Abstract

Although object tracking is not a new concept, it is always a hot topic and an important task in the field of computer vision. The design and implementation of a robust tracking system which is able to deal with the various situations is a real challenge for every programmer and researcher who is interested in such areas.

In this paper, an efficient colour-based multiple object tracking system in the particle filter framework is presented. In order to accelerate the tracking system while preserving its robustness, two enhancements are introduced in this project. The first one is the use of a background suppression technique for reducing the influence of the background noise. The second improvement is to facilitate the tracking system by optimizing the particle sampling process. We can dominate the species diversity of the samples in order to get the optimized tracking result by changing the propagation rate of the important samples according to the situations. The multithreaded tracking system is demonstrated in the context of the tennis and squash matches and the related experiment results are given in the following paper.

For My Parents

终于可以回家了！

Table of Contents

Chapter 1 - Introduction 7

1.1 Aims and Objectives 7

1.2 Organisation of the Dissertation 7

Chapter 2 - Backgrounds and Related Work 9

2.1 Tracking Methods 9

2.2 Bayesian Framework 11

2.2.1 Recursive Bayesian Estimation 12

2.2.2 Kalman Filter 13

2.2.3 Particle Filter 14

2.3 Summary 16

Chapter 3 - System Design and Implementation 17

3.1 Requirement Specification 17

3.2 Initialisation 17

3.3 Transition Model 19

3.4 Colour-Based Observation Model 21

3.5 Particle Sampling 25

3.6 Multithread Support 27

3.7 System Architecture 28

3.8 Summary 29

Chapter 4 - Experiments Result 30

4.1 User Interface and Guidance 30

4.2 Organisation of Experiments 30

4.3 Experiments 31

4.3.1 Single Object Tracking 31

4.3.2 Multiple Objects Tracking - 1 32

4.3.3 Multiple Objects Tracking - 2 33

4.3.4 Tracking partially occluded objects - 1 34

4.3.5 Tracking partially occluded objects - 2 36

4.4 Experiment Review 38

4.5 Summary 38

Chapter 5 - Future Works 39

5.1 Limitation and Possible Extension 39

5.2 Summary 41
Chapter 6 - Conclusion 42
Reference 43
Appendix: Sample Source Code 46

Chapter 1 - Introduction

In this paper, we present a colour-based tracking system, which is built on an enhanced particle filtering framework. From the probabilistic perspective, unlike the Kalman filter which relies on linear and Gaussian assumptions, the particle filter method makes use of the sequential stochastic sampling process to achieve the object tracking. This is an ideal choice to the nonlinear and non Gaussian environment.

Object tracking system plays a crucial role in many computer vision applications. The design and implementation of a robust tracking system which uses less computing resources and runs at a high frame rate is a very challenging task. Tracking non-rigid objects can be particularly difficult. The problem is compounded further if the varying objects are very small, arranged in a cluster, or presented in a noisy background. To accelerate the tracking system while preserving its robustness, two enhancements are adopted in this project. The first one is the use of a background suppression technique for reducing the affect of the background noise. The second is used to improve the tracking performance by optimizing the particle sampling process and multithreading the tracking system. The detailed descriptions about why and how these two enhancements are implemented are given in the following paper.

1.1 Aims and Objectives

The primary goal of this project is to implement an efficient colour-based tracking system, which is competent at tracking objects at a high frame rate. The tracking system must be capable of tracking objects that are both small and non-rigid. Lastly, the tracking system must be able to deal with objects that are presented in a cluster and against a noisy background. Along this goal, our work is divided into three parts:

- Designing a background suppression method to reduce the effect of the background noise and optimizing the colour based observation model through the experiments.
- Improving and facilitating the Sequential Importance Re-sampling (SIR) algorithm.
- Multithreading the tracking system in order to get a high performance on the modern multi-core or multiprocessor systems.
- handling partial occlusions.

1.2 Organisation of the Dissertation

The paper is organized as follows:

- ***Chapter 1 - Introduction.***
- ***Chapter 2 - Background and Related Works*****:** Discussing the backgrounds and related works which are found in the technical literatures and a detailed description their relevancies to the project is given in this chapter.

- ***Chapter 3 - System Design and Implementation*****:** Describing and explaining the theoretical and the practical aspects of the project.
- ***Chapter 4 - Experiment result*****:** Reporting on the experiment results and the achievements of the project.
- ***Chapter 5 - Further Works*****:** Discussing the limitations of our tracking system and the possible extensions
- ***Chapter 6 - Conclusion*****.**
- ***References*****.**
- ***Appendices: Sample source codes*****.**

Chapter 2 - Backgrounds and Related Work

The object tracking is a process of detecting and locating multiple moving targets of interest in a video sequence. The main challenges faced in designing and implementing a robust tracking system are how to efficiently reduce the interferences in a noisy and clutter environment and how to accurately identify the motions and the locations of the non-rigid objects in a variety of situations. Over the last few years, many tracking methods have been introduced. In the following section, we list several of them which are used commonly.

2.1 Tracking Methods

According to the comparison of the types of the techniques used, the results of the research as described below, presents several types of commonly used tracking methods. Normally, the methods listed below employ an approach which falls under the Bayesian framework such as Kalman filter and Particle filter to achieve the object tracking. The Bayesian framework and related algorithms are discussed in section 2.2.

- **Tracking based on image segmentation technique**
  Foreground object detecting processes are based on some form of the background subtraction techniques. Firstly, the new frame inputted is segmented into the foreground (moving objects) and background (stationary area) by computing the differences with a reference frame [1] or a set of the accumulated frames [2] [3]. Then, a set of variables are computed which represents the changes of the moving object's state, for instance, the object' position, size and many other properties.

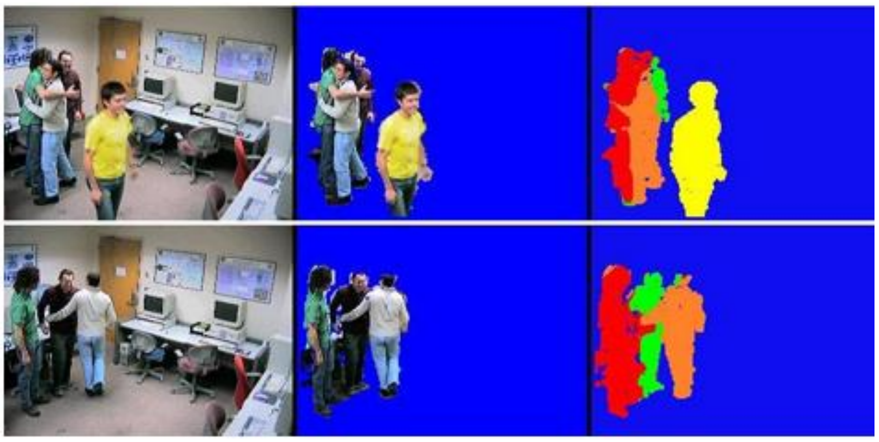


Figure 1, taken from Ref. [1]. Background Segmentation.

As shown in the figure 1, the foreground region comparison is able to achieve high reliability because it uses the global information of the tracking targets. However, its main drawback is that the computation process is an expensive and time consuming task, for instance, the noise reduction process such as removing the dynamic shadows. This problem is worse if the image size or the search area is very large. Additionally, this kind of method relies on the prior knowledge about the background and requires the appearance of the tracking targets not to have significant changes. For instance, the object's colour information could vary over time due to a change in illumination such as the blue colour on the object could be rendered as black in low-light situations resulting in becoming a part of the background accidentally. If this potential problem is not taken into consideration the ability of the tracking algorithm may suffer.

- **Contour based tracking method**
  The typical approach is Active Contour Model (ACM), also known as Snake Model. A snake is "an energy minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges" [4].

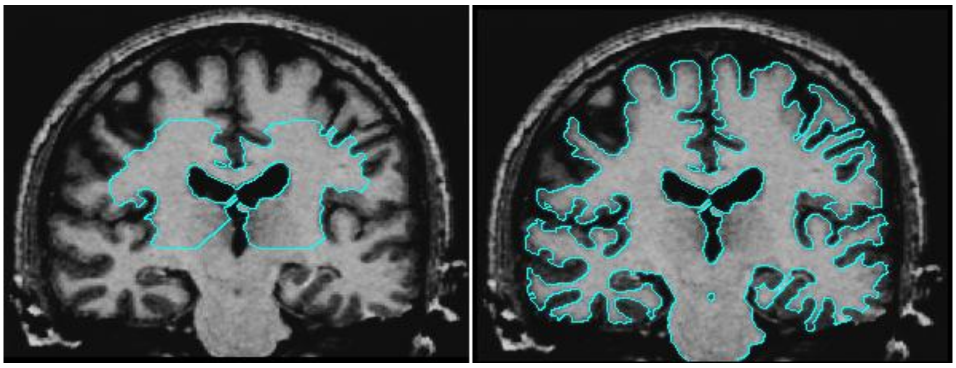


Figure 2, taken from Ref. [5]. Brain scan - the snake (bright blue colour) expands and forms a smooth contour of the brain.

By extracting the information about the features of the target, the object tracking can be achieved well in the cluttered background and the accuracy of the tracking is insensitive to the noise of the illumination [6]. This method is widely used in action interpretation systems as shown in the Figure 3.

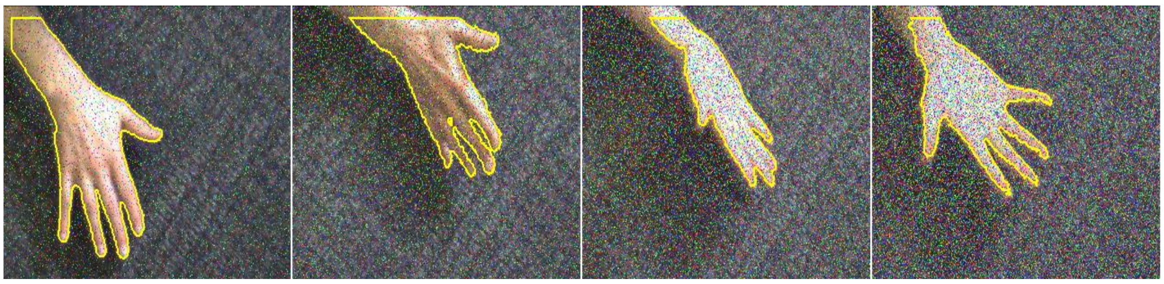


Figure 3, taken from Ref. [7]. Contour based hand tracking.

- **Feature point based tracking method**
  In an image, the moving objects can be represented by their feature points which can be extracted by using the Scale Invariant Feature Transform (SIFT) method. The SIFT is a popular image matching technique designed to compute the geometrical transformations of the images. We can monitor the changes of the object's state in two successive frames by measuring and matching the displacements of a set of grouped feature points of a object. This method is greatly used in the applications such as 2D or 3D facial tracking.

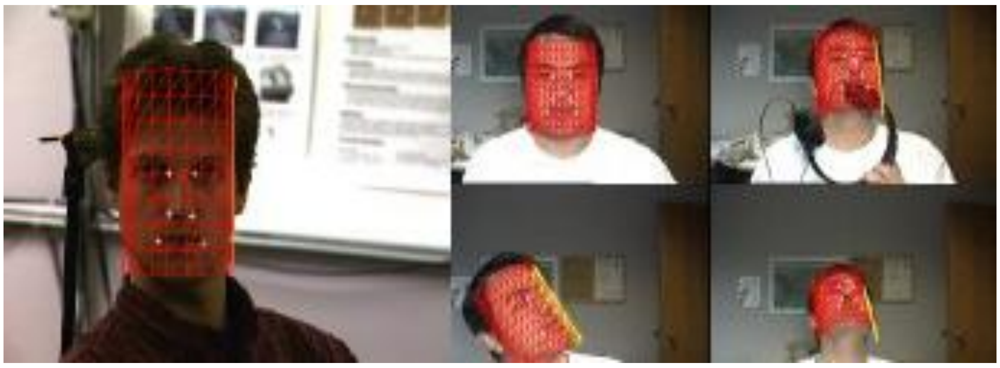


Figure 4, taken from Ref. [8]. 3D facial tracking (each node represents a feature point).

Because the feature points distribute throughout the targets, the tracking can still be continued even if some are partially blocked or partially disappear. The most critical issue with this kind of approach is that it is difficult to extract and update the properties of the feature points when unpredictable object motions occur. For instance, the object tracking could be very difficult if a target rotation occurs because most part of the feature points will disappear and new points will appear.

2.2 Bayesian Framework

This section gives a brief introduction to Recursive Bayesian Estimation, which is the theoretical basis of this project. When considered from the probabilistic point of view, visual tacking can be considered as predicating the possible locations or motions of a target of interest in a continuous time series. The target can be described by its set of parameters that represent its state at a time step, such as the size, colour and position. Therefore, object tracking can be further understood as equivalent to analysing the evolution of an object's states.

Nowadays, there are more visual tracking approaches based on Bayesian theory, for instance, Kalman filter and Particle filter. In the framework of Bayesian theorem, the state estimations as probabilistic inferences can be obtained by using a sequence of measurements, normally using the state space approach to analyze and describe the object tracking procedure [9].

The basic idea of Bayesian approach is that it converts the object tracking problems to recursive Bayesian state estimations - initialize priori probability density of a target's state, then continue to measure the posterior probability when the new knowledge (observation) is available. In other words, there is a causal relationship - the current state of an object only depends on its previous state and the current state will cause the measurement of the subsequent state at the following time step (First order Markov process). We can use the knowledge obtained from the observation along with the object's current state to compute the new state estimation of the next time step.

2.2.1 Recursive Bayesian Estimation

The state estimation is expressed as a pair of recursive equations in the time order which are named as 'prediction equation' and 'update equation' [9]. The state of a tracking object at time k ($k \in N$) is denoted by x_k, and the observation obtained at time k is denoted by y_k. Suppose that the previous posterior probability density function (PDF) $p(x_{k-1} \mid y_{1:k-1})$ is already known. The state transformation is a Markov process:

$$\text{Transition model: } p(x_k \mid x_{k-1},\ y_{1:k-1}) = p(x_k \mid x_{k-1}) \qquad (1)$$

The prior PDF at time k can be computed by using the Chapman-Kolmogorov equation:

$$p(x_k \mid y_{1:k-1}) = \int p(x_k \mid x_{k-1}) p(x_{k-1} \mid y_{1:k-1}) dx_{k-1} \qquad (2)$$

And the posterior PDF at time k can be computed as follows:

$$p(x_k \mid y_{1:k}) = \frac{p(y_k \mid x_k) p(x_k \mid y_{1:k-1})}{p(y_k \mid y_{1:k-1})} \qquad (3)$$

$p(y_k \mid x_k)$ is called the observation model, also known as the likelihood function which used to update the current state's prior PDF to obtain its posterior PDF. $p(y_k \mid y_{1:k-1})$ is a normalization constant given by:

$$p(y_k \mid y_{1:k-1}) = \int p(y_k \mid x_k)\, p(x_k \mid y_{1:k-1}) dx_k \qquad (4)$$

The equations (2) and (3) shown above construct the basis of the Bayesian approach. By substituting (2) and (4) into (3), the posterior PDF $p(x_k \mid y_{1:k})$ at time k can then be computed as follows:

$$\underbrace{p(x_k \mid y_{1:k})}_{\text{Posterior distribution}} = \underbrace{\underbrace{p(y \mid x_k)}_{\text{Observation model}}}_{\text{Update equation}} \underbrace{\int \underbrace{p(x_k \mid x_{k-1})}_{\text{Transition model}} \underbrace{p(x_{k-1} \mid y_{1:k-1})}_{\text{Previous posterior distribution}} dx_{k-1}}_{\text{Prediction equation (Current prior distribution)}} \qquad (5)$$

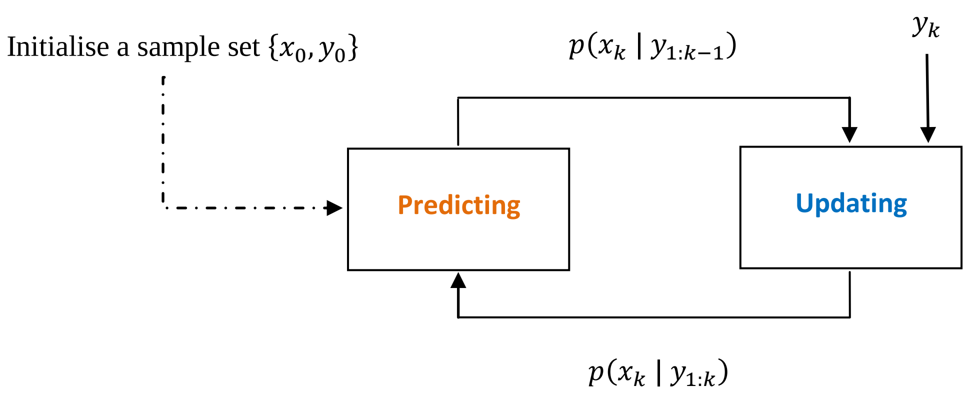


Figure 5, Recursive Bayesian state estimation loop.

Updating is an important process, which can be understood as a corrector used to indicate the similarity (or difference) of the predicted state and the observation at time k, thus it directly affects the accuracy of the tracking result.

2.2.2 Kalman Filter

Kalman Filter is a mathematical tool used to estimate the states of a process in a time sequence and was introduced in 1960 by R.E Kalman [10]. Over the last 50 years, the Kalman Filter and its extensions as state estimators have been wildly used in many subjects, especially in the field of computer vision. It is an efficient and optimal recursive estimator for linear systems.

The state x at time k can be represented by the following equation:

$$x_k = Ax_{k-1} + v_{k-1} \tag{6}$$

The measurement y at time k can be represented by the following equation:

$$y_k = Bx_k + w_k \tag{7}$$

A and B are transition and measurement matrix, which are possible time variant. The variances which are denoted by v and w respectively represent the transition and measurement noise at time k. Both of them are Gaussian distributions with zero mean and a deviation matrix Q and R:

$$p(v) \sim N(0, Q) \tag{8}$$

$$p(w) \sim N(0, R) \tag{9}$$

For instance, we track a ball which is moving forward along x-axis on a 2D plane, suppose its size won’t be changed, thus its possible x location at the next second is:

predicted_x_loc = (previous_moving_speed + current_x_loc) + acceleration_noise;

Obviously, the model of the Kalman filter is highly linear and deeply affected by the Gaussian noise - $x_k = Ax_{k-1} + v_{k-1}$ is a linear transition from x_{k-1} with the Gaussian noise v_{k-1}. Although the Kalman filter is an optimal estimator for the linear systems. However, in the real world, an object’s movement is non-linear and non-Gaussian hence tracking failure may occur caused by a large deviation in the state estimations.

Over last few years, many attempts have been made to cope with the nonlinearities. Several extensions to the Kalman filter have been proposed, with the most successful one being the Unscented Kalman filter. Such extensions use unscented transform techniques [11], which use a set of sigma points to approximate the non-linear state estimation.

Instead of using transition and measurement matrix, the Unscented Kalman filter uses two non-linear function f and h - the function f is used to compute the predicted state from the previous state and the function h is used to compute the measurement for updating the prediction.

$$x_k = f(x_{k-1}) + v_{k-1} \tag{10}$$

$$y_k = h(x_k) + w_k \tag{11}$$

2.2.3 Particle Filter

Particle filter, based on the Bayesian framework combined with Monte Carlo stochastic simulation method [9], has been proven to be very successful in dealing with general non-linear and non-Gaussian tracking problems. The object tracking can be achieved through continuously approximating the true posterior probability density of the object’s state by using a set of weighted samples which are called particles. In a tracking process, every particle represents a hypothesis about the object’s next state and its weight can be considered as the likelihood of a state hypothesis.

The main drawback of the Particle filter is that it requires a large number of samples to achieve high accuracy and stable object tracking. This factor could make the analysis process computationally expensive, resulting in the low frame process rate. To overcome this problem, we plan to use the multithread technique to achieve the optimal use of the computational resources of the modern multi-core processors.

The common issue faced by the Particle filter is the particle degeneracy problem. To overcome this problem, a subsequent extension to the Particle filter introduced a Sequential Importance Re-sampling (SIR) stage to avoid the important weights only apply to one or a very small number of the particles and eliminate the useless particles which have the negligible weights.

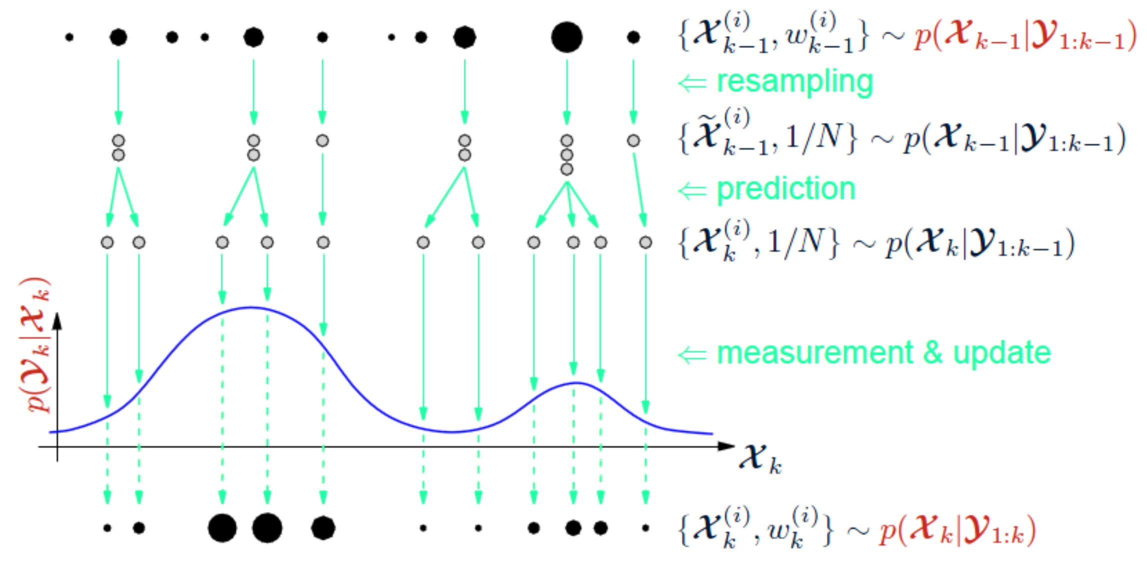

$\{\mathcal{X}_{k-1}^{(i)}, w_{k-1}^{(i)}\} \sim p(\mathcal{X}_{k-1}|\mathcal{Y}_{1:k-1})$
⇐ resampling
$\{\tilde{\mathcal{X}}_{k-1}^{(i)}, 1/N\} \sim p(\mathcal{X}_{k-1}|\mathcal{Y}_{1:k-1})$
⇐ prediction
$\{\mathcal{X}_{k}^{(i)}, 1/N\} \sim p(\mathcal{X}_{k}|\mathcal{Y}_{1:k-1})$
⇐ measurement & update
$p(\mathcal{Y}_k|\mathcal{X}_k)$
$\mathcal{X}_k$
$\{\mathcal{X}_{k}^{(i)}, w_{k}^{(i)}\} \sim p(\mathcal{X}_{k}|\mathcal{Y}_{1:k})$

Figure 6, taken from Ref. [12]. Sequential Importance Re-sampling (SIR).

The figure 6 illustrates the Sequential Importance Re-sampling (SIR) process. The drift represents the posterior distribution. The blobs represent a set of weighted particles and the size of the blob indicates its weight. The particles with the large importance weights will generate more copies of themselves and all of them will be assigned a same value (1/N, N is denoted as the number of the particles) at the end of the RESMAPLING stage. On the contrary, the particles which have light weights will be eliminated. The values of the particles' weights will be evaluated and updated at the MEASUREMENT stage and then repeat SIR process at next time step if necessary.

As shown in the figure 6, the Particle filter can be understood as a duplicator of the samples which have the important weights. The posterior PDF can be approximated by using a set of the particles $\{x_k^i; i = 1, 2, 3 \cdots N\}$ with their weights $\{w_k^i; i = 1, 2, 3, \cdots N\}$. The samples x_k^i are drawn from the importance density $q\left(x_k \mid x_{1:k-1}^i, y_{1:k}\right)$ and the weight can be computed by

$$w_k^i = w_{k-1}^i \frac{p\left(y_k \mid x_k^i\right) p\left(x_k^i \mid x_{k-1}^i\right)}{q\left(x_k^i \mid x_{k-1}^i, y_k\right)} \qquad (12)$$

The standard re-sampling process is given in [13] [14] [15] [16]. Although the re-sampling process can efficiently overcome the particle degeneracy problem, as discussed in [9], it leads to another potential issue which is called sample impoverishment. The result of the sample impoverishment implies a poor representation of the posterior density. In order to solve the particle degeneracy problem while retaining the species diversity of the samples, we designed a new particle sampling process which was modified and tested in the later stage of the project (see details in the section 3.5). A pseudo code of our particle sampling process is presented in algorithm 1.

Algorithm 1: Particle Sampling Algorithm

```
If k = 0, where k = the index of the frames;
    Initialise a particle set $\{x_k^i, w_k^i; i = 1, 2, 3 \cdots N\}$ ;
END IF STATEMENT
FOR i = 1 : N
    Draw a particle: $x_k^i \sim q\left(x_k \mid x_{1:k-1}^i, y_{1:k}\right)$;
    Calculate the weight $w_k^i$ according to the equation (12);
END FOR LOOP
Normalize the weights: $\widetilde{w}_k^i = w_k^i \,/\, \sum_{i=1}^{N}(w_k^i)$ ;
Set the threshold of the number of the effective samples: $N_{th} = 2N/3$ ;
Calculate the numbers of the effective samples in the particle set: $N_{\mathit{eff}} = \frac{1}{\sum_{i=1}^{N}(w_k^i)^2}$ ;
```

```
WHILE $N_{\text{eff}} < N_{th}$
    /* Increase the number of the effective samples according to the proportion of the threshold to the number of the particles. */
    Set a propagation: $propagation = (N_{th} - N_{eff}) * N_{th} / (N - N_{eff})$ ;
    /* In the first loop the searcher will find out the largest important weight in the particle set. In the second loop, the searcher will look for the second largest weight in the set, and so on.*/
    Search the particle L which has the (next) largest weight in the particle set $\{x_k^i, w_k^i; i = 1, 2, 3 \cdots N\}$ ;
    FOR i = 1 : prorogation
        Search the particle S which has the smallest weight in the particle set $\{x_k^i, w_k^i; i = 1, 2, 3 \cdots N\}$ ;
        Replace S with L;
    END FOR LOOP
    $N_{eff} = N_{eff} + propagation$ ;
END WHILE LOOP
RETURN the particle set;
```

2.3 Summary

In this chapter, the related works found in the technical literatures and their relevancies to this project are discussed in detail with several vivid examples. This is crucial stage, having this foundation of knowledge is of great benefit for our further works. In addition, a variation of the sampling process for the Particle filter is investigated and will be modified and test during the following design and implementation stage.

Chapter 3 - System Design and Implementation

3.1 Requirement Specification

Implementing an efficient tracking system has two primary challenges – robustness and efficiency. First of all, it is about the accuracy of the tracking result in the different environments. Secondly, it is about how many frames the tracking system can take and process per second. These two conditions are often mutually exclusive - improving the tracking accuracy means to increase the computational complexity and cost which could subsequently result in an increase in the frame processing time, particularly when tracking multiple objects. Currently, there are four standards of the frame rate used by the television and the film industries in Europe and USA - 24fps (24 frames per second), 25fps, 30fps and 50/60fps (HDTV system) [17]. Therefore, the expected system must take and process at the least 25 frames per second to facilitate the real-time frame stream tracking. With the popularity of the multi-core and multi-processor computers, we can achieve these two goals by making use of the multi-thread technique, which takes advantage of the parallel processing to carry out the scheduled operations and the multiple tracking tasks simultaneously [18]. The design of the multi-threaded program is given in section 3.6

Another tricky problem for the colour based particle filter tracker we used is that when the size of the target is very small and the background is very 'noisy' the foreground object is very easy to blur into the background resulting in loss of tracking ability. Furthermore, because the background could be changed frequently with the movement of the camera lens, the foreground/background classification approach cannot be used here. We can only use the current frame to extract the colour information of the tracking object. Moreover, processing images and comparing the histograms is a time consuming task and uses a lot of the computational resource, particularly when the size of the image is large. To reduce the amount of the image data used, thereby solving the computational bottleneck, a convenient observation method is presented in section 3.4, which can reduce the computational resource used and thus increase the frame processing rate.

The particle filtering is a conceptual model which can be implemented in many different ways. We introduced a new particle sampling process, which can effectively facilitate the processes of the Sequential Importance Re-sampling (SIR). The new sampling process, while consolidating and simplifying the SIR processes, gives a satisfactory and accurate result (A detailed discussion was given in the section 3.5).

3.2 Initialisation

The first step of implementing a tracking system is to extract the initial information about the moving objects of interest from the first frame or the first few frames of the video, for instance, the initial positions of the targets. Next, the system must store the states of the objects in a proper format which can facilitate the usage of the related data processing in the future.

A common way to detect the objects moving in the view of the camera was detailed and discussed in references [1] [19] [20] They used the background segmentation technique to classify the moving objects (foreground) and the stationary objects (background), and then the background segment will be discarded in the further processing stage, see the figure 7.

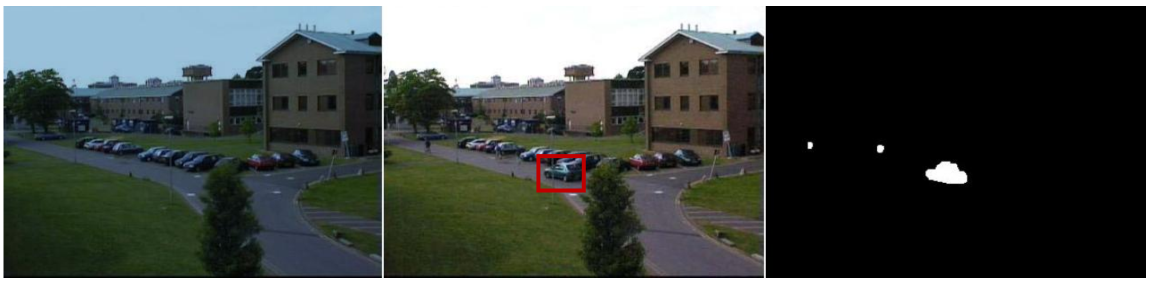


Figure 7, taken from Ref. [19]. (a) Reference Background. (b) Video Frame Inputted. (c) Segmentation Result

An excellent review of the background segmentation technique using the different colour space has been given by Fredrik Kristensen et.al [22]. The main drawback of the methods they introduced is that they require a reference background which has no moving objects, in other words it requires prior knowledge about the background. When the camera lens is moved or the lens aperture is changed, even if a slight change, the background segmentation result will be unreliable and subsequently result in a tracking failure. More recent works make use of online updating the foreground/background models to achieve the object tracking with a moving camera, for instance [21]. But, as their experiments shown, when a large change in the foreground or background occurs, the image segmenting process will fail, for instance, when a static object which was behind the tracking target begins to appear in the view of the camera, it will become a part of the tracking target. Also, it is usually the case that we only want to track the objects which we are interested in. For instance, we may want to track the players in a tennis match but not the ball-boys or spectators. Therefore, we planned to use the mouse to draw a box to specify the object's initial region and the image picked will be stored as a reference image. When a tracking object is selected, the related data of the object is computed and grouped into a data structure called "particleGroup", please see the code segment 1. The advantage of such a design is that it facilitates the following implementation of the multiple object tracking which will be used to support multi-threaded operation - each object has a self-organised "paricleGroup" and each tracking thread processes a "paricleGroup".

Code Segment 1

```
// Each tracking object will be assigned a particleGroup which consists all necessary data for the further
// processing.
```

```
typedef struct particleGroup {
    int numParticles; // The number of  the particles.
    CvMat* particleStatesMat; // 4 x numParticles matrix. The states (x, y, width, height) of all particles.
    CvMat* particleWeightsMat; // 1 x numParticles matrix. The weights of all particles.
    CvMat* stateDeviationsMat; // 4 x 1 matrix. The Gaussian deviations for all states (x, y, width, height).
    CvHistogram* histograms; // The Reference histogram.
    double bounds[2]; // The bounds of the frame.
    CvRNG randomSeed; // Random seed.
} particleGroup;
```

3.3 Transition Model

Transition model $p(x_k \mid x_{k-1})$ is used to predict the possible states of the object at the next time step. In our design, the predicted state at time k can be computed by using the following transition equation (suppose the state of the object at time k-1 is already known):

$$state[n]_k = state[n]_{k-1} + \ Gaussian\_Distribution\ (deviation) \tag{13}$$

The $state$ is a 4 x N matrix where N is denoted as the number of the particles assigned to an object.

$$state = \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \\ width_1 & \dots & width_n \\ height_1 & \dots & height_n \end{bmatrix}$$

The $deviation$ is a 1 x 4 matrix.

$$deviation = \begin{bmatrix} x\_deviation \\ y\_deviation \\ with\_deviation \\ height\_deviation \end{bmatrix}$$

The function $Gaussian\_Distribution\ (deviation)$ returns a 4 x 1 matrix which is filled with the normally (Gaussian) distributed random numbers with zero mean.

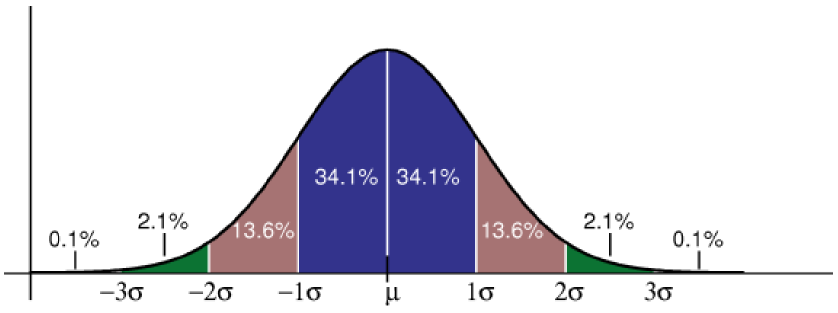


Figure 8, taken from Ref. [23]. Gaussian Distribution with mean μ and deviation σ.

As the figure 8 shows, about 95% of the values are generated within 1 and 2 standard deviations of the mean. About 5% of the values lie outside 2 standard deviations. Associate the Gaussian function to our project, when the state x_{k-1} at the time step k is already known, the state x_k can be predicted by normally distributing the particles on the basis of the previous state x_{k-1}. Then, the posterior distribution at time k can be approximated by these samples which are properly weighted in the following observation stage.

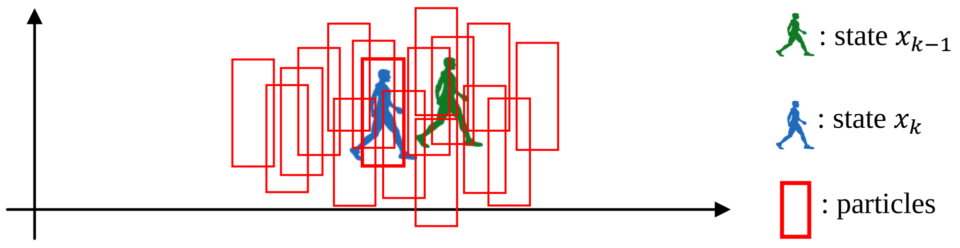


Figure 9. Particle Distribution

The figure 9 shows an example of tracking a person moving along with the x-axis on a 2D plane thus we discarded the variation of the size of the object. Each particle represents a possible location of an object at the next time step. In practice, identifying the motions and movements of the non-rigid object is more complex than that, for instance, the object may change the direction of movement or speed up suddenly. They could also run away from the lens position to the far corner resulting in a large scale change. If these kinds of situations are not taken into consideration the tracking algorithm may suffer. Our design principle is shown in the following figure 10.

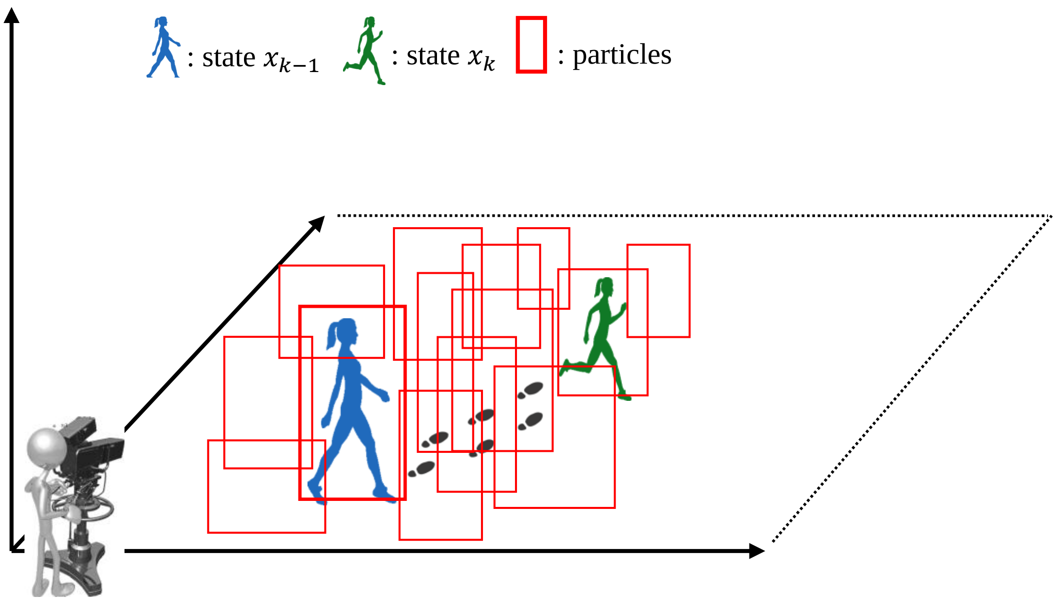


Figure 10. Tracking an object in the simulation of a real scene.

In the real scenes, from the point of view of the three-dimensional plane, the size of the moving object is changed frequently. In our design, we add the statistic noises to each prediction in order to deal with a variety of possibilities. As the figure 10 shows, the location (x, y) and the size (width, height) of each particle are normally distributed random variables which are transformed from the previous state. In addition, a threshold is added to the particles to make sure that they are distributed in an effective range not outside the bounds of the frame.

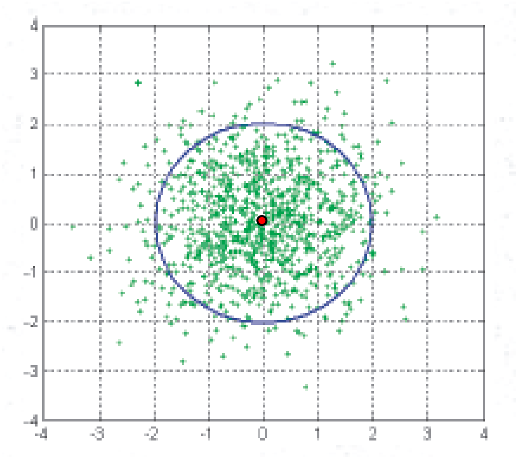


Figure 11, taken from Ref. [24]. Gaussian Distribution with 0 mean and 4 deviation.

The figure 11 illustrates an example of the Gaussian distribution. Let us review the $deviation$ matrix again. In the matrix, the $x\_deviation$ and $y\_deviation$ reflected the range of the changes of the object's location in the next time step. If the object moves quickly, the distance between current and next location will be increased. We can change the default values of the $x\_deviation$ and $y\_deviation$ from the user interface of the system to make sure that the distributions of the particles cover the scope of the object's activities - spread the particles further and wider. For the same reason, we can also change the values of the $width\_deviation$ and $height\_deviation$ to improve the accuracy of the tracking result.

Each particle represents a possibility of the object's state at the next time step. The accuracy of the tracking system can be improved while increasing the number of the particles. However, it is at the cost of the computational resource and time. In order to ensure that the tracking runs on a higher frame-rate, we improve the particle sampling process which was discussed in the section 3.5.

3.4 Colour-Based Observation Model

The particles are most like the 'search dogs', when the location of the object is confirmed at the time k-1, these 'search dogs' will be distributed around the object. At the time k, every 'search dog' will send back a response about its searching result. And then these searching results will be measured and weighted by the observation model. The location of a 'search dog' which has the largest weight is the most likely location of the object at time k.

In our design, each particle will return an image of its region and the observation process will calculate the histogram of the given image and compare it with the reference image. Finally, every particle will be marked up with a weight according to this measurement result. The colour histogram is robust to the non-rigidity and the partial occlusion but it is very sensitive to changes of illumination and suffers from the background noise. For the purpose of getting a reliable comparison result, our first step is to reduce the background noise. As the figure 12 shows, the colour of objects quoted is very similar to the background colour. During the tracking process, these objects are very easy to blur into the noisy background subsequently resulting in a loss of tracking.

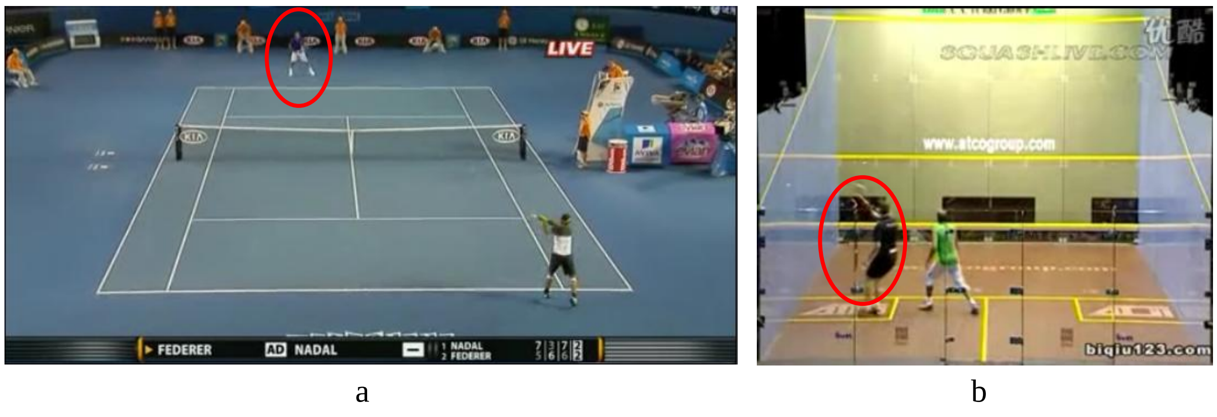


Figure 12. Video frames

Taking inspiration from the Adobe Photoshop Glowing Edges Filter [25] [26], we discovered a simple but efficient way to reduce the effect of the noise of the background, which is illustrated by using the following figures. The advantage of the background suppression method we used is that the tracking accuracy of the algorithm is improved while no significant increase in the amount of the computational cost.

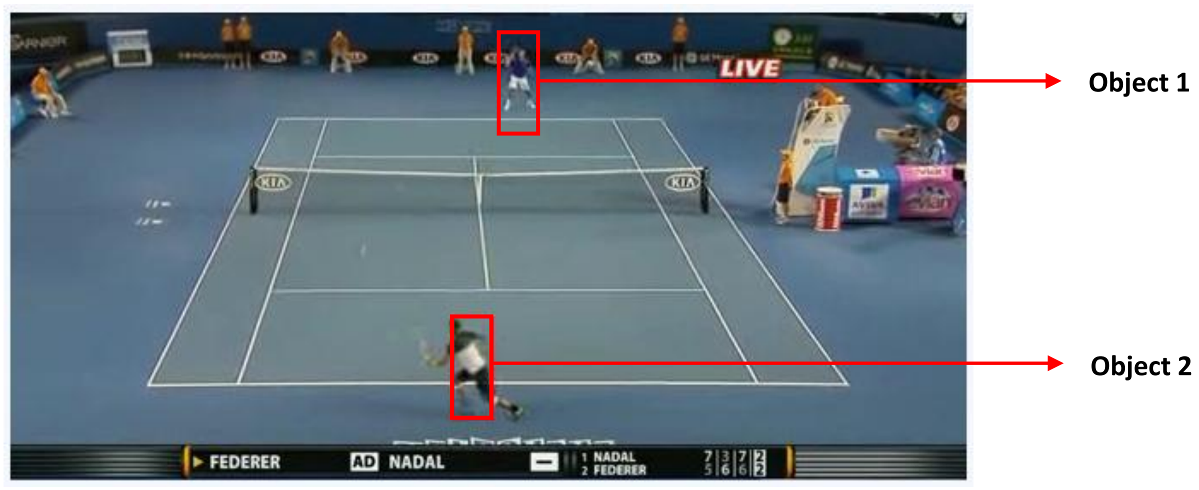


Figure 13. Image-1

Image-1 shown in the figure 12 is the original frame. Step 1: Invert (logical NOT operation) the Image-1 to create a negative image – Image-2 which is shown in the figure 14. Step 2: The expected Image-3 shown in the figure 15 is calculated by subtracting the Image-1 from the Image-2. In order to reduce the workload, the tracking system only converts the regions of where particles are instead of converting a whole image.

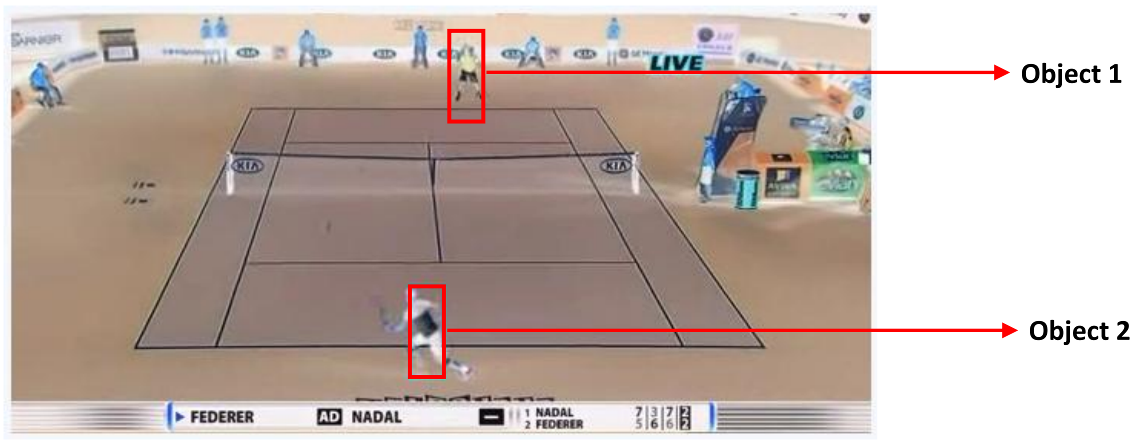


Figure 14. Image-2

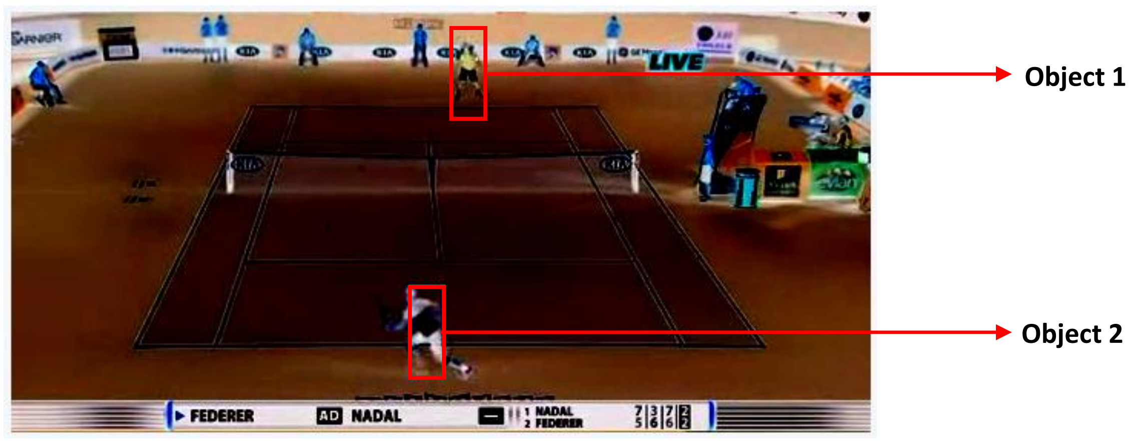


Figure 15. Image-3

After the preparation work is done, as the figure 15 shows, the background noise is greatly suppressed. The next step is the histogram comparison. As stated in [27] [28], the RGB colour space is very sensitive to the noises of the illumination and shadows, thus the tracking system requires an additional algorithm to reduce the affects of these noises, for instance [22]. A detailed discussion about the RGB colour space has been presented in [29]. By summarizing the above reasons and comparing the experiment results given by Emilio Maggio et.al [30], we plan to use HSV colour space instead of RGB.

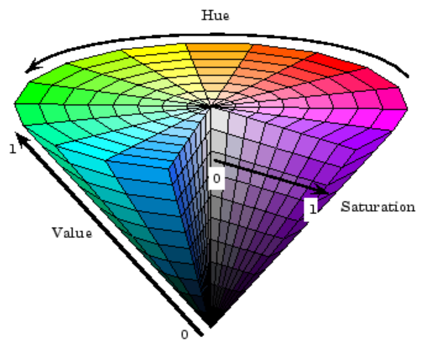


Figure 16, taken from Ref. [32]. HSV colour space.

Instead of using a combination of the primary colours to produce a secondary colour, HSV represents a colour in a circular cone model by using three components - Hue, Saturation and Value. Many colour based tracking systems rely on the illumination invariance of the frames. HSV provides an easy way to control the illumination of an image instead of using an image filter, for instance [31].The figure 17 shows the changes of the colours in the inconsistent illumination environment.

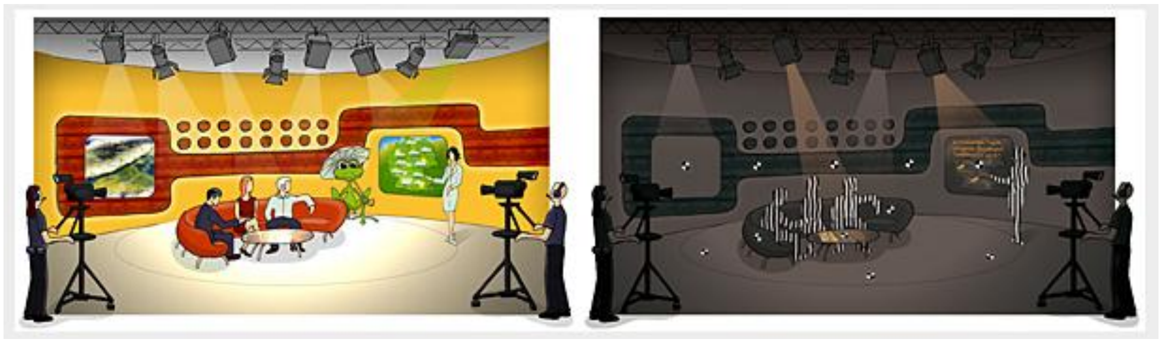
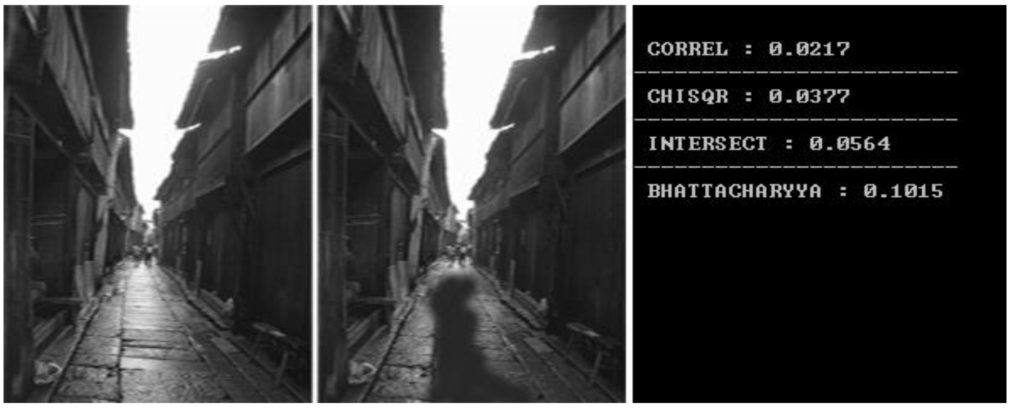


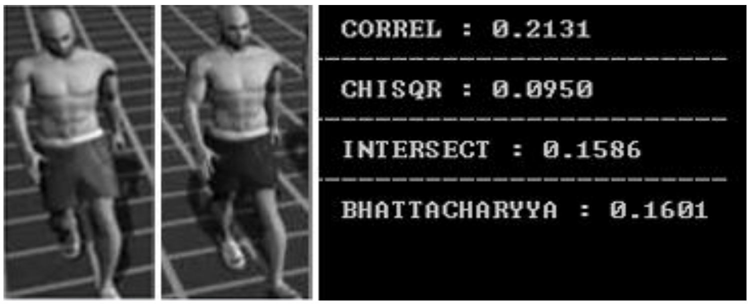
Figure 17, taken from Ref. [33]. A case of the inconsistent illumination situation.

The histograms are calculated in the HSV colour space with the 45 x 20 + 0 bins and the ranges of H, S and V channel are respectively set to 180, 255 and 0. The threshold to the histogram is set to 0.05 in order to help the histogram comparison to be more robust against the noise. The OpenCV 2.0 [34], provides four types of approaches for comparing the histograms - Correlation, Bhattacharyya, Chi-Square, and Intersection. The experiments are shown below:



```
CORREL : 0.0217
--------------------------
CHISQR : 0.0377
--------------------------
INTERSECT : 0.0564
--------------------------
BHATTACHARYYA : 0.1015
```

Figure 18. Histogram comparison - 1, the path in the second image is coloured to the black.



```
CORREL : 0.2131
--------------------------
CHISQR : 0.0950
--------------------------
INTERSECT : 0.1586
--------------------------
BHATTACHARYYA : 0.1601
```

Figure 19. Histogram comparison – 2. A walking man at the different time steps

The test results (if Hist1 = Hist2, then Result = 1) are different because these approaches are based on different statistics methodologies. As we have seen, no single approach performs the best in all situations. In order to make the comparison results of the histograms more reliable and discriminative, by summarizing our experiment results and the discussions given by Frank Porter [35] and Gary Bradski et.al [36], we planned to use two comparison methods – Intersection (14) and Bhattacharyya (15) - to compute the distance on the colour distributions between the reference image and the image picked by each particle at the time k.

$$Intersection\_Distance(Hist_1, Hist_2) = 1.0 - \sum_i^N min\big(Hist_1(i), Hist_2(i)\big) \tag{14}$$

$$Bhatacharyya\_Distance(Hist_1, Hist_2) = \left(1 - \sum_i^N \frac{\left(Hist_1(i) \cdot Hist_2(i)\right)^{\frac{1}{2}}}{\left(\sum_i^N Hist_1(i) \cdot \sum_i^N Hist_2(i)\right)}\right)^{\frac{1}{2}} \tag{15}$$

Two similar histograms have a short distance on the colour distributions which corresponds to a high weight particle and vice versa. Therefore, the weight update (observation) equation $p(y_k \mid x_k)$ is then evaluated in the following way:

$$w_k^i \propto \exp\,(2.0 - Intersection\big(image_0, image_k^i\big) - Bhatacharyya(image_o, image_k^i)) \tag{16}$$

In this project, we focus on tracking the small objects in a video which have fewer pixels on them. Therefore, we have not taken the online updating model of the reference image into consideration.

3.5 Particle Sampling

The particle sampling has two core parts - Sequential Importance Sampling (SIS) and the Sequential Importance Re-sampling (SIR). The SIS process is split into a transition model and an observation model which has been discussed in the previous sections. The SIR process is very similar to growing plant from seeds. Only the viable seeds will be produced in the following layers. In other words, the SIR particle filtering is a series of 'survival of the fittest' events throughout the tracking process. The particles which have high weights will generate more particles. On the contrary, the useless particles which have the negligible weights will be removed by the new active particles.

The enhanced particle sampling process is shown in the algorithm-2. The advantage of the SIR particle filtering algorithm we improved is that it efficiently avoids wasting the computational resource by eliminating the particles which have no contribution to the posterior PDF approximation. Furthermore, the particle degeneracy problem is also solved while it retains the species diversity of the samples.

Algorithm 2: Enhanced Particle Sampling Process

```
      If k = 0, where k = the index of the frames;
          Initialise a particle set $\{x_k^i,\ w_k^i; i = 1, 2, 3 \cdots N\}$ , where N = the number of particles;
      END IF STATEMENT
      FOR i = 1 : N
SIS       Draw a particle: $x_k^i \sim q\left(x_k \mid x_{1:k-1}^i, y_{1:k}\right)$;
          Calculate the weight $w_k^i$ according to the equation (12);
      END FOR LOOP
      Normalize the weights: $\widetilde{w}_k^i = w_k^i \ / \sum_{i=1}^N (w_k^i)$ ;
      propagation =  N/3;
      newParticles = 0;
      index = 0;
      Create a new particle set $\{new\_x^i, new\_w^i; i = 1, 2, 3 \cdots N\}$, where $new\_w^i$ = 1/N;
      FOR i = 1 : N
          Search the particle L {L.x, L.w} which has the largest weight in the particle set $\{x_k^i,\ w_k^i; i = 1, 2, 3 \cdots N\}$ ;
          newParticles = L.w * N + propagation;
          propagation = 2 * propagation/3;
          L.w = 0;
          WHILE newParticles > 0
              copy L.x to $new\_x^{index}$ ;
SIR           newParticle --;
              index ++ ;
              IF index = N THEN go to exitFlag;
          END WHILE LOOP
      END FOR LOOP

      WHILE index <  N
          copy $new\_x^0$ to $new\_x^{index}$ ;
          index ++ ;
      END WHILE LOOP
      exitFlag:
      RETURN the new particle set $\{new\_x^i, new\_w^i; i = 1, 2, 3 \cdots N\}$,
```

3.6 Multithread Support

The primary motivation for using multiple threads is to achieve an optimum performance on the modern multi-core or multiprocessor systems. Usually, a multiple object tracking system has a heavy workload which could result in it being unable to implement the tasks in real time. As shown in the figure 20, the great benefits of multithreading the tracking system on the multi-core processors are that not only can it save the computation time, but also, it can reduce the cost by properly paralleling the tracking tasks and sharing the memory resources.

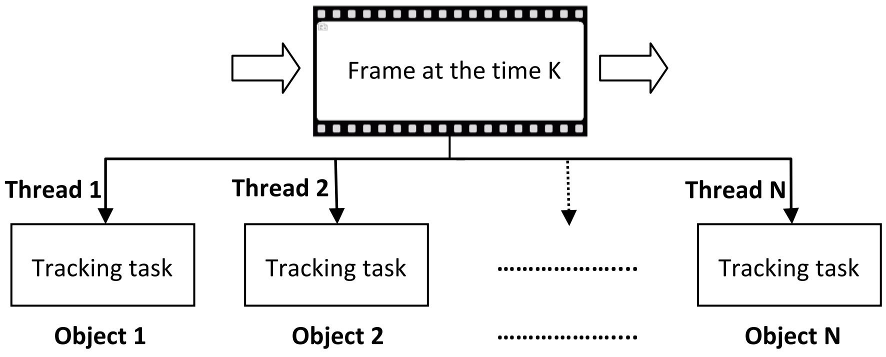
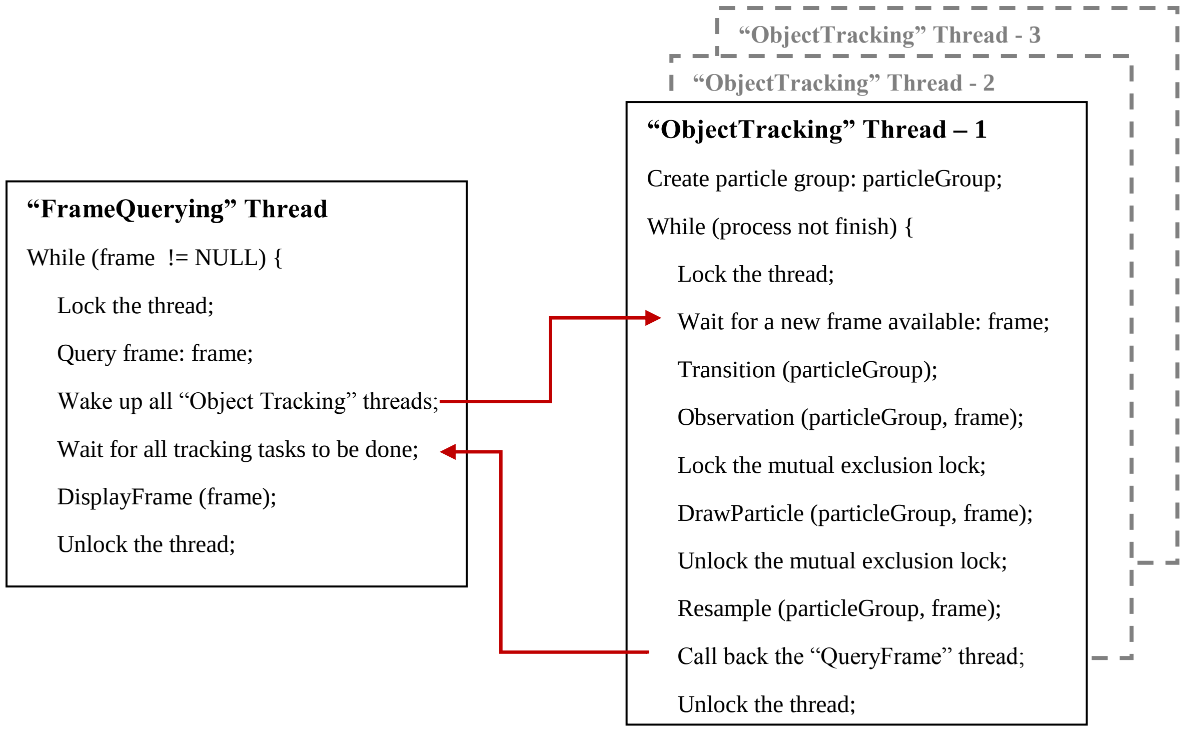


Figure 20. Paralleling tracking tasks.

In this project, we planned to use the Posix thread library [18] [37], which provides a high quality API for implementing multithreaded applications in the C programming language. The figure 21 illustrates the interactions between the "FrameQuerying" thread and the "ObjectTracking" thread.



```
"FrameQuerying" Thread
While (frame  != NULL) {
    Lock the thread;
    Query frame: frame;
    Wake up all "Object Tracking" threads;
    Wait for all tracking tasks to be done;
    DisplayFrame (frame);
    Unlock the thread;
```

```
"ObjectTracking" Thread – 1
Create particle group: particleGroup;
While (process not finish) {
    Lock the thread;
    Wait for a new frame available: frame;
    Transition (particleGroup);
    Observation (particleGroup, frame);
    Lock the mutual exclusion lock;
    DrawParticle (particleGroup, frame);
    Unlock the mutual exclusion lock;
    Resample (particleGroup, frame);
    Call back the "QueryFrame" thread;
    Unlock the thread;
```

Figure 21. Interaction between "FrameQuery" thread and "ObjectTracking" thread.

3.7 System Architecture

The figure 22 illustrates the procedure of our tracking system. The number of 'Object tracking' threads created is according to how many targets the user has selected.

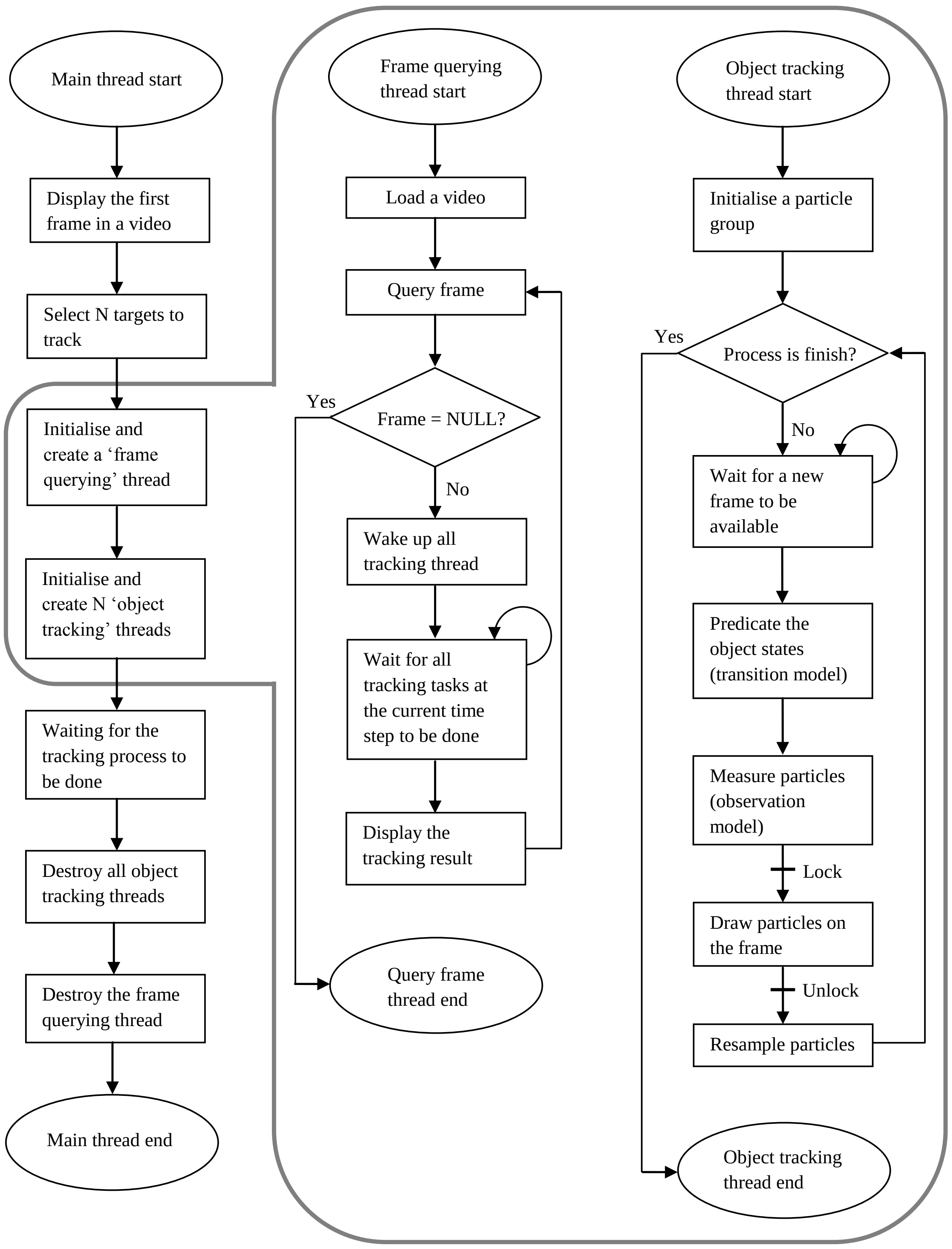

Main thread start
Display the first frame in a video
Select N targets to track
Initialise and create a 'frame querying' thread
Initialise and create N 'object tracking' threads
Waiting for the tracking process to be done
Destroy all object tracking threads
Destroy the frame querying thread
Main thread end
Frame querying thread start
Load a video
Query frame
Frame = NULL?
Yes
No
Wake up all tracking thread
Wait for all tracking tasks at the current time step to be done
Display the tracking result
Query frame thread end
Object tracking thread start
Initialise a particle group
Process is finish?
Yes
No
Wait for a new frame to be available
Predicate the object states (transition model)
Measure particles (observation model)
Lock
Draw particles on the frame
Unlock
Resample particles
Object tracking thread end

Figure 22. System architecture

The sub-threads 'FrameQuerying' and 'ObjectTracking' are created in the main thread and are destroyed when the video has end. A mutual exclusion lock is created in order to make sure that the frame at the current time step is consistent when the 'ObjectTracking' threads draw the tracking results on it. When a new frame is inputted, the 'FrameQuerying' thread will wake all 'ObjectTracking' threads so that they can perform their tracking tasks. The 'ObjectTracking' threads will go to sleep when the tasks are finished, and wait to be called again when a new frame is available. The 'FrameQuerying' thread will read the next frame in a video when it confirms that all tracking tasks are finished.

3.8 Summary

The tracking targets are extracted manually by drawing bounding boxes on the first frame. Once this is complete, a particle group is constructed for each target. A colour histogram is computed for each bounding box which serves as a reference template. The Intersection and Bhattacharyya distances between the reference template and the samples are then used to weight the particles.

For the purposes of reducing the affect of background noise, a simple but efficient background suppression method has been introduced and its performance will be illustrated by comparison in the next chapter. In addition, a variation of the sampling process for the Particle filter is designed and modified in the system implementation stage so as to make it more efficient and realistic. The related experiment result will be given and discussed in the next chapter.

Chapter 4 - Experiments Result

4.1 User Interface and Guidance

For the purpose of facilitating the data configuration process, instead of using the command line, we built a simple user interface which is shown below.

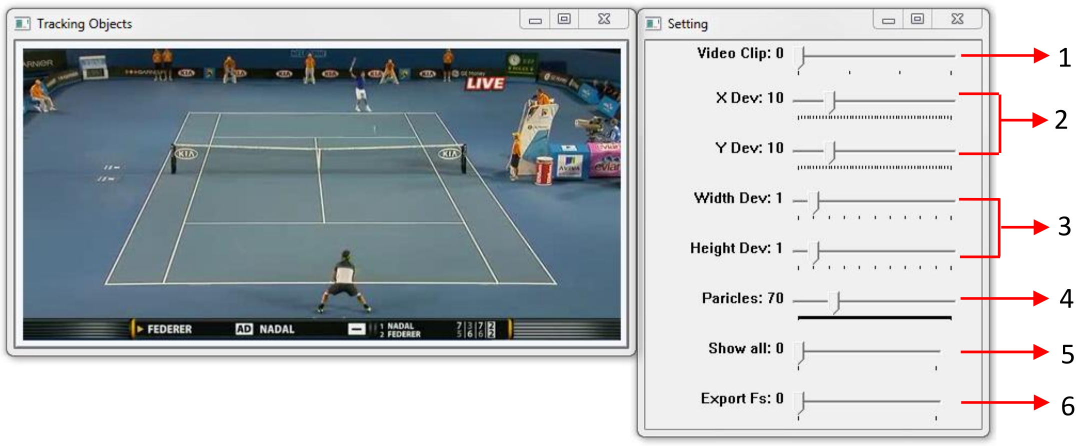


Figure 23. User Interface.

1. Choose a video clip - two tennis matches and three squash matches.
2. X and Y deviation, which refer to the variation of the objects' moving speed.
3. Width and Height deviation, which refer to the variation of the objects' size.
4. The number of the particles assigned to each object.
5. Displays all particles, or not.
6. Exports the frames which are marked by the particles, or not.

The tracking targets are selected manually by drawing bounding boxes, and then pressing the ENTER button to start tracking. (Note: the targets must be selected precisely, otherwise a tracking failure may occur. In order to produce an accurate result in various contexts, some HSV parameters could optionally be adjusted manually).

4.2 Organisation of Experiments

The tracking system was tested in several contexts. The experiments are organized as follows:

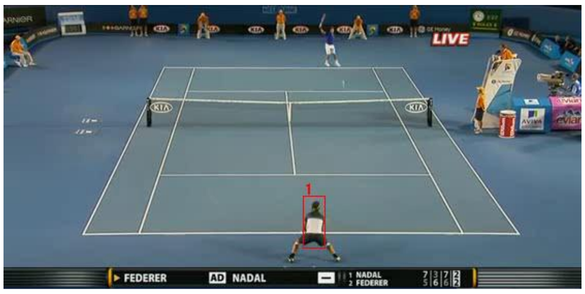
- Section 4.3.1 Single Object Tracking: Tracking a very flexible object.
- Section 4.3.2 Multiple Objects Tracking - 1: Tracking a small object in a clutter background and comparing with the tracking results when not using the background suppression method.
- Section 4.3.3 Multiple Objects Tracking - 2: Tracking multiple objects in a low resolution video.
- Section 4.3.4 Tracking partially occluded objects - 1: Tracking multiple objects that are partially occluded.
- Section 4.3.5 Tracking partially occluded objects - 2: Tracking multiple objects under the completed occlusions.

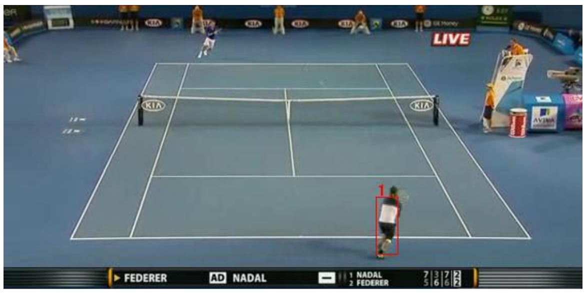
4.3 Experiments

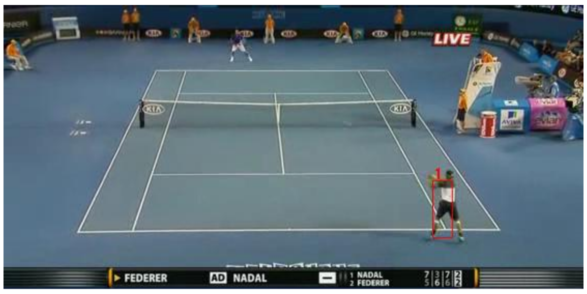
The tracking experiments are carried out on four video clips - two tennis and two squash matches.

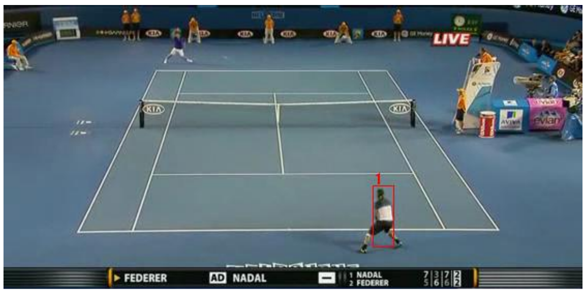
4.3.1 Single Object Tracking

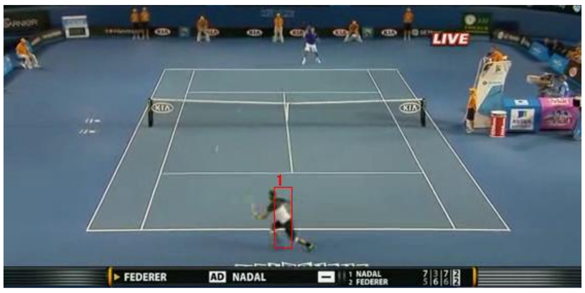
In this section, we present the experiment results of tracking a very flexible object in a relatively clear background. The video resolution is 512 x 228 pixels.

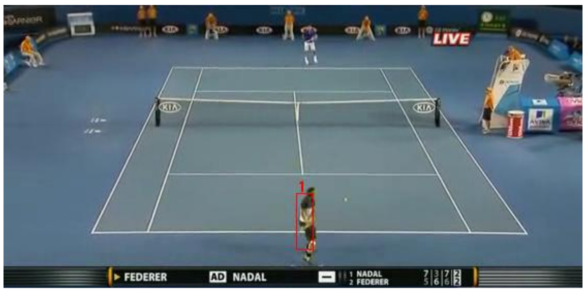

Frame 1.

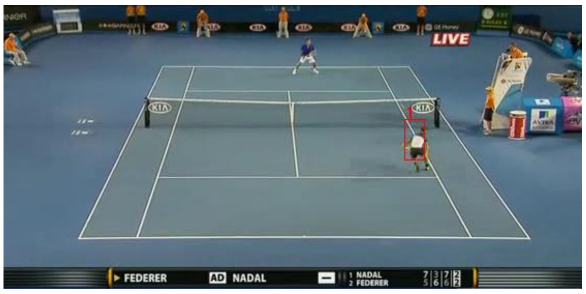

Frame 34.


Frame 49. Lens changed.


Frame 68.


Frame 112.


Frame 157.


Frame 212.

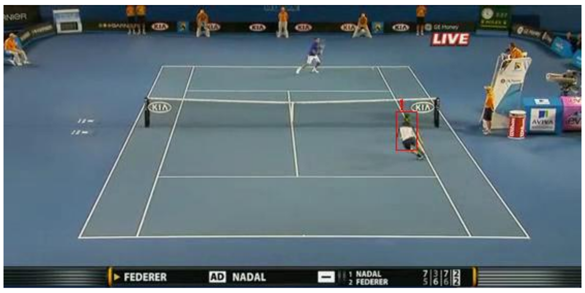
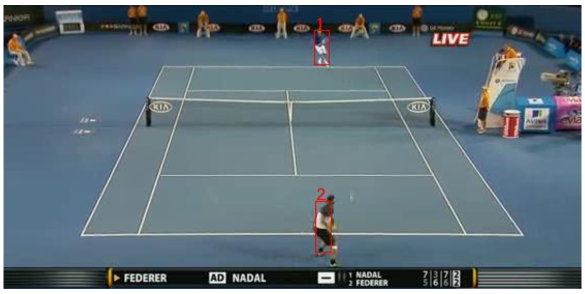

Frame 217.

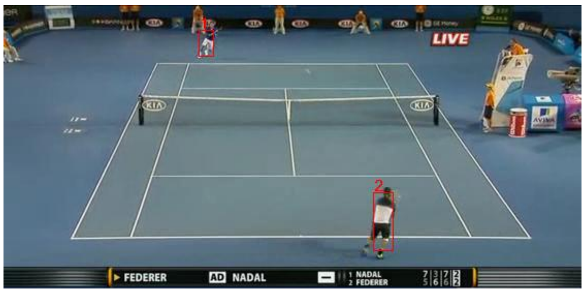
Figure 24. Single object tracking.

4.3.2 Multiple Objects Tracking - 1

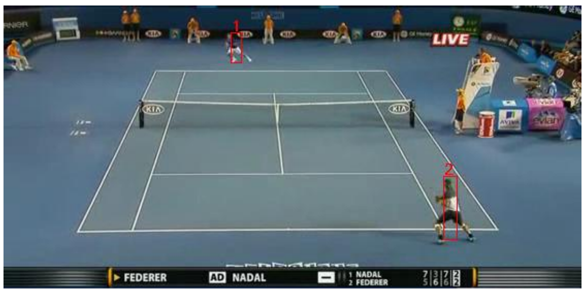
In this section, we present the experiment results of tracking a small object (object 1) in a clutter background and comparing with the tracking results while not using the background suppression method.



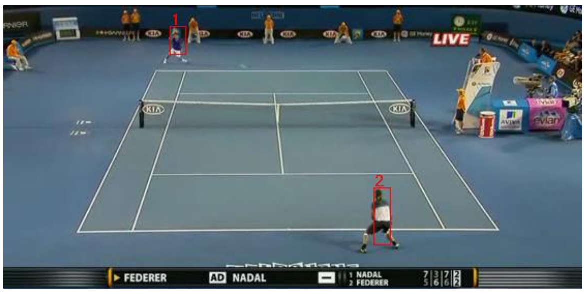
Frame 8.



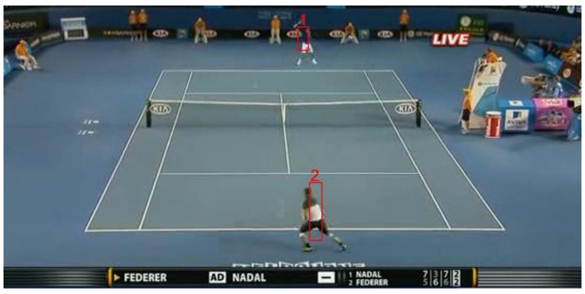
Frame 33.



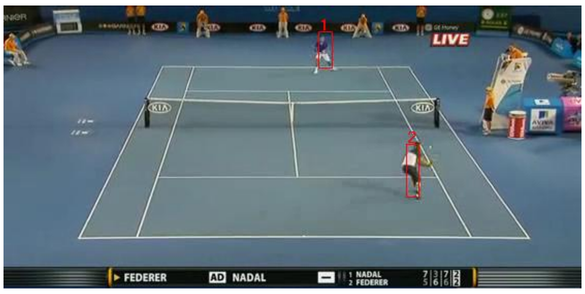
Frame 53.



Frame 69.



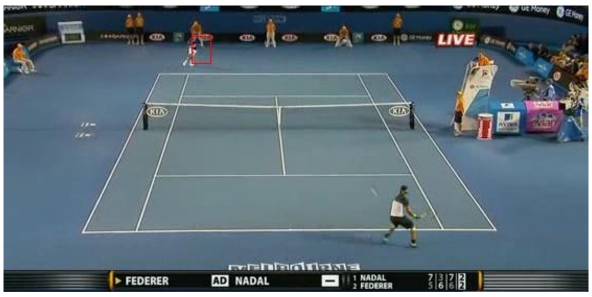
Frame 108.



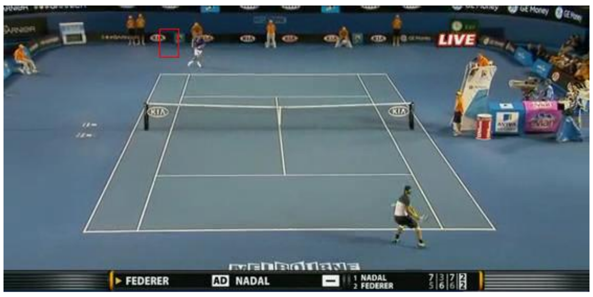
Frame 205.

Figure 25. Multiple objects Tracking.

While not using the background suppression method, the system lost the object in frame 78.



Frame 77.

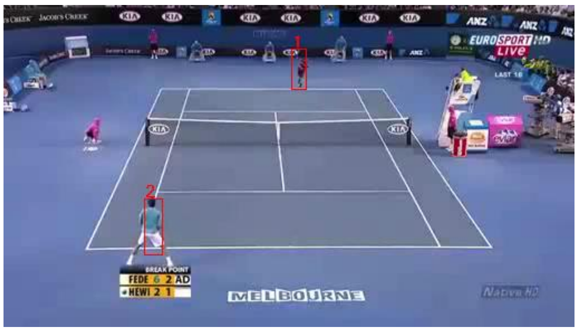


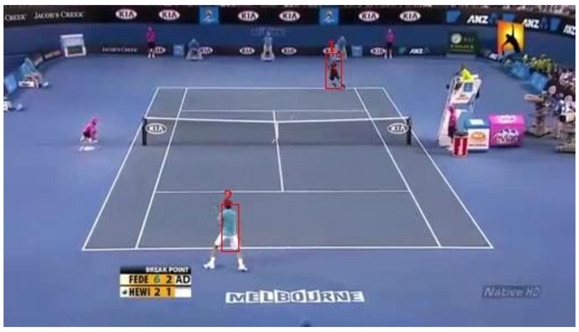
Frame 78.

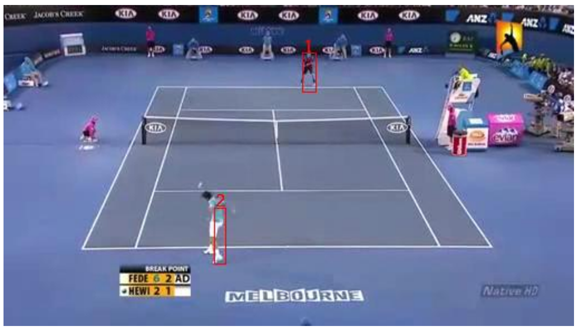
Figure 26. Multiple objects tracking without using the background suppression method.

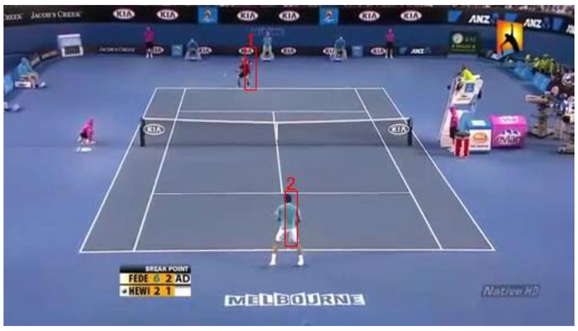
4.3.3 Multiple Objects Tracking - 2

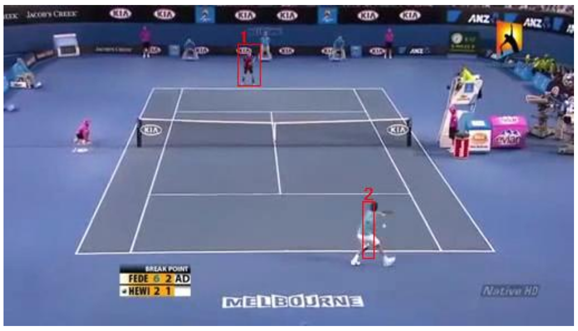
In this subsection, we present the experiment result of tracking multiple objects in a clutter background in a different context. The video resolution is 512 x 228 pixels.

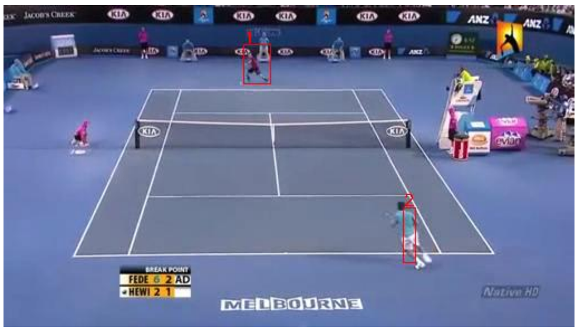

Frame 15.

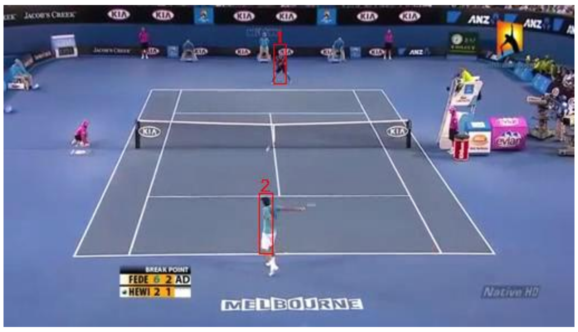

Frame 53.


Frame 63.


Frame 84.


Frame 98.


Frame 115.


Frame 142.

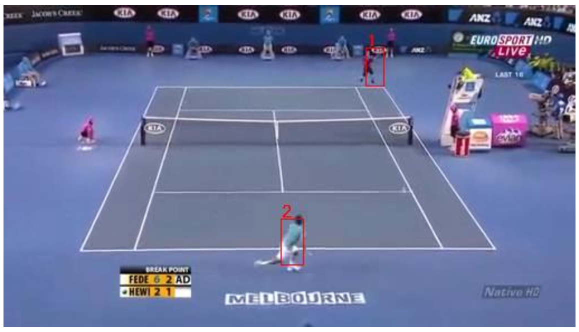
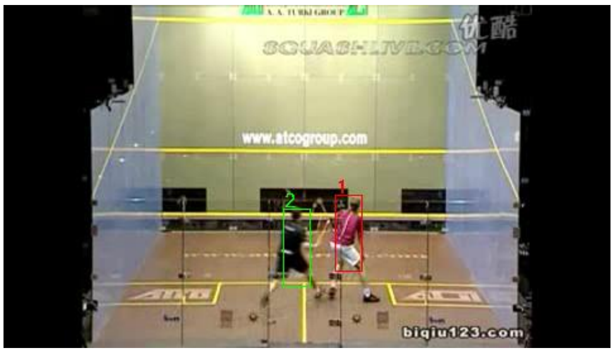
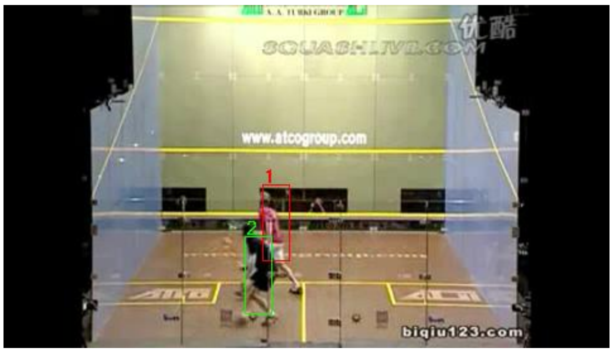

Frame 220.

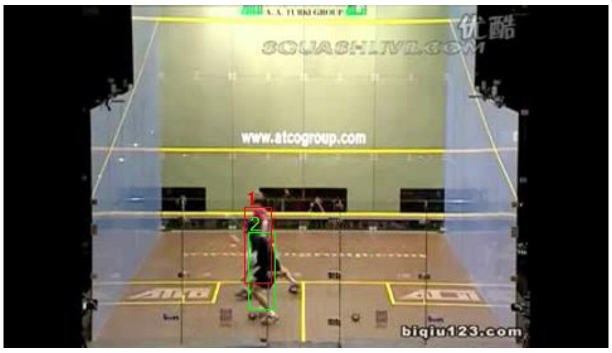
Figure 27. Multiple objects tracking.

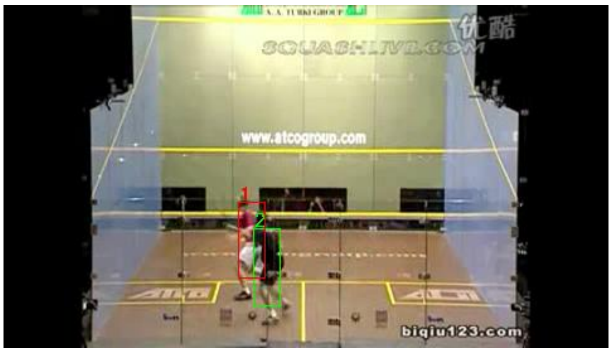
4.3.4 Tracking partially occluded objects - 1

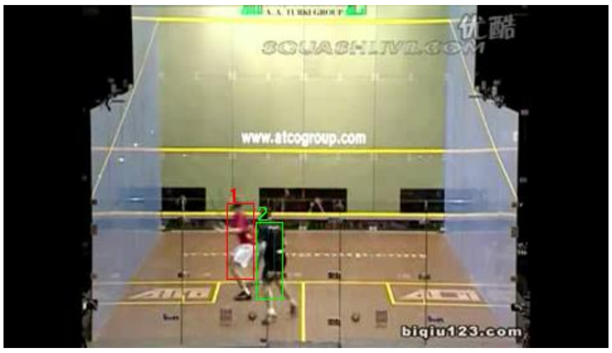
In the fourth experiment, we present the experiment results of tracking multiple objects under the partial occlusions. (Note: each object is assigned 300 particles. width and height deviation is set to zero). A partial occlusion occurs in frame 31.


Frame 1.


Frame 30. Partial occlusion starts.


Frame 31. Partial occlusion occurs.


Frame 33.


Frame 34. Partial occlusion ends.

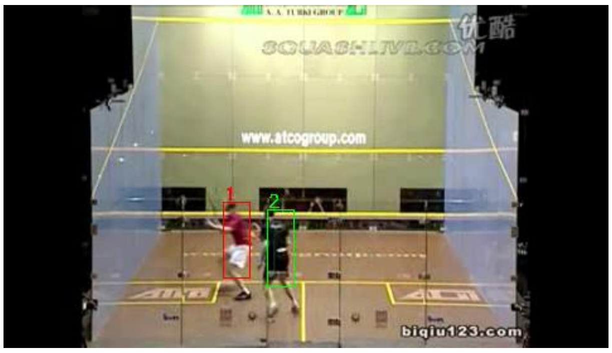
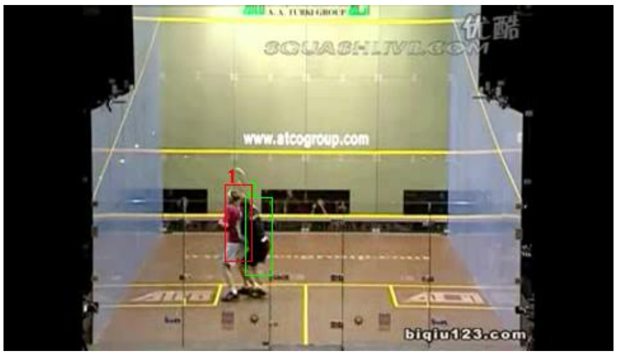
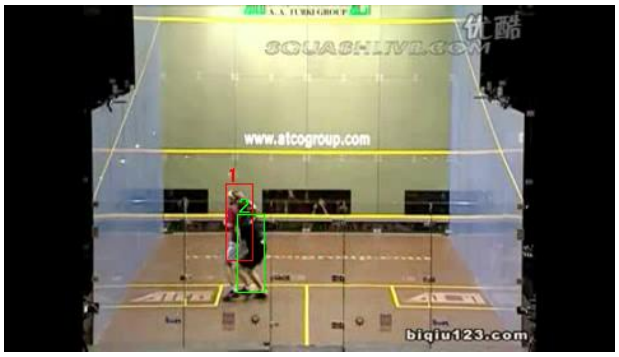
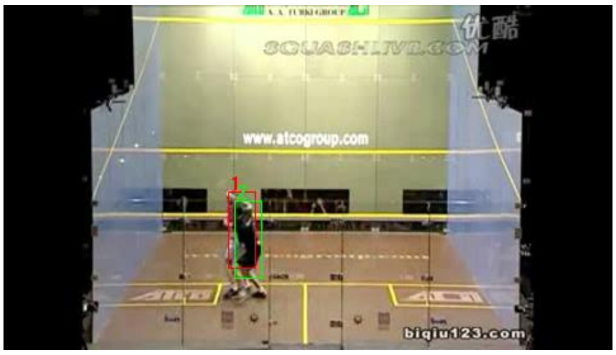

Frame 35.

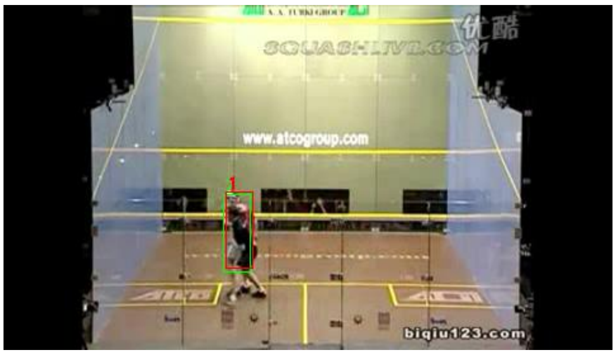
Figure 28. Tracking multiple objects under a partial occlusion.

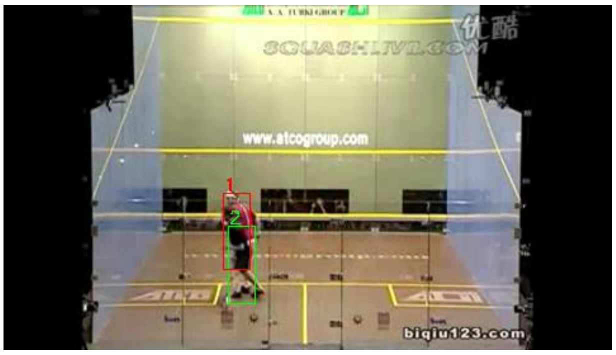
The following figure 29 illustrates the tracking result when the two players entangle with each other. The interaction starts in frame 51. An overlapping occurs in frame 54. As shown in these figures, our tracking system can efficiently and successfully handle the fast moving objects under the partial occlusions.

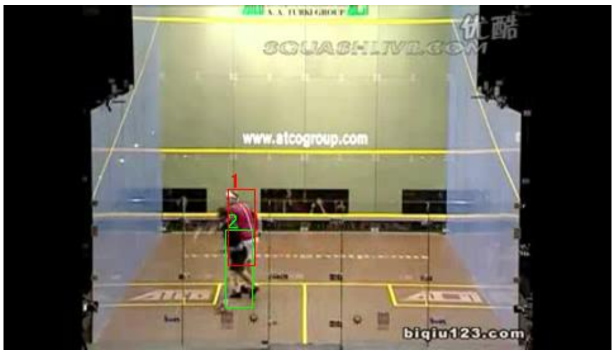

Frame 51. Interaction starts.

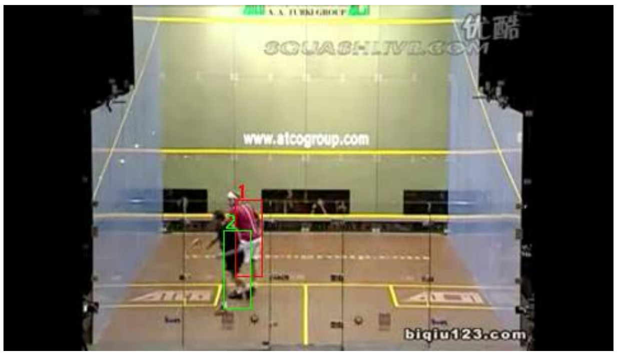

Frame 52.


Frame 53. Overlapping starts


Frame 54. Overlapping occurs


Frame 55.


Frame 56.


Frame 57. Interaction ends

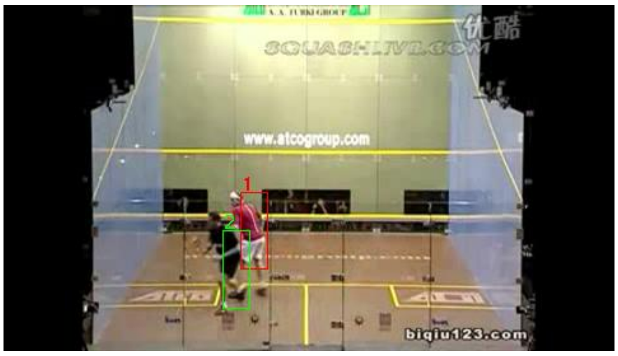
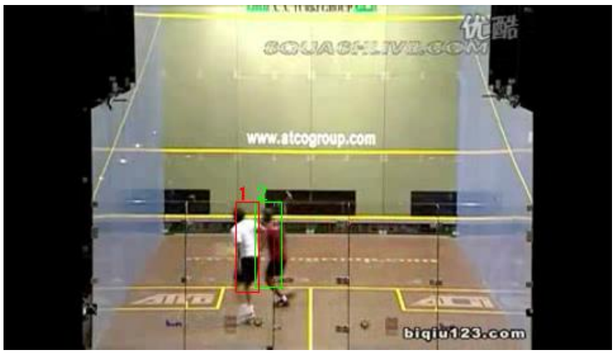

Frame 58.

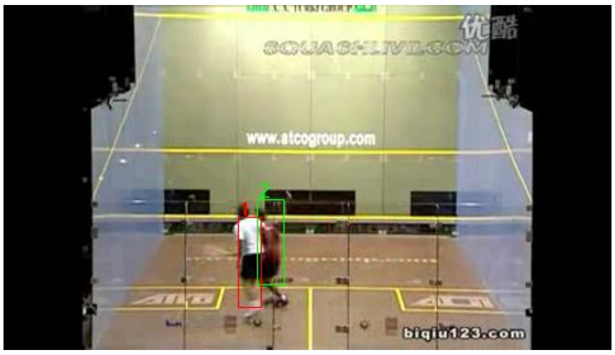
Figure 29. Tracking the multiple objects under a partial occlusion.

4.3.5 Tracking partially occluded objects - 2

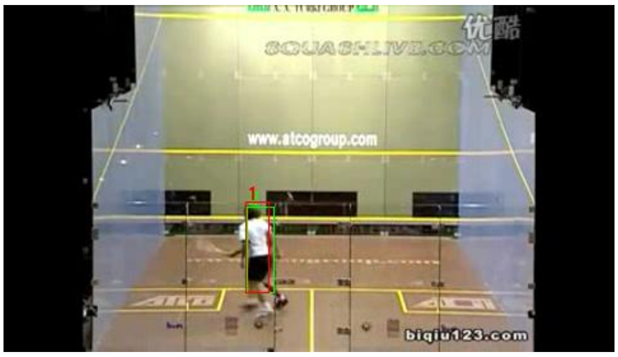
This subsection presents the experiment results of tracking multiple objects while under complete occlusions. The figure 30 illustrates an accurate tracking result when a complete occlusion occurs in frame 37.



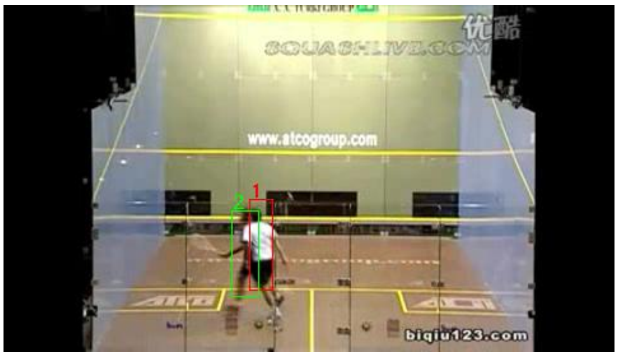
Frame 35.



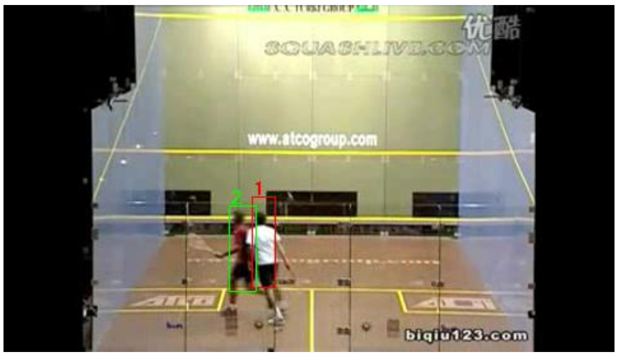
Frame 36. Occlusion starts



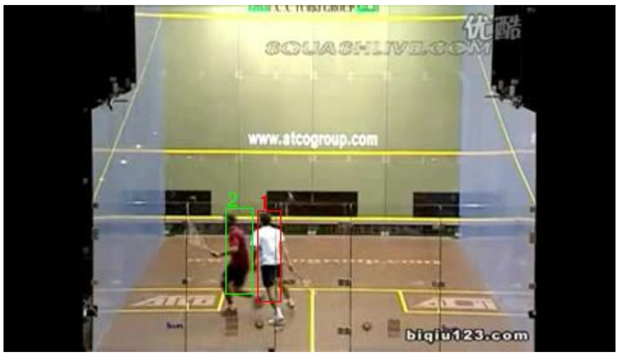
Frame 37. Complete occlusion occurs.



Frame 38.



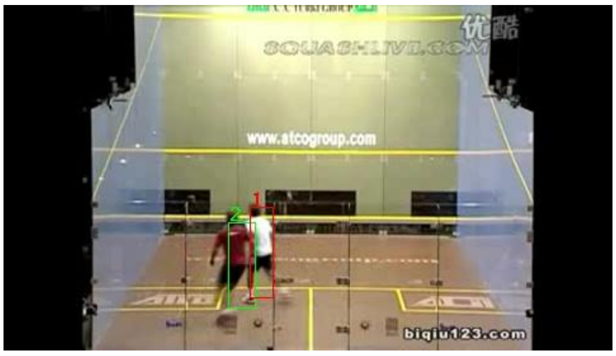
Frame 39.

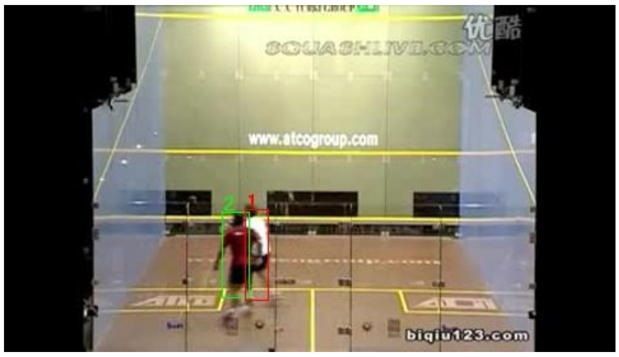


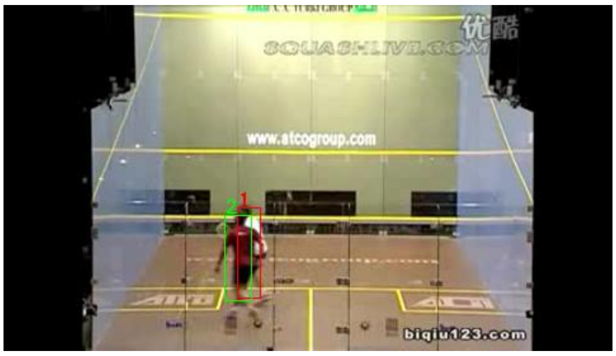
Frame 40. Occlusion ends

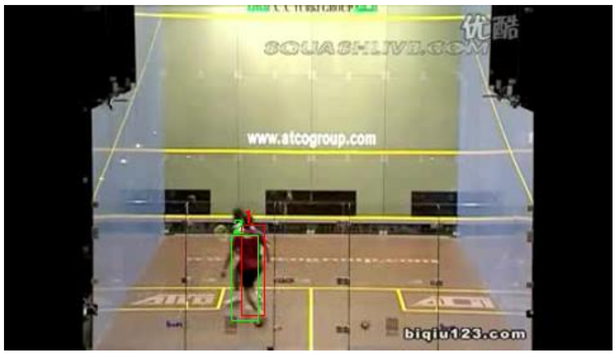
Figure 30. Tracking multiple objects under a complete occlusion.

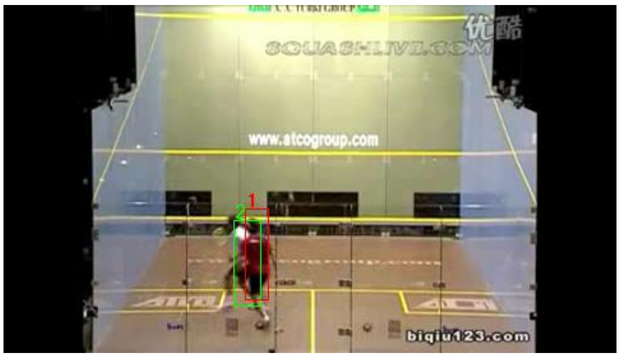
However, in the following video sequence, as shown in the figure 31, a tracking failure occurs due to a complete occlusion. In frame 61, we see the system tended to produce an incorrect tracking result. After the 62nd frame, one player has lost tacking. This can be solved by increasing the number of particles.

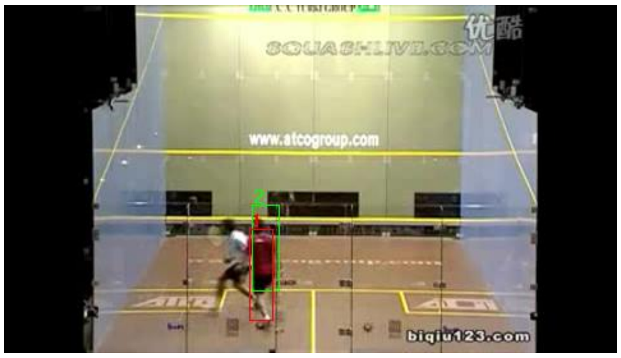

Frame 56.

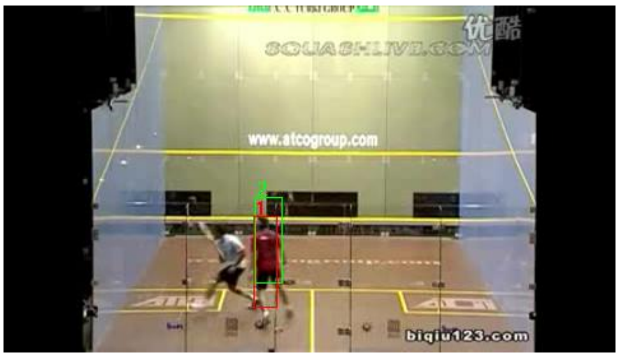

Frame 57. Occlusion starts


Frame 58.


Frame 59.


Frame 60.


Frame 61. Failure occurs.


Frame 62. One player loses tracking.

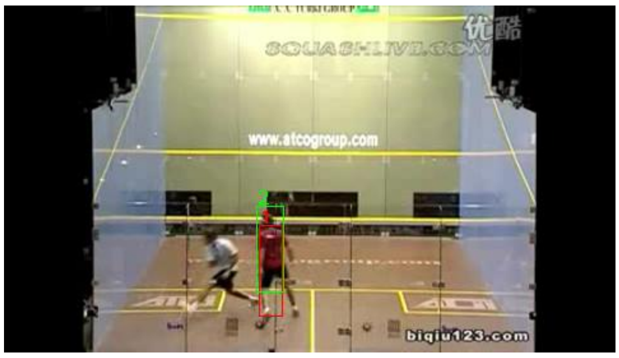
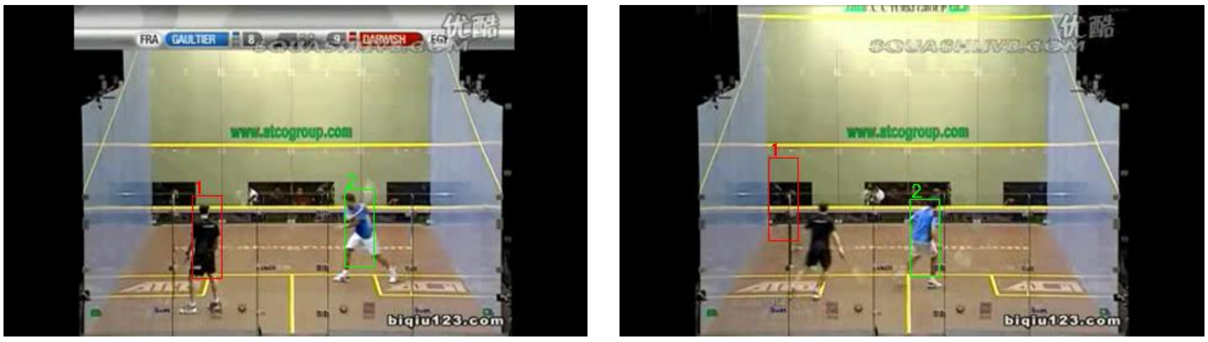

Frame 63.

Figure 31. Tracking failure due to a complete occlusion.

4.4 Experiment Review

A key issue addressed in the experiments is that a tracking failure may occur when a large part of the region of an object has the same or very similar colours as the background, as shown in the figure 32. This can be solved by properly selecting the HSV bin size and ranges or using the background subtraction techniques to cope with this problem.



a b

Figure 32. An issue in tracking object.

The main drawback of our experiments is that cannot provide a reliable report about the operational efficiency of the multithreaded tracking systems due to the hardware limitations. In the future, this work could be extended in the following directions: (I) Test the system on a multi-core or multiprocessor system. (II) Test the system in other contexts under more challenging scenarios. (III) Explore the limitations and improvements by comparing with other tracking methods.

4.5 Summary

We tested the tracking system with a variety of situations, including: tracking very flexible objects, tacking small objects in a clutter background, handling partial occlusions and complete occlusions. The experiment results show that our system is reasonably successful at working in these scenarios.

Chapter 5 - Future Works

5.1 Limitation and Possible Extension

Because our observation model is based on analysing the colour distribution of the targets, some HSV parameters could be adjusted manually in order to produce an accurate result in the different contexts. Furthermore, the main drawback of the colour based observation model is that a tracking failure may occur when two objects have the similar colour information, see the figure 33.

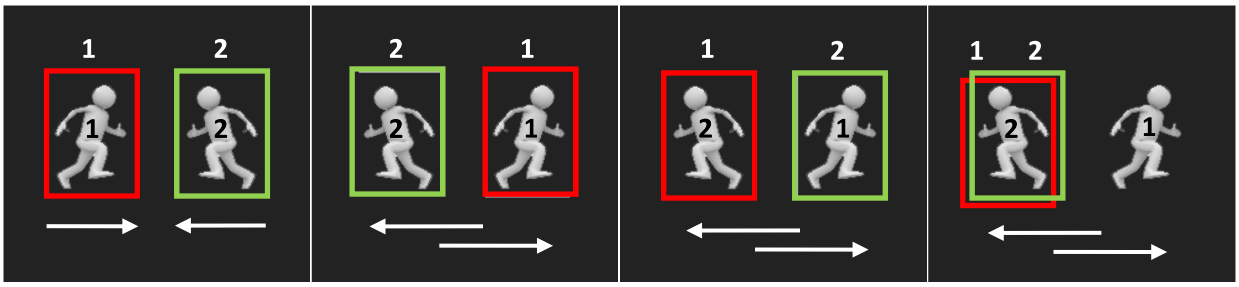


Figure 33. Object tracking scenarios.

(a) Represents two objects at the time k, all of them are assigned a tracking particle.
(b) Represents a correct tacking result after the two objects pass each other at the time k + 1, both of the tracking particles have continued to track the correct target.
(c) Represents an incorrect tacking result after the two objects have passed each other. Both of the tracking particles are now tracking the incorrect target for instance, the particle-1 should correspond to the object-1, but it is not.
(d) Represents an incorrect tacking result at the time k + 1, two tracking particles are focused on the same target, one object has lost tracing.

The problem shown above is a practical issue. We plan to use the Scale Invariant Feature Transform (SIFT) matching technique to cope with this tricky problem.

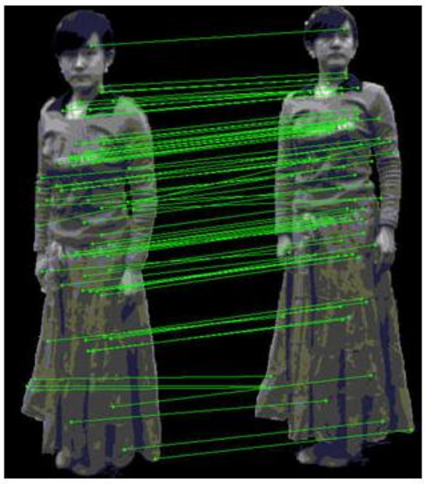
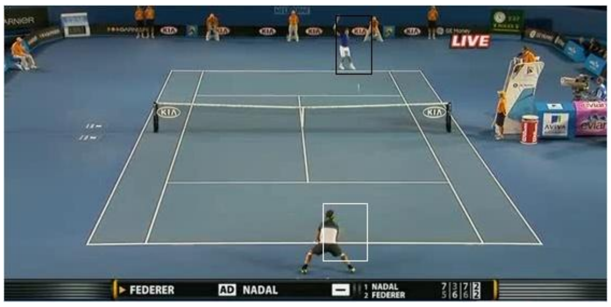


Figure 34, taken from Ref. [38]. SIFT image matching technique.

The SIFT image matching technique has been generally discussed in the section 2.1. The SIFT image matching process works by extracting the interest points and comparing their relationships in images. A detailed discussion is given in [39]. Further, an excellent example of tracking a single object is presented by Tom Mathes et.al [40]. Because the features of the unique SIFT points on an object are invariant to the changes of the object, such as a change in the size of an object, it is very suitable for use in our tracking system as a component used to distinguish between two objects which have very similar colour information. Because the workload of the SIFT tool is very heavy for a multiple object tracking system, it cannot exist in the entire life cycle of a tracking process. In our future plan, the SIFT function will only be called into action when an occlusion has occurred, and use a single thread to monitor the occlusion events by checking if two or several particles which belong to different groups overlap together.

Another crucial issue which may cause a tracking failure is the manual initialisation. In our design, the tracking target is selected by using the mouse to draw a bounding box. When a target selection is imprecise particularly in a clutter background, the subsequent tracking result may be inaccurate. See the figure below - A failure occurred in a tracking process which is caused by the unnecessary background information extracted.


a. Invalid manual initialisation.

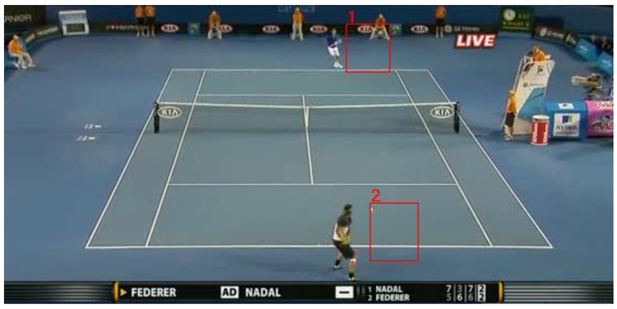
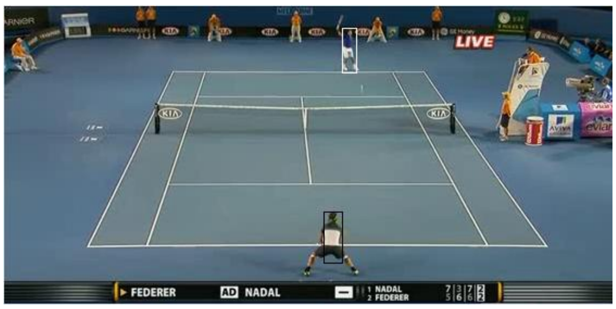

b. Frame 9.

Figure , (b) represent a tracking failure due to a invalid manual initialisation.


a. Valid manual initialisation.

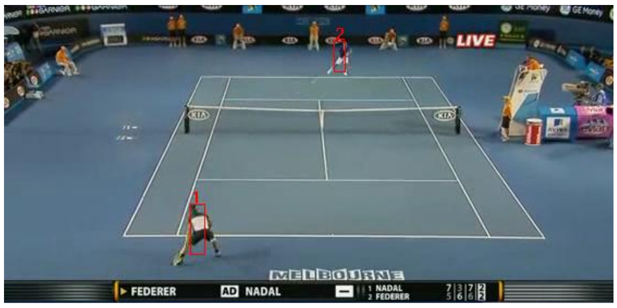

b. Frame 126.

Figure 35, (b) Represent a accurate tracking result.

One possible solution to this problem is to use an automatic detection instead of the manual initialisation by using the background segmentation techniques and adopting an online background update model to cope with the sudden changes of the background. This is an ideal solution for our design context where the camera is not fixed and the reference background image cannot be acquired offline. However, as the experiments shown in [33], the background segmentation methods still require the user interaction to get the accurate results, and the online background update model has a heavy workload which results in being unable to provide real-time responds.

5.2 Summary

In this chapter, we highlighted the undesirable issues which still remain, and related proposals are introduced in order to cope with these problems. Some of the extensions could introduce further problems. We leave these issues at the current step and will add to them later as a further development to our tracking system.

Chapter 6 - Conclusion

In this paper, we presented a colour based tracking system in a particle filtering framework which uses the Monte Carlo method to solve the recursive Bayesian estimation problem. The tracking targets are extracted manually by drawing bounding boxes. After the initialisation, a particle group is constructed for each target and the colour histogram inside the bounding box is computed and serves as a reference template. Then, the distances on the colour distributions between the template and the particles are used to weight the particles.

Our system is reasonably successful in tracking the players in the tennis and squash matches. In these contexts, the background noise can be reduced properly by using our background suppression method. The objects' movement and their positions can be accurately detected and located. The particle sample process we introduced is very flexible and can be extended in a number of useful ways. We can dominate the species diversity of the samples in order to get the optimized tracking result by changing the propagation rate of the important samples according to the situations.

By multithreading the colour based tracking system, not only can it speed up its performance on the multiprocessor or multi-core systems, but also easily integrate with other tracking methods by creating a new thread individually. As an example of this, it would be possible to use the colour based method which requires less computational cost to track basketball players and use the contour based method to track the ball's trajectory.

This project has completed all aims and objectives listed in the beginning of the paper. The entire life cycle of the software development has been properly presented. However, due to the time constraints, several drawbacks are still remaining and many issues should be considered in the future.

A robust tracking system should be competent in a variety of situations with a few configuration changes, for instance, a versatile system which is used to track the players in a football match is also able to be a vehicle tracking system. Furthermore, for a real time tennis player tracking system an indispensable functionality is that it can detect and record the key events occurring during a match, for instance, ball strike. We have a long way to go to create a fully-fledged tracking system, but we are very optimistic that our tracking system can be improved in the near future to adapt to the varied needs of the users.

Reference

1. Marti Balcells, Daniel DeMenthon and David. Doermann. ***An Apparent Based Approach for Consistent Labelling of Humans and Objects In Video***. University of Maryland, USA. (Cited on pages 9, 18).
2. Le Lu and Gregory Hager. ***Dynamic Foreground/Background Extraction from Images and Videos using Random Patches***. Johns Hopkins University, USA. (Cited on pages 9).
3. Nan Lu, Jihong Wang, Q.H. Wu and Li Yang. ***An Improved Motion Detection Method for Real-Time Surveillance***. IAENG International Journal of Computer Science. (Cited on page 9).
4. Ramani Pichumani. ***Snakes: an active model***. (Cited on page 10). http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RAMANI1/node30.html
5. Bristol Computer Vision Group. ***Active Contours***. University of Bristol, UK. (Cited on page 10). http://www.cs.bris.ac.uk/Research/Vision/activecontours.jsp
6. Masayuki Yokoyama and Tomaso Poggio. ***A Contour-Based Moving Object Detection and Tracking***. Sony Corporation, Japan. (Cited on page 10).
7. Xianghua Xie. ***Contour Based Object Tracking***. University of Swansea, UK. (Cited on page 10). http://www.cs.swan.ac.uk/~csjason/prins/prins.htm
8. Takeo Kanade and Jun-Su Jang. ***Feature-based 3D Head Tracking***. (Cited on page 11). http://www.ri.cmu.edu/research_project_detail.html?project_id=619&menu_id=261
9. Simon Maskell, Neil Gordon et.al. ***A tutorial on Parcile Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking***. IEEE. (Cited on pages 11, 12, 14, 15).
10. Greg Welch and Gary Bishop. ***An Introduction to the Kalman Filter***. University of North Carolina, USA. (Cited on page 13).
11. Gabriel A. Terejanu. ***Unscented Kalman Filter Tutorial***. University at Buffalo, USA. (Cited on page 13).
12. Eric Lehmann. ***Particle Filtering***. The Australian National University, Australian. (Cited on page 14). http://users.cecs.anu.edu.au/~hartley/Vision-Reading-Course/Particle-filtering.pdf
13. Carine Hue, Jean-Pierre Le Cadre and Patrick Perez. ***A Particle Filter to Track Multiple Objects***. Microsoft Research, Cambridge, UK. (Cited on page 15).
14. Daniel Rowe. ***Multiple Target Tracking based on Particle Filtering***. (Cited on page 15). http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/ROWE1/chapter4.pdf
15. Arnaud Doucet and Adam M. Johansen. ***A Tutorial on Particle Filtering and Smoothing: Fifteen years later.*** University of Warwick, UK. (Cited on page 15).
16. Randal Douc, Eric Moulines and Olivier Cappe. ***Comparison of Resampling Schemes for Particle Filtering***. (Cited on page 15).
17. M Armstrong, D Flynn, M Hammond, S Jolly and R Salmon. ***High Frame-Rate Television***. BBC, UK. (Cited on page 17).
18. Blaise Barney. ***POSIX Threads Programming***. Lawrence Livermore National Laboratory, USA. (Cited on page 17, 27). https://computing.llnl.gov/tutorials/pthreads/

19. Quming Zhou and J.K.Aggarwal. ***Tracking and Classifying Moving Objects from Video***. University of Texas, USA. (Cited on page 18).
20. S.Indupalli, M.A.Ali and B.Boufama. ***A Novel Clustering-Based Method for Adaptive Background Segmentation***. University of Windsor, Canada. (Cited on pages 18).
21. Le Lu and Gregory Hager. ***Dynamic Foreground/Background Extraction from Images and Videos using Random Patches***. Johns Hopkins University, USA. (Cited on page 18, 24).
22. Fredrik Kristensen, Peter Nilsson and Viktor Owall. ***Background Segmentation Beyond RGB. Lund University***, Sweden. (Cited on pages 18, 23).
23. Robert Sedgewick and Kevin Wayne. ***Gaussian Distribution***. Princeton University, USA. (Cited on page 19). http://www.cs.princeton.edu/introcs/11gaussian/
24. Yilin Su. ***Simultaneous Localization and Mapping***. University of Bristol. (Cited on page 21). http://www.cs.bris.ac.uk/Teaching/advanced/archive/ysposter.pdf
25. Tina Chen. ***Black is beautiful - 2d art with Photoshop Glowing Edges.*** (Cited on page 22). http://black2dart.blogspot.com/2009_02_01_archive.html
26. Dennis Steinauer. ***Too much time on my hands.*** (Cited on page 22). http://www.pbase.com/dsteinauer/image/37375860
27. Richard Y. D. Xu, John G. Allen and Jesse S. Jin. ***Robust Real-Time Tracking of Non-rigid Objects***. University of Sydney, Australia. (Cited on page 23).
28. Yong Shan and Runsheng Wang. ***Improved Algorithms for Motion Detection and Tracking.*** National University of Defense Technology, China. (Cited on page 23).
29. Slawomir Bogumil Wesolkowski. Color Image Edge Detection and Segmentation: ***A Comparison of the Vector Angle and the Euclidean Distance Color Similarity Measures.*** University of Waterloo, Canada. (Cited on page 23).
30. Emilio Maggio and Andrea Cavallaro. ***Multi-Part Target Representation for Color Tracking***. University of London, UK. (Cited on page 23).
31. Ognjen Arandjelovic and Roberto Cipolla. ***Achieving Illumination Invariance using Image Filters***. University of Cambridge, UK. (Cited on page 24).
32. MathWorks. ***Converting Color Data between Color Spaces***. (Cited on page 23). http://www.mathworks.com/help/toolbox/images/f8-20792.html
33. Oliver Bimber, Anselm Grundhofer, Stefanie Zollmann and Daniel Kolster. ***Digital Illumination for Augmented Studios***. Bauhaus-University Weimar, Germany. (Cited on page 24, 42).
34. OpenCVWiki. ***OpenCV 2.0 Reference***. (Cited on page 24). http://opencv.willowgarage.com/documentation/histograms.html
35. Frank Porter. ***Testing Consistency of Two Histograms***. California Institute of Technology. USA. (Cited on page 25).
36. Gary Bradski and Adrian Kaehler. ***Learning OpenCV: Computer Vision with the OpenCV Library.*** O'REILLY. (Cited on page 25).
37. Ross Johnson et.al. ***POSIX Threads for Win32***. (Cited on page 27). http://sources.redhat.com/pthreads-win32/

38. Guannan Li. ***SIFT-assist Image-based Modeling***. Tsinghua University, China. (Cited on page 39). http://media.au.tsinghua.edu.cn/liguannan.jsp
39. Xiaosong Wang. ***A SIFT Tool***. University of Bristol, UK. (Cited on page 40).
40. Tom Mathes and Justus H. Piater. ***Robust Non-Rigid Object Tracking Using Point Distribution Models***. University of Liege, Belgium. (Cited on page 40).

Appendix: Sample Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include "particles.h"
#include "observation.h"

void particleObservation(particleGroup* particleGroup, IplImage* bgrFrame) {
   CvHistogram *newHistogram = (CvHistogram *) cvAlloc(sizeof ( CvHistogram));
   double weight = 0;
   int col;

   for (col = 0; col < particleGroup->numParticles; col++) {
      newHistogram = calculateHistogram(particleGroup, col, bgrFrame);
      weight = 1.0 - cvCompareHist(newHistogram, particleGroup->histograms, CV_COMP_INTERSECT);
      weight = weight + cvCompareHist(newHistogram, particleGroup->histograms, CV_COMP_BHATTACHARYYA);
      weight = 2.0 - weight;
      //printf("w = %lf\n", weight);
      weight = exp(weight);
      //printf("exp(w) = %lf\n", weight);
      cvmSet(particleGroup->particleWeightsMat, 0, col, weight);
      cvReleaseHist(&newHistogram);
   }
}
CvHistogram* calculateHistogram(particleGroup* particleGroup, int col, IplImage* bgrFrame) {
   int hBins = 45, sBins = 20;
   int histogramSize[] = {hBins, sBins};
   float hRanges[] = {0, 180};
   float sRanges[] = {0, 255};
   float * ranges[] = {hRanges, sRanges};

   CvHistogram *histogram = (CvHistogram *) cvAlloc(sizeof ( CvHistogram));
   histogram = cvCreateHist(1, histogramSize, CV_HIST_ARRAY, ranges, 1);

   CvRect region = cvRect(cvRound(cvmGet(particleGroup->particleStatesMat, 0, col)),
         cvRound(cvmGet(particleGroup->particleStatesMat, 1, col)),
         cvRound(cvmGet(particleGroup->particleStatesMat, 2, col)),
         cvRound(cvmGet(particleGroup->particleStatesMat, 3, col)));
   cvSetImageROI(bgrFrame, region);
   IplImage* temp = cvCreateImage(cvGetSize(bgrFrame), IPL_DEPTH_8U, 3);
   cvNot(bgrFrame, temp);
cvSub(temp, bgrFrame, temp);
   cvResetImageROI(bgrFrame);

   cvNot(temp, temp);
   IplImage* hsvFrame = cvCreateImage(cvGetSize(temp), 8, 3);
   cvCvtColor(temp, hsvFrame, CV_BGR2HSV);
   IplImage* hPlane = cvCreateImage(cvGetSize(hsvFrame), 8, 1);
   IplImage* sPlane = cvCreateImage(cvGetSize(hsvFrame), 8, 1);
   IplImage * planes[] = {hPlane, sPlane};
   cvCvtPixToPlane(hsvFrame, hPlane, sPlane, 0, 0);

   cvCalcHist(planes, histogram, 0, 0);
   cvNormalizeHist(histogram, 100.0);
   cvThreshHist(histogram, 0.05);

   //cvNamedWindow("h",1);
   //cvNamedWindow("s",1);
   //cvShowImage("h", hPlane);
   //cvShowImage("s", sPlane);

   cvReleaseImage(&temp);
   cvReleaseImage(&hsvFrame);
   cvReleaseImage(&hPlane);
   cvReleaseImage(&sPlane);

   return histogram;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include "particles.h"
#include "observation.h"

particleGroup* createParticleGroup(int numParticles) {
   particleGroup *pg;
   pg = (particleGroup *) cvAlloc(sizeof ( particleGroup));
   pg->numParticles = numParticles;
   pg->particleStatesMat = cvCreateMat(4, numParticles, CV_64FC1);
   pg->histograms = (CvHistogram *) cvAlloc(sizeof ( CvHistogram));
   pg->particleWeightsMat = cvCreateMat(1, numParticles, CV_64FC1);
   pg->stateDeviationsMat = cvCreateMat(4, 1, CV_64FC1);
   CvRNG rng = cvRNG(time(NULL));
   pg->randomSeed = rng;

   return pg;
}


void intializeParticleGroup(particleGroup* particleGroup, particleState initialParticleState,
                     IplImage* bgrFrame, double* stateDeviation) {
   int col;

   for (col = 0; col < particleGroup->numParticles; col++) {
      cvmSet(particleGroup->particleStatesMat, 0, col, initialParticleState.x);
      cvmSet(particleGroup->particleStatesMat, 1, col, initialParticleState.y);
      cvmSet(particleGroup->particleStatesMat, 2, col, initialParticleState.width);
      cvmSet(particleGroup->particleStatesMat, 3, col, initialParticleState.height);
      cvmSet(particleGroup->particleWeightsMat, 0, col, 0);
   }

   particleGroup->bounds[0] = cvGetSize(bgrFrame).width;
   particleGroup->bounds[1] = cvGetSize(bgrFrame).height;

   particleGroup->histograms = calculateHistogram(particleGroup, 0, bgrFrame);
   CvMat sd = cvMat(4, 1, CV_64FC1, stateDeviation);
   cvCopy(&sd, particleGroup->stateDeviationsMat);
}
```

```
void particleTransition(particleGroup* particleGroup) {
   CvMat* transitionsMat = cvCreateMat(4, particleGroup->numParticles, CV_64FC1);
   CvMat* noisesMat = cvCreateMat(4, particleGroup->numParticles, CV_64FC1);
   CvMat* noiseArr = cvCreateMat(4, 1, CV_64FC1);
   CvMat submat;

   int row, col;
   double x, y, width, height;
   double stateDeviation;

   cvCopy(particleGroup->particleStatesMat, transitionsMat);

   for (row = 0; row < 4; row++) {
      noiseArr = cvGetRow(noisesMat, &submat, row);
      stateDeviation = cvmGet(particleGroup->stateDeviationsMat, row, 0);
      cvRandArr(&particleGroup->randomSeed, noiseArr, CV_RAND_NORMAL, cvScalar(0.0),
cvScalar(stateDeviation));
   }

   cvAdd(transitionsMat, noisesMat, particleGroup->particleStatesMat);

   for (col = 0; col < particleGroup->numParticles; col++) {
      x = cvmGet(particleGroup->particleStatesMat, 0, col);
      y = cvmGet(particleGroup->particleStatesMat, 1, col);
      width = cvmGet(particleGroup->particleStatesMat, 2, col);
      height = cvmGet(particleGroup->particleStatesMat, 3, col);

      if (width < 10.0) {
         cvmSet(particleGroup->particleStatesMat, 2, col, 10.0);
         width = cvmGet(particleGroup->particleStatesMat, 2, col);
      }
      if (height < 10.0) {
         cvmSet(particleGroup->particleStatesMat, 3, col, 10.0);
         height = cvmGet(particleGroup->particleStatesMat, 3, col);
      }
      if (x < 1.0) {
         cvmSet(particleGroup->particleStatesMat, 0, col, 1.0);
      }
      if (x + width > particleGroup->bounds[0]) {
         cvmSet(particleGroup->particleStatesMat, 0, col, particleGroup->bounds[0] - width - 1.0);
      }
      if (y < 1.0) {
         cvmSet(particleGroup->particleStatesMat, 1, col, 1.0);
      }
      if (y + height > particleGroup->bounds[1]) {
         cvmSet(particleGroup->particleStatesMat, 1, col, particleGroup->bounds[1] - height - 1.0);
      }
   }

   cvReleaseMat(&noisesMat);
   cvReleaseMat(&transitionsMat);
}

void checkValue(particleGroup* particleGroup) {
   int col;
   double x;
   double y;
   double width;
   double height;

   for (col = 0; col < particleGroup->numParticles; col++) {
      x = cvmGet(particleGroup->particleStatesMat, 0, col);
      y = cvmGet(particleGroup->particleStatesMat, 1, col);
      width = cvmGet(particleGroup->particleStatesMat, 2, col);
      height = cvmGet(particleGroup->particleStatesMat, 3, col);
      printf("x = %lf \n", x);
      printf("y = %lf \n", y);
      printf("width = %lf \n", width);
      printf("height = %lf \n", height);
      printf("-----------------\n");
   }
}

void normalizeWeights(particleGroup* particleGroup) {
   int col;
   double weight;

   CvScalar sum = cvSum(particleGroup->particleWeightsMat);
   //printf("sum of weights = %lf \n",  sum.val[0]);

   for (col = 0; col < particleGroup->numParticles; col++) {
      weight = cvmGet(particleGroup->particleWeightsMat, 0, col) / sum.val[0];
      //printf("nw = %lf\n", weight);
      //printf("nw*100 = %lf\n", weight * 100);
      cvmSet(particleGroup->particleWeightsMat, 0, col, weight);
   }
}
```

```
void resampleParticles(particleGroup* particleGroup) {
   CvMat* particleState;
   CvMat * tempParticleState;
   CvMat* newParticleStatesMat = cvCreateMat(4, particleGroup->numParticles, CV_64FC1);
   CvMat subMat1, subMat2, subMat3;

   double minWeightValue, maxWeightValue;
   CvPoint minWeightLocation, maxWeightLocation;
   int numNewParticles = 0;
   int propagation = cvRound(particleGroup->numParticles / 3);
   int counter = 0;
   int i;

   for (i = 0; i < particleGroup->numParticles; i++) {
      cvMinMaxLoc(particleGroup->particleWeightsMat, &minWeightValue, &maxWeightValue,
&minWeightLocation, &maxWeightLocation);
      numNewParticles = cvRound(maxWeightValue * (particleGroup->numParticles)) + propagation;
      propagation = cvRound(propagation * 2 / 3);
      cvmSet(particleGroup->particleWeightsMat, 0, maxWeightLocation.x, 0.0);
      particleState = cvGetCol(particleGroup->particleStatesMat, &subMat1, maxWeightLocation.x);

      while (numNewParticles > 0) {
         tempParticleState = cvGetCol(newParticleStatesMat, &subMat2, counter);
         cvCopy(particleState, tempParticleState);
         numNewParticles--;
         counter++;
         if (counter == particleGroup->numParticles)
            goto finalStep;
      }
   }

   if (counter < particleGroup->numParticles) {
      particleState = cvGetCol(newParticleStatesMat, &subMat2, 0);

      while (counter < particleGroup->numParticles) {
         tempParticleState = cvGetCol(newParticleStatesMat, &subMat3, counter);
         cvCopy(particleState, tempParticleState);
         counter++;
      }
   }
finalStep:
   cvReleaseMat(&particleGroup->particleStatesMat);
   particleGroup->particleStatesMat = newParticleStatesMat;
}
```