-occurrence of the change in pixel values between frames are chosen as a feature vector and again, \mathbf{v}_k is substituted in (4.2.1) by $\mathbf{o}_k = [r_{k-1} \ g_{k-1} \ b_{k-1} \ r_k \ g_k \ b_k]^T$. Selecting the colour co-occurrence feature is based on the observation that, for a moving background, the pixel values varies greatly, and always at the same location in an image. Both states $\mathbf{S}_{\mathbf{c}_k}^{\mathbf{r},k,i}$ and $\mathbf{S}_{\mathbf{o}_k}^{\mathbf{r},k,i}$ are stored for every pixel to represent the multiple states. Representing the background using multiple states allows for alternating pixel values without noisy interference with foreground objects. In [11] the complete algorithm is discussed in detail.

4.2.2 Mixture of Gaussian background modelling

The adaptive mixture of Gaussian (MOG) models the variation in pixel values using a Gaussian mixture model (GMM) consisting of up to K Gaussians, where $3 \leq K \leq 5$. Each pixel in an image is modelled by a MOG distributions. Different Gaussian represents different colours. Note that we have mentioned that background pixels are present in a scene for longer periods. Then, a weight w is applied to each Gaussian that is proportional to the time those colours stay in a scene. The idea is that a pixel is drawn from a GMM allowing for multi-modal distributions of pixel values. The first N most frequent occurrences of a specific colour is considered to rep-

resent the background model. The adaptive background mixture model is developed in [9] and builds on previous work done by Grimson and Stauffer [19]. This method improves the update speed (learning time) of the background model.

MOG Model The K Gaussian's at pixel $\mathbf{r} = (x, y)$ models the probability of the colour values $\mathbf{c}_k = [r_k g_k b_k]^T$ at time k and we write

$$p(\mathbf{c}_k) = \sum_{i=1}^K w_i \eta(\mathbf{c}_k | \mu_i, \Sigma_i), \quad (4.2.3)$$

$$\sum_{i=0}^K w_i = 1. \quad (4.2.4)$$

as a 1 dimensional GMM. Then w_i is the weight of the i^{th} Gaussian and $\eta(\mathbf{c}_k | \mu_i, \Sigma_i)$ is its normal distribution. Training is needed to find w_i, μ_i, Σ_i and the standard EM algorithms are used. The method is improved upon in [9] to speed up the learning time. A two step process is used in the optimised equations. Firstly, estimation of the mixture model by the EM algorithms are performed. After this initial estimate, the updating step only considers the last L frames allowing current changes in the scene to have a higher priority. This improved adaptive MOG adapts quicker and has a learning time much shorter than [19].

4.3 Motion tracking

Motion tracking picks up constant motion based on repeated detection in every frame. Motion detection is used to find regions of interest which are defined as regions which are consistently present in consecutive frames to minimise noise. Since noise is considered random it is assumed not to have a constant motion and is only present in a sequence of frames for short periods. Scenes composed of objects in constant movement in front of static backgrounds are assumed. Note that motion tracking refers to a

simple method for detecting objects in a scene and the tracking only refers to the matching (keeping track) of these regions between frames.

4.3.1 Motion Detection

Motion detection is the first step in processing input frames. Connected components (blobs) in image M are segmented into rectangular regions. From experiments it is found that many rectangle regions of the same moving object overlap. These overlapping rectangles are combined to form a larger rectangular region to completely bound the object. Filtering out of small regions is done after overlapping rectangular regions have been combined. These regions are processed as described in the following section.

Motion detection builds up a motion image that captures pixels that change between frames. Motion is detected by maintaining a sequence of the last consecutive frames in gray scale. A silhouette image, S , is calculated by the absolute difference between frame at time k and its preceding frame at time $k - 1$, and then thresholded to remove small and isolated noisy regions. The motion image M is constructed and maintained by updating M using S ,

$$M(x, y) = \begin{cases} k, & \text{if } S(x, y) \neq 0, \\ 0, & \text{if } S(x, y) = 0 \text{ and } M(x, y) < k-D, \\ M(x, y), & \text{otherwise,} \end{cases}$$

where D is the duration that pixels are allowed to be present in a scene. D is a user-defined constant value in milliseconds. A large value for D increases the time a pixel is present in M and usually results in a delayed shadow or ghost effect. A small value of D decreases the likelihood that slow motion is detected.

4.3.2 Motion tracking implementation

Regions detected in the motion algorithm are processed and tracked. Not all of these regions are consistent in their motion over a time period and need to be discarded. The process is divided into logical sections and is described in each of the following steps: detecting tentative regions, confirming a tentative region and preparing a region for initialisation for feature extraction in the particle filter.

Step 1: Tentatives Motion tracking keeps track of regions that have been detected. Newly detected regions, blobs of motion pixels group together, are labelled as tentative when they first appear. Matching of these regions to previously detected regions is done in a nearest-neighbour fashion. Only regions within the tentative region's neighbourhood are tested for a match. Regions are matched by their width and breadth. These regions are allowed to change in size in consecutive frames. If there is no match to previous regions, the new region is given a timestamp and linked to a tentative list of regions.

Step 2: Confirmed Continuous detected tentative regions are upgraded to a confirmed region if motion is present for a minimum time limit. Any region from the tentative or confirmed list is removed if they are not detected within that minimum time limit. This step has the advantage that background noise is quickly removed before the region is confirmed. Also, any confirmed regions are removed if in consecutive frames there is no new detected region that matches the size and velocity in that region's close proximity.

Step 3: Initialisation Each region in this tracker has a history vector \mathbf{h} describing its positions (x, y) , width and height (w, h) and speed components (v_x, v_y) , such that $\mathbf{h} = (x, y, w, h, v_x, v_y)$ over a period of sequential frames. These parameters are used when confirmed regions are passed to the particle filter to automatically initialise the dynamics and extract features.

4.3.3 Motion tracking results

This section discusses the results of the motion tracking method described above. A video sequence is chosen to illustrate the basic tracker and its shortcomings. In Figure 4.3 a sequence of frames is shown. Tracked regions which are confirmed, are shown in yellow, while tentative regions are green. A number label inside each rectangle is added to each tracked region for identification.

From frame 380 shown in Figure 4.3 it can be seen that 3 regions are correctly labelled. A car stopping at the stop sign is lost (no motion regions are detected) in frame 430 and tracked again when motion resumes in frame 530. Two regions intersect in frame 680 with matching area sizes. Motion detection creates a noisy region shown in green that quickly disappears again. The two intersected regions are still separate in frame 730, but their labels have switched.

Motion detection has another unwanted property due to its construction. Any fast moving object in a frame creates a ghost effect. The effect is the result of pixels that are present in the motion image for a fixed amount of time due to the frame buffer used. This unwanted effect causes a much larger region of interest than the actual object size. A combination of the fast motion tracker used with a background modelling technique creates an accurate region of interest. The background model is only updated when new tentative regions are created by the motion detection process to maintain real-time speeds. These more accurate estimated region sizes are used when initialising the particle filter.

4.3.4 In summary

From the results it is clear that inconsistent labelling is problematic. The motion tracker does, however, allow a means of finding interesting objects to track. Any constant movement in a sequence of frames is picked up and motion history information is obtained. Using one of the background



Figure 4.3: Frames 380, 430, 480, 530, 580, 630, 680, 730 show the motion tracking of 2 cars and a group of people walking along the side walk.

Parameters	Descriptions	Values
Motion detection		
Motion history duration	Time a pixel is part of history	10 ms
Minimum region size	Smallest amount of pixels forming a region	30 pixels
Tracking Settings		
Tentatives region	Time needed to become trackable	5 ms
Deletion	Time needed for lost region to be removed	5ms
Search range	Matching regions between frames	30 pixels
Region growth	Size change allowed	50%
Colour model		
(h,s,v) bins	Histogram bin sizes	(10,10,10)
(HS-V) weight ration	Combining HS- and V-histogram	(0.7 : 0.3)
Alpha blending	Adaption rate	0.1
Hog model		
Histogram bins	Object rotation = 360/Histogram bins	18
Particle filter		
Sigma	Selection aggressiveness	0.1
Particles	Amount	100
FGD & MOG		
Minimum region size	Noise removal	15 pixels

Table 4.1: User defined parameters to tweak module performances.

detection schemes like MOG or FGD, segments of confirmed regions can be better identified.

4.4 User defined parameters

Each of the modules, motion detection, motion tracking, background modelling and particle filtering contains user defined parameters that could improve performance if selected correctly for a particular scene. This section summarises the most important parameters in Table 4.4. Also, values used in the experiments are the same unless stated otherwise in the text.

4.5 Object detection experimental results

As previously mentioned, the FGD and MOG algorithms are implemented in the OpenCV library. A background module interface is implemented and integrated into the system. Using this interface each algorithm is tested.

FGD and MOG background modelling techniques were developed for complex scenes containing changing backgrounds. Comparison of different video sequences shows that for static backgrounds, both methods perform similarly, giving excellent results. Problems using these techniques become quickly apparent. Tweaking parameters for each model needs to be done for different scenes, see Table 4.4. Also, model update speeds are slow due to the complexity and amount of calculations needed. Real-time processing is achievable only at low resolutions of 320x240 on hardware as described in Appendix A.5.

Figure 4.4 shows a frame from a video sequence where a person dressed in black walks across the scene. The scene is composed of dynamically changing background scenery, e.g. trees, while the foreground contains a person. The idea is to detect the person moving in the foreground against the dynamically changing tree branches in the background.

The foreground frames are shown for four different background modelling techniques: frame differencing, motion detection, FGD and MOG. Both FGD and MOG are trained for 700 frames beforehand to ensure a reliable background model is present. The following paragraphs explain the results obtained in Figure 4.4 for each of the four techniques.

Frame differencing is a simple approach for detecting foreground regions by subtracting consecutive frames and thresholding to minimise noise. For the results in Figure 4.4 (b), a running average, weighted sum of frame $I(x, y)$ is calculated,

$$I_k(x, y) = (1 - \alpha)I_{k-1}(x, y) + \alpha \times F_k(x, y). \quad (4.5.1)$$

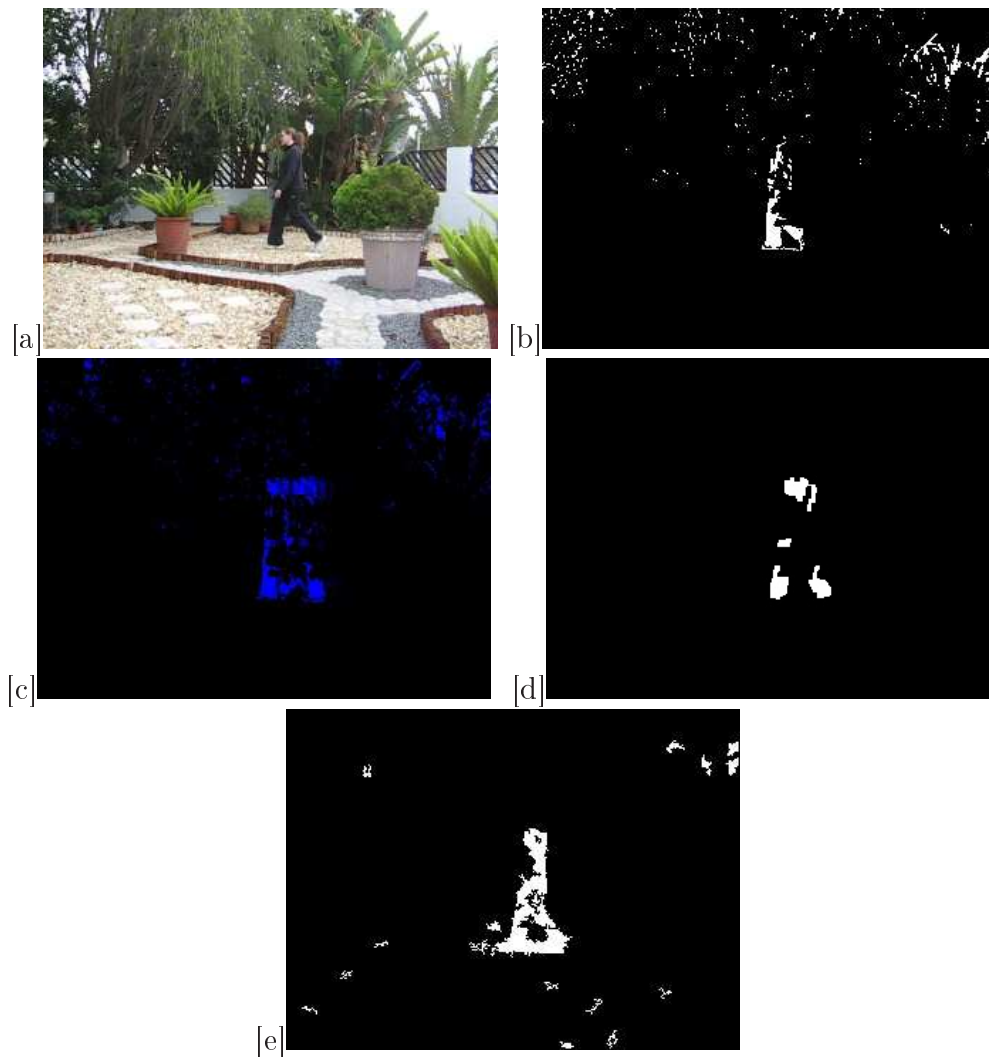


Figure 4.4: (a) frame 260, (b) frame differencing, (c) motion detection, (d) MOG, (e) FGD

Every new frame F_k is weighted by a constant value α at time k , in this case set to $\alpha = 0.1$. Each of the colour channels is thresholded and combined by a logic OR operator to form a binary foreground image. Throughout the video sequence noise regions are detected where the background moves and the foreground image is cluttered. Frame differencing has the advantage that it runs real-time and is appropriate for static scenes.

Motion Motion detection is shown in Figure 4.4 (c) and explained fully in Section 4.3.1. There is no background modelling, only foreground detection from moving objects. Moving shapes that come to rest are lost from view. Any moving foreground object is detected and an empirically chosen threshold, $T = 50$ (minimum blob size in pixels) is used in this sequence. Figure 4.4 shows that motion contains a significant amount of useful information for tracking purposes. It also shows how the background changed for a short period of time before the frames shown in Figures 4.4.

FGD Foreground detection, as described in Section 4.2.1, is tested using various input parameters. Frame rates of 30 frames per second is achieved at a resolution of 320x240. The foreground person is detected successfully in the video sequence shown in Figure 4.4 (d). FGD fails to detect the foreground object (person in Figure 4.4 (a)) in cases where the background pixels are indistinguishable from the foreground (black clothing over shadows). This is understandable since the model statistics cannot distinguish the foreground from the background when the pixel values are the same. Also, note that most of the moving tree branches in the background are successfully distinguished from the foreground.

MOG Similar to FGD, parameters need to be set according to the scene composition. Using 3 Gaussians, a speed of 25 frames is achieved at a resolution of 320x240. Foreground regions are detected with small noisy patches. Using the same minimum blob size as FGD, MOG detected more foreground pixels in this case, as seen in Figure 4.4 (e).

4.5.1 In summary

Both MOG and FGD can handle dynamically changing backgrounds and are well suited for complex scenes. Processing is computationally expensive and slows down more when scenes are busy. Setting up model parameters also require tweaking to obtain useable results. These background modelling methods were developed to deal with complex dynamic scenes. In cases

where scenes are static, foreground objects are much easier to detect using only the motion in a scene.

4.6 Particle filter tracking

Real and simulated videos are used to test the feature-based particle filter. Experiments are first done by selecting the object to be tracked by hand in the first frame. Then experiments are done by automatically obtaining objects and extracting features to track. The final experiment illustrates the integration of all the modules, background modelling, motion tracking and multiple feature-based particle filters.

4.6.1 Colour only tracking

Using only the object's colour feature, a simulated soccer video from dataset <http://www.multitel.be/trictrac> is tested. The object to be tracked is first selected by hand. The test illustrates the use of tracking confidence as well as the particle filter's multiple hypotheses when dealing with objects with similar appearance.

Soccer scene description The video sequence consists of soccer players, all similar in appearance, on a football field. In the sequence, the camera, moving quicker than the players, is panning from left to right. In each frame in Figure 4.5 the particles are shown as red circles tracking a player as well as the tracking confidence (top left). The tracking confidence is shown in terms of "tracking object" or "object lost". Each particle's weight is represented by the circle's radius, where the radius is directly related to the particle weight.

Detection of tracker loss In frame 348 the camera moves across the field past the players and the tracked player is lost. A clear indication of a tracker loss is seen when particles, equally weighted, are distributed with a large standard deviation. In the sequence of frames different players



Figure 4.5: Selected frames 301,315,319,322,348,353,371,571 show the tracking of simulated similarly clothed players using a moving camera.

are tracked when the player is lost from view and the initialisation detects another player that has a high likeliness to the target model.

It is clear that particles are distributed between players that are close in distance, a clear indication of multiple hypothesis of the pdf distribution properties. This can be seen in Figure 4.5 frame 322 where the two players on the left are both weighted by particle circles after the automatic initialise function detected the players.

4.6.2 Texture only tracking

HOG is used as a feature model in this experiment to track a person's eyes and nose (see Figure 4.6). The experiment illustrates a real person's face being tracked. The confidence measure is tested for the HOG feature model.

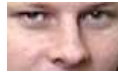


Figure 4.6: Face image used to initialise tracking

The red circles, in Figure 4.7, have a radius directly related to their probability and are well distributed across the face. The predicted location, shown by the ellipse bounding the most likely position, is not an accurate prediction in every frame. Through the sequence the tracking confidence stays low and a message is displayed showing that the object is considered lost. However, the face is tracked for the entire sequence. In Figure 4.7 the number of particles are set to 150 and a 72 bin histogram is used. Both the number of particles and the number of HOG bins are increased greatly to obtain the results shown. Performance is negatively affected by the increase in particles. Real-time tracking is not achieved since feature extraction of the face regions for every particle is slow.

4.6.3 Feature combination tracking

Both features, colour and texture, are used during tracking. Again, tracking a person's face is considered. The result is compared with the HOG tracking in the previous section. The soccer sequence is not used in to illustrate the combined features. The soccer players texture information is very little, due to the uniform colour of the players, and tracking results do not improve.

To increase the confidence measure of texture tracking the colour feature is included. This allows for less particles to be used to increase speed. Also, the increase in tracking confidence allows that the region size can be adjusted appropriately. The number of particles is set to 75 using a colour histogram descriptor of 10x10 bins for HS-histogram, 10 bins V-histogram and a HOG descriptor of 36 bins. Each feature is weighted equally. Performance and accuracy is greatly increased and the facial region is tracked with high confidence as seen in Figure 4.8. Note the high accuracy of the most likely positions during the end of the sequence where both colour and shape become distorted.

4.7 Automatic initialisation tracking

The feature-based particle filter requires prior knowledge of features, whereas motion tracking as implemented in Section 4.3, captures any region of moving objects. This section describes how to integrate motion detected objects, representing them as features, to the feature-based particle filter.

Other initialisation techniques, which rely on features being trained beforehand, are possible. Support vector machines for example are used to obtain an object's features from a large descriptive dataset of similarly appearing objects. These methods work well if the object that needs to be tracked is specifically known. This is possible when designing a specific application. Using motion tracking, a general approach is taken to illustrate the power of a feature-based particle filter to track objects.

4.7.1 Snooker ball experiment

To illustrate the integration of each of the module's background modelling, motion tracking and particle filtering, a sequence of snooker balls are tracked. No prior knowledge is used to train the features or the background model. Tracking snooker balls, each with its own instantiated particle filter, is illustrated in this section. Tracking snooker balls is considered to maintain a controlled environment.

The video sequence contains 3 snooker balls placed at the end of the table. New snooker balls enter the frames periodically. When a new snooker ball enters the camera view, its motion is captured and tracked using motion tracking. Once the region is confirmed, the background model is used to find the minimum bounding box containing the snooker ball for accurate feature extraction. The bounding box is used to initialise a particle filter to handle further tracking. Different coloured balls are used since ball shapes are identical and HOG would fail when used alone. The snooker ball experiment is shown in Figure 4.9 and illustrates the following:

- Automatic initialisation is possible using the motion tracker
- Due to modular design multiple objects can be tracked simultaneously
- Necessary good feature extraction is handled by a background model
- Adapting to the region size automatically (motion direction is away from camera)

Details of the snooker sequence A description of the video sequence is required to understand the experiment completely. The video resolution is 320x240 with the light source above and behind the table. Illumination changes are present when the balls move across the table beneath the light. The illumination changes are handled by the colour feature adaption. Also, the light source creates pixel noise detected by the motion detection. Ball speeds vary but all enter the scene quickly. The balls are rolled hard enough by hand to reach the opposite side of the table and bump into other balls

on the way back. The effect of this is that motion detection, detects a much larger region when the ball enters due to the motion pixel history described in Section 4.3.1. Also, it is important to note that as the ball moves away from the camera its size also becomes much smaller.

Parameter settings In Figure 4.9 the bounding box regions are drawn in black. The number of particles are set to 60, using both HOG and colour features blended at 50%. The motion tracker is set to confirm tentative regions in 0.5 seconds. The maximum number of particle filters that may be instantiated is set to 8. Automatic initialisation after a loss of an object, which is very time consuming, is disabled to avoid interference with the motion tracker timing. A MOG background model is used whenever new regions are confirmed. A FGD can also be used with similar results, however with slightly slower execution.

Result discussion The first blue snooker ball is detected from frame 55. The movement is mostly away from the camera and its bounding box decreases quickly. The bounding region is mostly adapted in frames 74, 75 as seen by the size of the white circles corresponding to the particle's location with the highest probability.

The introduction of new snooker balls in frames 74, 85, 103 is each handled separately. Features are extracted while the balls are moving at their quickest, not always bounded exactly and where shadows are also included. These features are slowly adapted and most balls are tracked successfully, although two do become lost. The effect of the HOG feature does, in some cases where the colour ball is lost, allow other colour balls to be tracked and the colour adapts to the new ball colour (such is the case with the black ball).

Notice that the motion tracking bounded regions, shown in green (tentatives) and yellow (confirmed), are much larger than the actual target

objects. A much more accurate bounding region is thus found from feature tracking.

There are slight illumination variation as the balls move across the table and closer to the light source. To handle the illumination changes the colour feature model needs to adapt accordingly. Adaption is successful in all but 2 cases, the blue and black ball is not tracked the entire sequence. The reason the balls are lost is the result of bad feature representation. The similarity with the side and back of the snooker table where the edge of the table casts a shadow has a high similarity with the feature and the black ball is not tracked. In the case of the blue ball, another ball (purple ball) with similar colour, is tracked when the blue ball moves into the table's edge shadow (seen in frame 112).

Using eight particle filters is computationally expensive and the sequence of snooker balls are not tracked real-time. When only one or two balls are tracked using the particle filter, real-time tracking is possible.

4.8 Summary and conclusion

In this chapter, the development of modules, background modelling, motion tracking and feature tracking are integrated and tested. Results show that when parameters are setup correctly and descriptive model features are selected, then tracking is successful. Table 4.8 shows a summary of timing results for experiments in this chapter. Results are based on hardware described in Appendix A.5.

Generally, in situations where the object is detected using motion detection, descriptive features are either not present or not automatically selected well. It is important to note that tracking only performs as well as the initial model selections. The snooker ball tracking experiment works well, due to simple but descriptive features, little background noise and a semi-controlled environment.

The goal of creating real-time methods is achieved. Each module can operate in real-time when used on its own. However, when integrating the modules the process slows down significantly. Background modelling speeds are mostly subject to how busy a scene is, while the particle filter speeds are mostly affected by the region size that need to be processed for each particle to extract features. Using the very fast motion tracking background modelling is only necessary for newly confirmed regions and greater speeds are obtained.



Figure 4.7: Face tracking video sequence using HOG with 150 particles and 72 bins. Selected frames 55, 100, 125, 150, 180, 212, 300, 336 shown.



Figure 4.8: Face tracking video sequence using colour and HOG features with 75 particles. 110 colour bins and 36 HOG bins. Selected frames 55, 100, 125, 150, 371, 372, 373, 374 shown.



Figure 4.9: Selected frames 55, 57, 74, 75, 85, 88, 103, 112, 128, 129, 143, 201 shown from top left to bottom right.

Experiment name	Features used	Fps	# particles	image/patch size(pixels)
FGD	×	33	×	320×240
MOG	×	25	×	320×240
Motion	×	35	×	640×480
Soccer sequence	Colour	10	150	10×20
Face 1	Texture	7	150	75×45
Face 2	Combination	5	75	75×45
Snooker	Combination	0.3	70×8=560	varies

Table 4.2: Summary of experimental timing results

Chapter 5

Conclusion

The problem was to investigate a means of tracking that could be extended to various fields for the purpose of general use.

A sub-optimal general particle filter was implemented that can be used in different tracking applications. Other, more accurate trackers do exist and are much more complex, but are just too computationally expensive. Incorporating feature descriptors allowed objects to be modelled and video sequences to be analysed. Previous work suggests that colour-based features obtained good results; the colour-based feature was implemented. It also became clear that using other features increased robustness, and histograms of gradient (HOG) were implemented suitably adapted for tracking purposes. Re-initialisation after tracker loss and a confidence measure was added to increase robustness during adaption. Dynamic and object information was obtained using the motion tracker and automatic initialisation was implemented. Noise needed to be removed before capturing good descriptive features and background modelling techniques partially solved this problem.

5.1 Future challenges

Some challenging scenarios exist where tracking will fail in most cases. Interestingly, in general, indoor locations are more difficult than outdoor. Indoor locations have many surfaces that reflect or cast shadows. Also, electric lighting creates more challenges, such as noise, and objects reflections and shadows are more pronounced. This poses a challenging task to accurately extract features only relating to the object. Tracking during night-time is another challenge that requires detailed attention to illumination sources.

5.2 Restrictions

It is important to realise that each module, such as motion tracking, background modelling and feature tracking, have their own restrictions. Both motion tracking as well as background modelling, are restricted by a static camera setup. Also, much time is needed setting up parameters or waiting for background models to be trained correctly. In most cases, settings are scene-and-object specific.

Particle filtering is considered due to its simplicity to easily track objects in real-time. However, it was found that there are hidden computational costs. The feature-based particle filtering is computationally expensive for multiple features. Execution time suffers when the number of particles is increased or the image patch for each particle is very large. Most of the computational time is spent on extracting features for each observation which is equal to the number of particles. In such cases the requirement of real-time tracking is not met. Also, tracking fails when models are a bad representation of the object.

5.3 Goals achieved

An adaptive feature-based tracker was implemented and tested. Having a tracking confidence meant that features could be adapted automatically and smartly. These tools allow robust tracking of any object that meets the restrictions. These restrictions depend on their model descriptors and better models would increase robustness or execution speeds. Prior knowledge of an object's descriptor was obtained using a motion tracking technique based on repeated detection in each frame. Improvement of the prior model obtained from the detection, used dynamic background modelling techniques to obtain more accurate bounding boxes, greatly improving results.

5.4 Recommendations

Real-time execution is always a high priority in certain applications. What is more important is the cost effectiveness in the sense of being able to drive multiple cameras from the same CPU. Even the light-weight tracking methods discussed here are hard to implement in real-time. Detailed attention to optimisation is required. Many of the algorithms might also have much simpler or better ways that as a whole, minimise computations. It is suspected that such optimisations are possible and beneficial if applications are to become useful. Our studies do show however, that light-weight schemes can be quite robust and have the advantage of being adaptable to more specific applications. Research into application development is most likely the next logical step. Application specific solutions would also show much better results and have its own set of optimisations for robustness and speed. This thesis has presented a tracking tool that is easily extendable using different features. An application might now be developed that solves a much more specific problem using the tools developed here.

Appendices

Appendix A

Project files

In this section project files, installation and file interaction are described. All software libraries used are freeware or open source and cross-platform (developed and tested in Linux).

A.1 Required software

Development was done in a Linux environment on an Ubuntu Dapper system. Eclipse version 3.1.2 with a Cdt (*C++*) plug-in is used as editor for the tools. A pydev (python 2.4.3) plug-in is used for the interface and requires wxpython (2.6.2.1) to be installed. The Eclipse workspace contains all the compile settings, project files, images and videos used for development and compiled with gcc version 4.0.3 (Ubuntu 4.0.3-1ubuntu5). OpenCV Intel libraries (1.0.0) as well as the ffmpeg (libavutil, libavcodec, libavformat) libraries are required. Packages such as libjpeg are needed to display and save images. Note that the versions only indicate how the built binaries were constructed. In general always install newest versions when building from source.

A.2 Installing tracking tools

Running binaries An interface was created to easily enable or disable modules for testing. Each module setting is also included and is saved to a `config.txt` file. To run the precompiled binaries, simply copy the project directory and run `<./PF/src/ python interface_params.py>`. To save images and videos to the disk write access is required. In Linux log on as a superuser or execute the above command with a `sudo` prefix.

Building from source Building from source is easiest using Eclipse. Import the project workspace, all my compiler settings and directory structures are included. Select build all from menu and select run. Otherwise, compile using the `<makefile.from.source>` file.

A.3 Running an example

Open the GUI from the main project directory, `<./PF/src/ python interface_params.py>` and select from the start menu, run demo. The main GUI window is shown in figure A.1. There are 8 quick demo's to view. If at any stage it seems that a demo window is not responding hit any key. To exit hit the ESC key.

A.4 Directory structure and file description

Figure A.2 shows how the project files interact.

GIO.cpp Graphical input and output functions. All drawing functions, opening and saving of images and video stream functions are included in this file.

Backgroundmodel.cpp Both the Mixture of Gaussians and Foreground Detector is included and uses the opencv libraries background model.

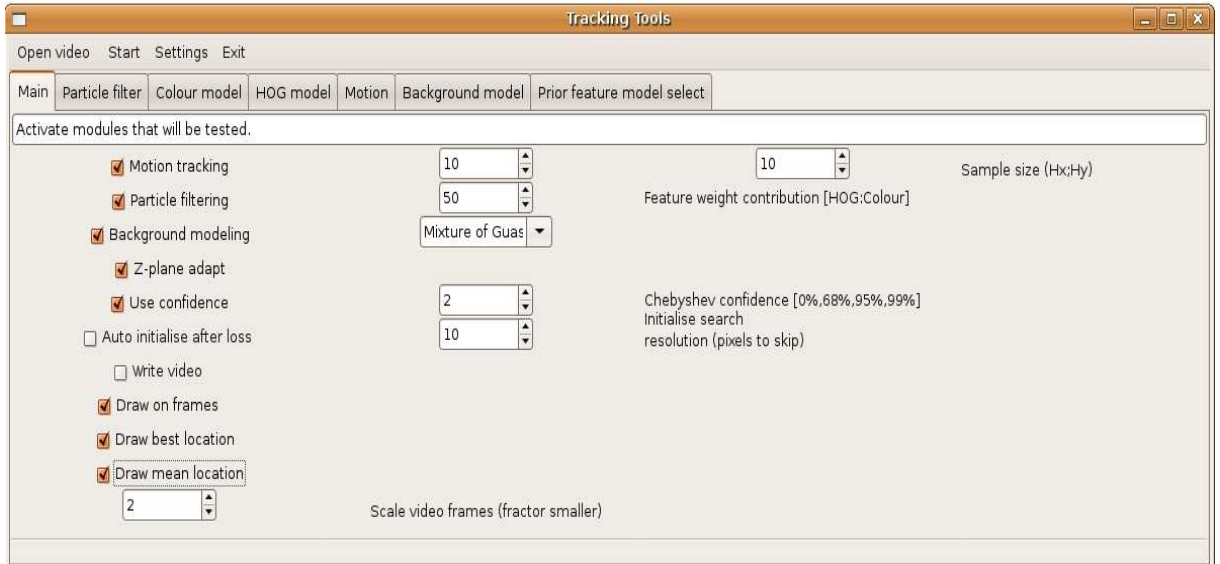


Figure A.1: GUI main window

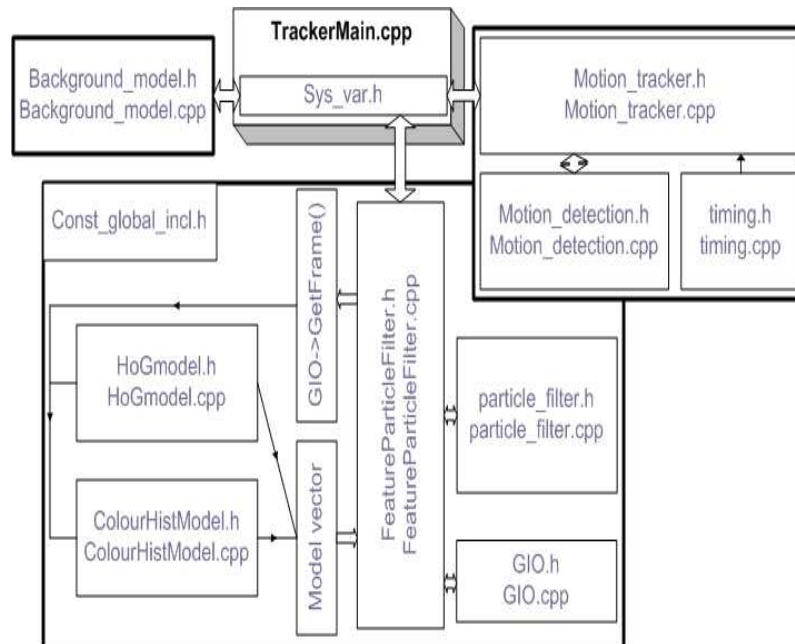


Figure A.2: Project file interaction

Timing.cpp Contains timing calculations to start, stop and get elapsed time functions.

MotionDetection.cpp Implementation of motion detection. Updates motion model using `update_mhi` function.

MotionTracker.cpp Processes the regions detected by motion detection (function `process()` is used). Labels are added to the regions and timing is used to obtain region which are present for constant time periods. `FilterOverlappingRegions()` function merges rectangles which intersect.

ParticleFilter.cpp Contains functions to calculate the pdfs and statistics from the distribution.

HoGmodel.cpp Texture feature is implemented using HOG. Histogram texture features are extracted from a colour image.

ColourHistmodel.cpp Colour image histograms are built for HS and V planes of the image.

FeatureParticleFilter.cpp Contains the tools that adapt and calculate confidence. The tracking steps, update pdf from samples and predict using motion models are included.

TrackerMain.cpp Contains the main loop that parses input parameters, open the video for display and calls each module.

Sys_var.h System and module settings and variables.

Const_global_incl.h The partile structure and other global constants.

Processor	Intel Core 2 T5500
Ram	1GB DDR2
Camera 1	Sony DSC-V1 320x240, 640x480 video (15 fps)
Camera 2	Sony DSC-H1 320x240, 640x480 video (15 fps)
Camera 3	Axis 221 320x240, 640x480 (up to 60 fps)
Camera 4	Axis 207 320x240, 640x480 (up to 30 fps)

Table A.1: Hardware used in experiments

A.5 Hardware configuration used

Experimental results were obtained using the following hardware in Table A.1. Different cameras were used to test the effectiveness of the algorithms on different hardware. Camera 1 and Camera 2 are both digital cameras with video functionality. For results shown in this thesis Camera 2 was used for the face tracking, background modelling experiments. And Camera 3 was used for the motion tracking experiment.

References

- [1] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. In *IEEE Transactions of Signal Processing*, Vol. 50(2), pages 174–188.
- [2] M Bolić, P.M. Djurić, and S. Hong. Resampling algorithms for particle filters: a computational complexity perspective. *EURASIP J. Appl. Signal Process.*, 2004(1):2267–2277, 2004.
- [3] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *Lecture Notes in Computer Science*, 1407:484–488, 1998.
- [4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society.
- [5] A. Doucet. On sequential Monte Carlo sampling methods for Bayesian Filtering, 1998.
- [6] S. Fleck and W. Strasser. Adaptive probabilistic tracking embedded in a smart camera. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*, page 134, Washington, DC, USA, 2005. IEEE Computer Society.
- [7] Alexandre Francois Institute. Real-time multi-resolution blob tracking.
- [8] A. Jepson, D. Fleet, and T. Maraghi. Robust online appearance models for visual tracking. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1296–1311, Washington, DC, USA, 2003. IEEE Computer Society.

- [9] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection, 2001.
- [10] B Leibe and B Schiele. Analyzing appearance and contour based methods for object categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, Madison, WI, June 2003.
- [11] L Li, W Huang, I.Y.H. Gu, and Q. Tian. Foreground object detection from videos containing complex background. In *Proceedings of the 11th ACM International Conference on Multimedia*, pages 2–10, November 4-6 2003.
- [12] Liu, Jun S. and Chen, Rong. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, sep 1998.
- [13] M.Isard and A.Black. *Active contours*, pp 259-262. Springer, 2000.
- [14] K. Nummiaro, E. Koller-Meier, and L. Van Gool. A color-based particle filter. *First int. Workshop on Generative-Model-Based Vision (GMBV'02)*, 2002.
- [15] Y. Raja, S. J. McKenna, and S. Gong. Tracking and segmenting people in varying lighting conditions using colour. In *Proceedings of the third International Conference on Face & Gesture Recognition*, page 228, Washington, DC, USA, 1998. IEEE Computer Society.
- [16] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman filter: Particle filters for tracking applications*, p35-65. Artech House, 2004.
- [17] D. Salmond and N. Gordon. An introduction to particle filters. September 2005.
- [18] J. Shi and C. Tomasi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593-600, 1994.
- [19] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 246–252, Los Alamitos, June 23–25 1999. IEEE Computer Society.

- [20] D. Thirde, M Borg, J. Ferryman, J Aquilera, M. Kampel, and G. Fernandez. Multi-camera tracking for visual surveillance application. *Computer Vision Winter Workshop*, 2006.
- [21] M. Xu, L. Lowey, and J. Orwell. Architecture and algorithms for tracking football players with multiple cameras. *In Proceedings of the IEEE Workshop on Intelligent Distributed Surveillance Systems* , pp. 51-56, London, UK, February, 2004.