STUDY OF THE EFFECTS OF BACKGROUND AND MOTION CAMERA ON THE EFFICACY OF KALMAN AND PARTICLE FILTER ALGORITHMS

Yasuhiro Morita

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2009

APPROVED:

Parthasarathy Guturu, Major Professor
Kamesh Namuduri, Committee Member
Bill P. Buckles, Committee Member
Murali R. Varanasi, Chair of the Department of Electrical Engineering
Costas Tsatsoulis, Dean of College of Engineering
Michael Monticino, Dean of the Robert B. Toulouse School of Graduate Studies

Morita,Yasuhiro. Study of the effects of background and motion camera on the efficacy of Kalman and particle filter algorithms. Master of Science (Electrical Engineering), August 2009, 56 pp., 2 tables, 10 illustrations, references, 16 titles.

This study compares independent use of two known algorithms (Kalmar filter with background subtraction and Particle Filter) that are commonly deployed in object tracking applications.

Object tracking in general is very challenging; it presents numerous problems that need to be addressed by the application in order to facilitate its successful deployment. Such problems range from abrupt object motion, during tracking, to a change in appearance of the scene and the object, as well as object to scene occlusions, and camera motion among others.

It is important to take into consideration some issues, such as, accounting for noise associated with the image in question, ability to predict to an acceptable statistical accuracy, the position of the object at a particular time given its current position.

This study tackles some of the issues raised above prior to addressing how the use of either of the aforementioned algorithm, minimize or in some cases eliminate the negative effects.

ACKNOWLEDGEMENT

I am very grateful to Dr. Parthasarathy Guturu, Dr. Kamesh Namuduri, and Dr. Bill P Buckles, who formed my advisory committee. I am particular grateful to Dr. Parthasarathy Guturu for his mentoring and guidance over the years. I have had a good study and experience in the Department of Electrical Engineering, I am very grateful and I appreciate the entire faculty for having taught me a lot of materials. I enjoyed having many projects.

I am very grateful to my parents (Mr. & Mrs. Morita) for their support, both financially and morally.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

Object tracking has become a very important area in the field of computer vision. Here are some examples of areas in which it has been deployed:

1) Automated surveillance systems: These are employed in most cases for security purposes. The system monitors a scene to detect a change in scenery or suspicious activities.
2) Military Systems: Employed in the surveillance of the war zone area, to help monitor enemy troops movement. An example is the use of unmanned drones in the current ongoing war in Afghanistan.
3) Traffic Monitoring applications

Issues encountered in tracking are numerous some of which are quite complex. The issues of noise in the images have to be addressed if we are to get any discernable result. What happens when the scene illumination changes? Or what happens when there are partial or full object occlusions? How is the issue of the complexity in the shape of an object addressed? These are some of the issues that need to be addressed.

Most tracking algorithms are predicated on some basic assumptions such as "Object motion is at constant velocity" etc. In the cases studied, the object shape representation employed is the primitive geometric shape. Here, the object shape is represented by as ellipse and its motion is modeled by affine translation and projective transformations.

CHAPTER 2 - BASIC CONCEPT

Continuous Random Variable

The object tracking and motion capture in computer vision have very important idea which is continuous random variables. A continuous random variable can be set as a function that randomly plots all points in the sample plane to real numbers. For example the continuous random variable $X(t)$ plots time to position [3]. This can be explained as follow: At any point in time t (where't' is the sample plane) X_t would be the expected position or value. Probability of an experience is generally managed within some space. The probability of random variables is described as the cumulative distribution function which is from negative infinity to current value and in which the probability is accumulated [2]:

$$F_X(x) = P(-\infty, x) \tag{1}$$

The average value with weighted occurrence can be used to approximate the expected value.

$$\overline{X} = \frac{(P_1 N)x_1 + (P_2 N)x_2 + ... + (P_n N)x_n}{N} \tag{2}$$

where N is the number of samples or events. The above equation will lead to the following equation [1 2].

$$\text{Expected Value of } X = E(X) = \sum_{i=1}^{n} P_i x_i \tag{3}$$

This is for n possible results $(X_1 .....X_n)$ and the matching probabilities $(P_1 .....P_n)$ [2]

.In the case of continuous random variable, the equation of the expected value is below:

$$\text{Expected Value of } X = E(X) = \int_{-\infty}^{\infty} x f_x(x)dx \tag{4}$$

here f_x is the probability density function from $\frac{d}{dx} F_x(x)$, it is the derivative of the cumulative distribution function.

The equations 3 and 4 above can also rewrite to the following equations:

$$E(g(x)) = \sum_{i=1}^{n} P_i g(x_i) \tag{5}$$

$$E(g(x)) = \int_{-\infty}^{\infty} g(x) f_x(x)dx \tag{6}$$

$$E(X^K) = \int_{-\infty}^{\infty} x^K f_x(x)dx \tag{7}$$

if $g(x) = x - E(x)$

Therefore from above equations, the variance of X is following [2]:

$$\text{Variance of } X = E[(X - E(X))^2] = E(X^2) - E(X)^2 \tag{8}$$

Variance is a good statistical tool for random value. This is because if the variance of a signal is stable around the mean, the value of the variance is an estimate of noise in the signal [1]. Standard deviation of X is defined as $\sigma_x =$ square root of the variance of X.

In order to account for interference such as noise, we need to understand the best mathematical representation to use.

According to the central limit theorem, we can say that normal distribution is commonly assumed for a lot of the case of random processes in real world. A normal distribution is characterized by the following probability density function [2, 3]:

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp[-\frac{1}{2\sigma^2}(x - m_x)^2] \tag{9}$$

Where the expected value and squared Variance

$$m_x = \int_{-\infty}^{\infty} x f_x(x) dx \tag{10}$$

$$\sigma^2 = \int_{-\infty}^{\infty} (x - m_x)^2 f_X(x) dx \tag{11}$$

From above information, we can say that the variance can be used as the measurement of noise, but not as the information in time space. For the time space, we can use autocorrelation which is related to time and is the correlation of signal with itself. The following is the definition of the autocorrelation of a random signal X (t), in which the relationship between autocorrelation and time is clearly mentioned and is the function of time, on the other hand spectral density function is accepted in the frequency domain [2]:

$$R_X(t_1, t_2) = E[X(t_1)X(t_2)] \quad \text{for sample times } t_1, \text{ and } t_2 \tag{12}$$

For white noise, the autocorrelation function is taken as a Dirac delta function $\delta(\tau)$.

This function [3] has a value of zero everywhere except when $\tau = 0$

$$R_x(\tau) = \begin{cases} \text{if } \tau = 0 \text{ then A} \\ \text{else 0} \end{cases} \quad \text{for some constant magnitude A} \tag{13}$$

From the above equation with condition, if we use the Fourier transforms, the white noise can be constant in frequency domain, so spectrum density function is flat.

This describes in mathematical terms the behavior of white noise. It can be thought that there are constant power at any frequency in the frequency domain and being completely correlated with itself only at $\tau = 0$. It is the reason why white noise signals are mentioned as independent, because for any sample of the signal except for $\tau = 0$, the current sample is independent from other samples. [2]

Image Feature Extraction

To be able to track an image effectively, it is important to extract features of different types from the image. These features are used for identification and tracking purpose. There are three types of very important feature sets that are commonly used in object tracking. They are color features, Texture features and Intensity gradient features.

Color Features

Because the color feature generally doesn't change for rotation and scaling, it can be used for a lot of applications. It has the following three color spaces [3, 4, 5]:

a) Red green and blue color image. Also known as RGB color image. We can get these values directly from the image.

b) Hue-saturation-value (HSV): An important feature for human insight of color. HSV is a representation of points in an RGB color image, which shows the perpetual color relationships more accurately than RGB and also keeping its

computational simplicity [3, 4, 5].

c) r, g, b: r, g, and b for each pixel are obtained by R divided by R+G+B, G divided by R+G+B and B divided by R+G+B. this is normalized value. This scaling of RGB to obtain r, g, and b values offers additional robustness against changes in illumination [3, 4, 5].

Texture Features

Texture is a function of the pattern in pixel intensities (gray values) [13]. These can be obtained by using several mathematical tools such as: co-occurrence matrices, wavelet packets and Gabor filters [4].

This feature can be used as set with color feature for object tracking, object recognition and any other topic in computer vision to achieve more accurate result.

Intensity Gradient Features

It is important to be able to differentiate between the background and foreground images. The intensity gradient features prove capable of achieving this purpose. It can be obtained by using sobel mask, which is used to find horizontal and vertical differential images. For the strong edges, the gradient direction is determined, and they will go above the certain threshold value. There are some directions for edge of image to reach the robustness against changing level of intensity for background and foreground for the path of the motion object [3, 4, 5]. It is important that values of this feature set are quantized into a given number of levels, and in so doing we assumed that the gradient vectors with

opposite directions are at the same quantized levels as their counterpart in the contrary direction, as long as they have same initial value and only differ in direction prior to quantization.

The result is such that "the edge feature set at any pixel site is has at least one feature value corresponding to the direction of the gradient equal to one" [4]. The condition is set such that if a pixel is not edgel, all set of edge feature values for that pixel will equate to zero [3, 4, 5].

Features Extraction

The object region is divided into two frames (reference and target frames). They both need to be corresponding based on chosen features. The sets of features extracted from the images [3, 4] (reference and target) are first and foremost normalized such that they exist in the region $[0, 1]$.

It is important that the feature extractor chosen for extraction is capable of extracting features that contain substantial information in order to facilitate easy separation between background and foreground pixels [3, 4].

According to the author of [4], the method used is described below:

1) Find the covariance of the feature vectors of foreground and background of observed pixels centered on the respective class means". This is denoted as "S_W" [4].
2) Find "the covariance matrix of the class mean vectors centered on the overall sample mean". This is denoted as "S_B" [4].

Apply Fischer's discrimination function which is that the main few Eigen vectors of the matrix "e" will give the feature extractor and it is from the Eigen value equation, [4]:

$$Ae = \Lambda e \tag{14}$$

Where $A = S_W^{-1}S_B$ and Λ = Eigen value matrix of the system.

The foreground and background pixel can be now obtained from the reference and target frame observation windows, and the foreground and background are depending on as if they are located in the observation window or not[3, 4, 5].

The features of the equivalent pixels in the reference and target frames are used to compute the four sample mean vectors, and "S_W" is now computed by using the pattern vector samples minus the respective class mean

The matrix "S_B" is calculated from four samples as shown below [4]:

Let u = Overall sample mean, u_{fg}^{T} = Target sample mean, u_{fg}^{R} = Background Class mean, and "t" = Transpose of the column matrices.

The matrix "S_B" is thus computed from follow [3, 4].

$$\begin{aligned}
&(u_{fg}^{T} - u)(u_{fg}^{T} - u)^{t},\\
&(u_{fg}^{R} - u)(u_{fg}^{R} - u)^{t},\\
&(u_{bg}^{T} - u)(u_{bg}^{T} - u)^{t},\\
&(u_{bg}^{R} - u)(u_{bg}^{R} - u)^{t}
\end{aligned} \tag{15}$$

Finding Similarities in Measurements

This computation is used for image matching purpose and carried out by independently computing these similar measurements for each of the three feature vector components in the reduced feature space. It is aggregated via a process called feature fusion [4].

The first step is to quantize all possible aspect values in L levels of the region. L (x_i) represents a "function that maps feature value of a pixel located at "x_i" to a level "L", where $L \in [1, L]$" [4].

The discrete feature probability distribution ($P_t(x) = \{P_t^L(x)\}_{L=1.....L}$) which is the probability function inside of a bounding ellipse is defined. The region inside of ellipse is centered at "x" and the object should be inside of it [4].

$P_t^L(x)$ is calculated as follow [3, 4]:

$$P_t^L(x) = C\sum_{i=1}^{N_R} K(\frac{\|x - x_i\|}{h})(L(x_i) - l) \qquad (16)$$

where x_i = Location of pixel inside of the ellipse. N_R is the number of pixels inside of ellipse. K is the Kernel function. δ is the Kronecker delta function. $\| \ \|$ is Euclidean Norm. h and C are normalization constants that make certain that the function K and the sum respectively are identified [4].

$$h = \sqrt{a^2 + b^2} \qquad (17)$$

here "a" and "b" are the major and minor axis of the ellipse divide by two.

$$C = \frac{1}{\sum_{i=1}^{N_R} K(\frac{\|x - x_i\|}{h})} \tag{18}$$

This is for pixels within an elliptic region centered at a point “x”. The elliptic region is assumed to enclose the object fully.

The kernel function can be chosen by the main idea that if the pixels given by the feature information are far from the center point of the ellipse which means the pixels are more similar to background, these samples are less important and weights of them should be reduced to ignore these samples. This idea gives the following kernel function [3, 4, 5]:

$$\begin{cases} K(r) = 1 - r^2 & \text{if } r < 1 \\ \text{Else } = 0 & \text{otherwise} \end{cases} \tag{19}$$

After the feature probability distribution inside of ellipse for target and reference frames centered at X_{tgt} and X_{ref} is computed, the Bhattacharyya distance formula can be used to compute the match of them [4]:

$$d_m = \sqrt{1-\rho(P_t(X_{tgr}), P_t(X_{ref}))} = \sqrt{1-\sum_{L=1}^{L}\sqrt{P_t^L(X_{tgt})P_t^L(X_{ref})}} \tag{20}$$

From the above equation ρ is the sum term in the equation on the right hand side.

This term is called Bhattacharyya coefficient. It gives a measure of the similarity between the distributions, and if ρ is close to 1, then two regions are more similar [3, 4, 5].

CHAPTER 3 - KALMAN FILTER

Basic concept of Kalman Filter

A Kalman filter is the estimation tool which consists of available measurement values and previous states of the system and also the error of the estimation is tried to be minimized for desired variable [15]. The algorithm is recursive processing in which the Kalman filter estimate and then correct the values to get more accurate results for every recursive loop. Figure 1 shows the examples of the Kalman filter application [15].

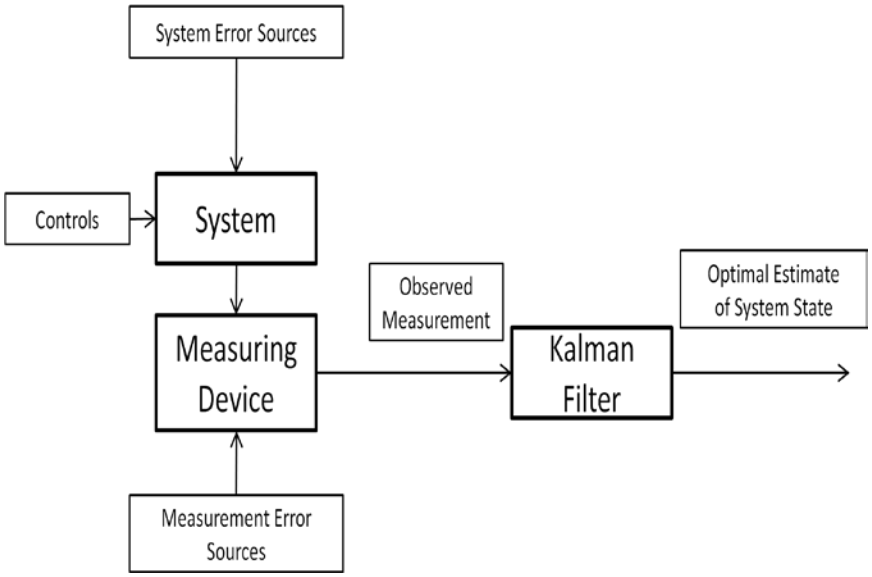

Figure 1: Typical Kalman Filter Application

According to the author in [15], the three following are important information that is required to pass on to the Kalman filter:

1) “Knowledge of the system and measurement device dynamics”.

2) “Statistical description of the system noises, measurement errors and uncertainty in the dynamic models”.

3) “Any available information about the initial conditions of the variables of interest”.

From the Bayesian point of view, the filter should increase the conditional probability density of the value we want, predicated on the actual data information from the measuring tools.

$$f_{X(i)|Z(1),Z(2),.....Z(i)}(x|Z_1, Z_2, ...Z_i) \tag{21}$$

The above equation can be interpreted as the conditional probability density of the value of “x” at any time “i”. And this means that $X(i)$ can be obtained by information of other samples which has probabilities (or quantities) from $Z(1)$ through $Z(i)$. After a conditional probability density can be obtained, it will lead the optimal prediction and following ideas are led:

1) The Mean is defined as the middle point of probability quantity for approximation.
2) The mode is defined that the probability which value is highest on the peak of probability density function.
3) The median is defined as the point on the center of x axis.

   For these ideas, A Kalman filter should be used for a linear system and white Gaussian [15].

Linear System Model

Whiteness implies no correlation of the noise in time domain and constant power for frequency domain. Kalman filter tries to predict the intermediate state of the discrete time system by using following equation [2, 3]:

$$X_K = AX_{K-1} + Bu_{K-1} + W_{K-1} \tag{22}$$

We also have a "measurement residual" [1, 2, 3] given by the following equation:

$$Z_K = HX_K + V_K \tag{23}$$

The meaning of the above two equations are as follow:

"A" is an "n x n" matrix. It will be multiplied to the previous state 'X" at time " $K-1$" to obtain the current state at time "K".

"B" is an "n x L" matrix. It will be multiplied to the optional control vector" $u \in \Re^1$ ".

"H" is an "m x n" matrix that will be multiplied to the current prediction for the measurement " Zk "

W_K and V_K are the random variable of the noise for the estimation and measurement. The following are assumed concerning the W_K and V_K parameters [2]:

1) They are independent of each other.
2) They are white Gaussian noise:

$$P(w) \sim N(0, Q) \tag{24}$$

$$P(v) \sim N(0, R) \tag{25}$$

Where "Q" is covariance of the estimation noise and "R" is covariance of the measurement residual noise with constant [1, 2].

Computation

We set the $\hat{X}_K^- \in \Re^n$ as a "priori state" which is current state obtained by previous state and The $\hat{X}_K \in \Re^n$ is a "posteriori state" which is also the current state obtained by the measurement residual Z_K. The prior estimate error is defined as:

$$\hat{e}_K \equiv X_K - \hat{X}_K^- \tag{26}$$

The a posteriori estimate error is defined as:

$$e_K \equiv X_K - \hat{X}_K \tag{27}$$

The a priori estimate error covariance is defined calculated as follow:

$$P_K^- = E[e_K^- e_K^{-T}] \tag{28}$$

The a posteriori estimate error covariance is calculated as follow:

$$P_K = E[e_K e_K^T] \tag{29}$$

The goal is to obtain the value of posteriori that can be calculated by the addition of the priori value and the difference of actual measurement and priori with $H_{\hat{X}_K^-}$ with gain which imply the linear system [1]. The equation is shown below:

$$\hat{X}_K = \hat{X}_K^- + K(Z_K - H\hat{X}_K^-) \tag{30}$$

$(Z_K - H\hat{X}_K^-)$ is called the residual. If this difference is zero, it means priori value is the actual estimated value [1, 2].

The name of "K" in the above equation is Kalman gain and it is an n x m matrix.

It is for the minimization of a posteriori error covariance which can be obtained by substituting equation (28), (29) in equation (30). To minimize K, the derivation of the equation can be done with respect to K and set it to zero, and then it will be the following result [1, 2]:

$$K_K = \frac{P_K^- H^T}{H P_K^- H^T + R} \tag{31}$$

From above equation, if the error covariance R of the measurement reach to zero, the denominator and numerator have the common terms and they can be canceled out, then we can obtain that $K_K = H^{-1}$ [1, 2, 3]. In this case, we can also say that if the R is almost close to zero, Zk has no error which mean the measurement is more accurate, but the estimated measurement is less [1, 2]. However if the error covariance of priori state P_K^- is close to zero which mean the priori estimation is reliable more, the gain "K" can be calculate by zero divide by R which means "$K_K = 0$". From these ideas, the actual measurement is less reliable but the predicted measurement is more trustable.

Figure 2 shows the main idea of the Kalman filter. From the above theories, the Kalman filter has two important steps which names are predictor and corrector to calculate priori state and posteriori respectively and [1, 2, 3]:

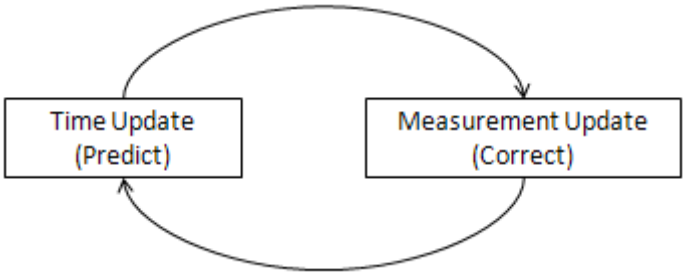

Figure 2: Kalman Filter Cycle

Initially the priori can be obtained by equation (22) in predictor and then posteriori can be calculated by equation (30) in the corrector [1]. The corrector is calculated for the purpose to achieve the more accurate estimation.

In the predictor part, by using the previous estimation result of posterior and its covariance, the current priori can be obtained with its covariance [1, 2].

The corrector engages to update for obtaining the more accurate estimation by using measurement to calculate the posteriori and its covariance for next estimation [1, 2, 3].

Time predictor equations are as follow:

$$\hat{X}_K^- = A\hat{X}_{K-1} + Bu_{K-1} \tag{32}$$

$$P_K^- = AP_{K-1}A^T + Q \tag{33}$$

where P_{K-1} is the result of the covariance of posteriori which is calculated in previous step.

Measurement corrector equations are as follow:

$$\hat{X}_K = \hat{X}_K^- + K(Z_K - H\hat{X}_K^-) \tag{34}$$

$$P_K = (1 - K_K H)P_K^- \tag{35}$$

where K is from equation (31).

In the corrector part, initially the Kalman gain can be calculated by using equation (31) by using prior covariance, H' matrix and noise R. after that the measurement Z_K can be obtained by calculation or observation, and then the posteriori can be obtained by

using equation (34). Finally the error covariance of posteriori can be obtained by equation (35) for next the recursive loop. For the estimation of next point, this recursive loop can be continued[1, 2].

CHAPTER 4 - PARTICLE FILTER

Particle Filter

It is also known as sequential Monte-Carlo technique, which is a kind of recursive Bayesian based on Monte-Carlo simulation. The state space can be allocated into many regions and inside of the region, the particles or samples can be obtained based on the certain probability assumption. If the probability is higher, density of the samples will be increased and useful [10]. The posterior probability density is given by a weighted sum of "N_P" samples which is from the region based on the certain probability distribution given by the following equation [5, 6, 8, 9, 10]:

$$P(X_n \left| y_n \right.) \approx \frac{1}{N_P} \sum_{n=1}^{N_P} \delta(X_n - X_n^{(i)}) \equiv \hat{P}(X_n \left| y_n \right.) \qquad (36)$$

where $X_n^{(i)}$ can be independent and identically distributed (i,i,d). They are led from "$\hat{P}(X_n \left| y_n \right.)$" if the number of samples or particles "N_P" is enough, the priori probability $\hat{P}(X_n \left| y_n \right.)$ in the above equation can be assumed to the correct posterior "$P(X_n \left| y_n \right.)$"[10]. This approximation is used to predict the mean of a non-linear function as shown in the equation below [7, 8, 10, 11]:

$$\begin{aligned} E[f(X_n)] \approx \int f(X_n) \hat{P}(X_n \left| y_n \right.) dX_n &= \frac{1}{N_P} \sum_{i=1}^{N_P} \int f(X_n) \delta\delta(\ _n - X_n^{(i)}) dX_n \\ &= \frac{1}{N_P} \sum_{i=1}^{N_P} f(X_n^{(i)}) \equiv \hat{f}_{N_p}(x) \end{aligned} \qquad (37)$$

Sampling is done from a proposal distribution" $q(X_n|y_n)$ ", because sampling of a true posteriori is impracticable. Therefore equation (37) can be rewritten as shown below [7, 8, 10, 11]:

$$\begin{aligned} E[f(X_n)] &= \int f(X_n)\frac{P(X_n|y_n)}{q(X_n|y_n)}q(X_n|y_n)dX_n = \int f(X_n)\frac{W_n(X_n)}{P(y_n)}q(X_n|y_n)dX_n \\ &= \frac{1}{P(y_n)}\int f(X_n)W_n(X_n)q(X_n|y_n)dX_n \end{aligned} \quad (38)$$

$$\text{where } W_n(X_n) = \frac{P(y_n|X_n)P(X_n)}{q(X_n|y_n)} \quad (39)$$

Equation (38) can be written as follow [7, 8, 10, 11]:

$$\begin{aligned} E[f(X_n)] &= \frac{\int f(X_n)W_n(X_n)q(X_n|y_n)dX_n}{\int P(y_n|X_n)P(X_n)dX_n} \\ &= \frac{\int f(X_n)W_n(X_n)q(X_n|y_n)dX_n}{\int W_n(X_n)q(X_n|y_n)dX_n} \\ &= \frac{E_{q(X_n|y_n)}[W_n(X_n)f(X_n)]}{E_{q(X_n|y_n)}[W_n(X_n)]} \end{aligned} \quad (40)$$

Because of the samples $\{X_n^{(i)}\}$ from $q(X_n|y_n)$ which is independent and identical distribution, now equation (40) above can be revised as follow [10]:

$$E[f(X_n)] \approx \frac{\frac{1}{N_P}\sum_{i=1}^{N_P} W_n(X_n^{(i)})f(X_n^{(i)})}{\frac{1}{N_P}\sum_{i=1}^{N_P} W_n(X_n^{(i)})} = \sum_{i=1}^{N_P} \tilde{W}_n(X_n^{(i)})f(X_n^{(i)}) \equiv \hat{f}(X) \quad (41)$$

$$\text{Where } \tilde{W}_n(X_n^{(i)}) = \frac{W_n(X_n^{(i)})}{\sum_{j=1}^{N_P} W_n(X_n^{(j)})} \quad (42)$$

To make the simple implementation, probability related to all the previous state $P(X_{0:n}|y_{0:n})$ cannot be considered, instead of this, we consider the current state estimation $P(X_n|y_{0:n})$. If we consider that $q(X_n^{(i)}|X_{0:n-1}^{(i)}, y_{0:n})$ is equal to $q(X_n^{(i)}|X_{0:n-1}^{(i)}, y_n)$, then the following equation can be obtained from equation (42) [4, 5, 10]:

$$W_n^{(i)} = W_{n-1}^{(i)} \frac{P(y_n|X_n^{(i)})P(X_n^{(i)}|X_{n-1}^{(i)})}{q(X_n^{(i)}|X_{0:n-1}^{(i)}, y_n)} \tag{43}$$

Weight degeneration is very important issue, which occurred after any iteration, and because of this, the important weights will become skewed, and a lot of the weights of the particles will be equal to zero, so estimation will be failed. It is important to use the sampling importance resampling to avoid the weight degeneration and obtain the effective sampling size equation [6, 7, 8, 10, 11]:

$$N_{eff} = \frac{N_P}{1 + Var_{q(.|y_{0:n})}[\tilde{W}(X_{0:n}]} = \frac{N_P}{E_{q(.|y_{0:n})}[(\tilde{W}(X_{0:n}))^2]} \leq N_P \tag{44}$$

The above equation can be obtained from:

$Var[\xi] = E[\xi^2] - E[\xi]^2$, and $E_q[\tilde{W}] = 1$. But, in real world, the correct Neff is not obtainable, thus the following equation gives its prediction [10]:

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N_P} (\tilde{W}_n^{(i)})^2} \tag{45}$$

Re-sampling steps can be done when $(\hat{N}_{eff})$ falls less than a predetermined threshold value (N_T). If it happens, each sample can be chosen with a minimum probability of $\{1, W_n^{(i)} / N_T\}$ and it will have new weights $W_n^{(j)} = \max\{N_T, W_n^{(i)}\}$.

The following table describes a general algorithm used in SIR particle filter based on the above equations [7, 8, 10, 11]:

Table 1: Pseudo Code of Particle Filter

For Time steps n=0, 1, 2,…….

1: Initialization: For i=1,……., N_P,

Sample $X_0 \sim P(X_0)$,

$$W_o = \frac{1}{N_P}$$

2: Importance Sampling:

For $i = 1,....., N_P$,

Draw samples $\hat{X}_n^{(i)} \sim P(X_n | X_{n-1}^{(i)})$,

Set $\hat{X}_{0:n}^{(i)} = \{X_{0:n-1}^{(i)}, \hat{X}_n^{(i)}\}$

3: Weight Update:

Calculate Importance Weights $W_n^{(i)} = P(y_n | \hat{X}_n^{(i)})$

4: Normalize the Importance weights: $\tilde{W}_n^{(i)} = \frac{W_n^{(i)}}{\sum_{j=1}^{N_P} W_n^{(j)}}$

5: Re-sampling: Generate (N_P) new particles $(X_n^{(i)})$ from the set $\{\hat{X}_n^{(i)}\}$ according to the importance weights $(\tilde{W}_n^{(i)})$

6: Repeat steps 2 to 5

CHAPTER 5 - IMPLEMENTATION

In this section, I will talk about the overview of this implementation (experiment), after that I will also roughly talk about the Kalman filter and particle filter again and then, the details of algorithms and structures of them will be discussed. The Matlab code of these algorithms will also be included and explained in these sections.

Implementation Overviews

To achieve the goal which is the study of the effects of the background on the efficacy of Kalman and particle filter, I implemented the both Kalman and particle filter algorithm by using Matlab, and I used the two methods to extract the object: background subtraction and color histogram, respectively.

Overviews of Kalman Filter and Particle Filter

The Kalman filter is the important mathematical device to obtain the state for stochastic process from noisy measurements, in other words, it is to predict the future state by using the current state from the observation and estimated state from previous states. This method has two different categories: a predictor and a corrector. In the predictor, the current estimated state can be calculated by addition of the transition model multiplied by previous estimated state, the control input model with

control vector, and noise (equation 32). The covariance of the estimated state can be calculated for update process. In the corrector part, the Kalman gain can be obtained and it is multiplied by the value of the subtraction of the observation of the center point and the estimated value in predictor. This value is added to the estimated value again to achieve final estimation and the updated covariance is also calculated again (equation 34).

Particle filter generally has the posterior density and the observation density which are non Gaussian. And the main idea of particle filter is that in order to estimate the object state, the weighted samples which consist of multiplications of estimated samples and weights will be obtained. The color feature is also important. To consider object, background, and the occlusion, the object model is obtained by color distribution, and they are calculated by histograms. To achieve better color distribution of the object model means that the observed value of the image is provided. Same as Kalman filter, the difference of observed value and estimated value can be obtained to update the final estimated point.

Kalman Filter Algorithm

In previous section, I discussed the theoretical overview of Kalman filter and particle filter. In this section, I will discuss the actual algorithm of Kalman filter. The Figure 2 shows the brief idea of object tracking algorithm with Kalman filter.

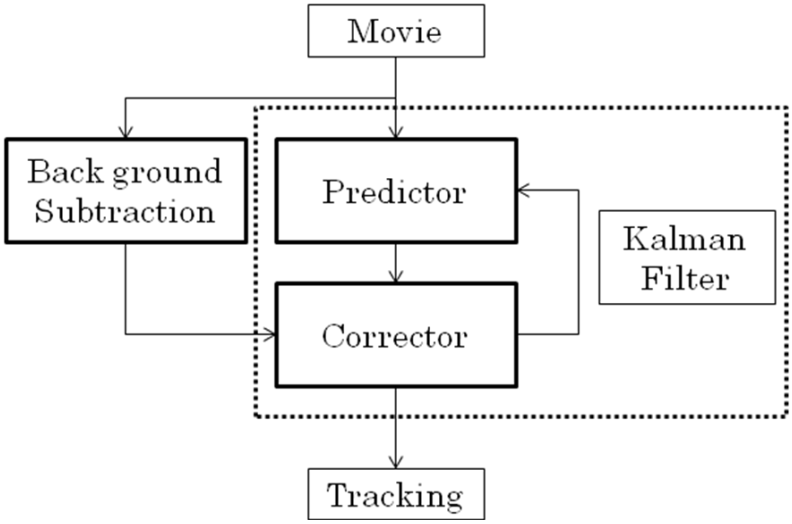

Figure 3: Structure of Kalman Filter Algorithm

From the videos, I tried to obtain the information of each frame and their respective background information. Each frame image is used to estimate the center point of the object, however, the background image are used to find the observation point. Initially I read the several frames of videos which don't include the target object, and then divided the sum by the number of frames. This process gives the background information.

In predictor, I predict the state X which constructs the matrix [5, 10]:

$$X = [\, x \;\; x_v \;\; y \;\; y_v \;\; H_l \;\; H_s \;\; H_r \,] \tag{46}$$

Where x is the x axis value of the center point of the object, x_v is the x axis velocity of the center point of the object, y is the y axis value of the center point of the object, y_v is the y axis velocity of the center point of the object, H_l is the semimajor axis value of the ellipse, H_s is the semiminor axis value of the ellipse, H_r is the angle velocity of the ellipse.

I set the transition matrix A as[5, 10]:

$$A = \begin{bmatrix} 1 & dt & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{47}$$

By using these values, I calculate the predicted state Xp [2, 16]

$$Xp = A \times X(i-1, :)' + Ck \times N(i-1, :) \tag{48}$$

where $Ck \times N(i-1, :)$ are the control vector multiplied with the process noise (see Matlab Code in appendix).

Also the initial error covariance is as:

$$P = 100 \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{49}$$

The process noise covariance Q is:

$$Q = 0.01 \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{50}$$

By using these values, I calculated the predicted state covariance.

In corrector side, once the predicted state and its covariance are obtained, I calculate the Kalman Gain K [2]:

$$K = PP \times H' \times \text{inv}\ (H \times PP \times H' + R) \tag{51}$$

where the observation model matrix H is:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{52}$$

And R is the measurement noise covariance (see Matlab Code in appendix). The measurement residual yk is obtained by:

$$yk(:, i) = K \times ([cc(i), cr(i)]' - H \times Xp) \tag{53}$$

where $[cc(i), cr(i)]$ is the observation point from background subtraction. To correct the estimate with measurement, I calculated X and the error covariance:

$$\begin{aligned} X(i, :) &= (Xp + yk(:, i))' \\ P &= (\text{eye}(7) - K \times H) \times PP \end{aligned} \tag{54}$$

By using these updated values, I estimated the center point for next frame again.

Particle Filter Algorithm

In the previous section, I discussed about the algorithm of Kalman Filter with background subtraction. In this section I will mention about the algorithm of particle filter. The Figure 4 shows the structure of object tracking algorithm with particle filter. From Figure 4, you can see that my algorithm is based on the sampling importance resampling (SIR).

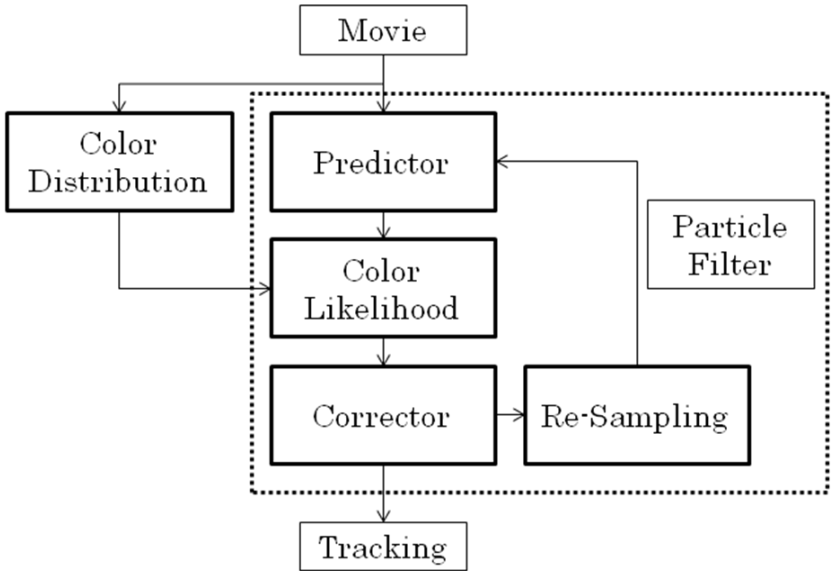

Figure 4: Particle Filter Algorithm

Initially the video file will be read into Matlab, and first frame of video is extracted to get color distribution. There are initially RGB (red, green, blue) components in the image and they are converted to HSV (hue, saturation, value). From these values, the histogram, the cumulative distribution function and edges of HSV can be obtained.

In particle filter [5, 10, 16], I set the state as Smean

$$\text{Smean} = [\,x \;\; x_v \;\; y \;\; y_v \;\; H_l \;\; H_s \;\; H_r\,] \qquad (55)$$

where x is the x axis mean (weighted) value of the center point of the object, x_v is the x axis mean (weighted) velocity of the center point of the object, y is the y axis mean (weighted) value of the center point of the object, y_v is the y axis mean (weighted) velocity of the center point of the object, H_l is the semimajor axis mean (weighted) value of the ellipse, H_s is the semiminor axis mean (weighted) value of the ellipse, H_r is the mean (weighted) angle velocity of the ellipse. And the final goal of the algorithm is to estimate $Smean$ by:

$$Smean = \text{mean of } (\text{Weighted Samples (Particles)} \times \text{Predicted States}) \tag{56}$$

In the predictor, I set the transition matrix A as [14, 16]:

$$A = \begin{bmatrix} 1 & dt & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{57}$$

By using these values, I calculated the predicted state Sk

$$Sk = A \times Sk + Ck \times \text{randn}(d, N) \tag{58}$$

where Ck is the covariance of the noise and $\text{randn}(d, N)$ is the random noise. These values are similar to Kalman filter's information, however, still there are some differences. For example, Sk is not a vector, the value N is the number of particles and Nx7 matrix is be constructed. The value Ck is 7x7 matrix which is calculated by the following equations[16]:

$$Rk = \begin{bmatrix} Ry & 0 \\ 0 & Re \end{bmatrix} \tag{59}$$

$$Ry = sigmay \times \begin{bmatrix} delta^3/3 & delta^2/2 & 0 & 0 \\ delta^2/2 & delta & 0 & 0 \\ 0 & 0 & delta^3/3 & delta^2/2 \\ 0 & 0 & delta^2/2 & delta \end{bmatrix} \tag{60}$$

$$Re = \begin{bmatrix} sigmaHx^2 & 0 & 0 \\ 0 & sigmaHy^2 & 0 \\ 0 & 0 & sigmatheta^2 \end{bmatrix} \tag{61}$$

$$Ck = chol(Rk)' \tag{62}$$

where Rk, Ry and Re is the kinematic of the ellipsoid. And then Ck is calculated by the Cholesky factorization of Rk. Once I predict the state Sk, I have to check the similarity of observed and target image to update the state Sk. I obtained the color distribution Py of an area R from following equation [5, 11]:

$$Py^{(u)} = f \sum_{x_i \in R} k\left(\frac{\|y - x_i\|}{a}\right) \delta[h(x_i) - u] \tag{63}$$

where δ is the Kronecker delta function and one of the m-bins of the histogram at x_i is allocated to $h(x_i)$, a gives invariance against range of the area and f is the normalization factor ensured by the condition [5, 11]:

$$\sum_{u=1}^{m} Py^{(u)} = 1 \tag{64}$$

As I mentioned in Chapter 2, I used the Bhattacharyya coefficient [12]. Based on these values, I found the observation probability distribution function so that I will weigh each particle. The equation is:

$$\begin{aligned} W &= W \times \text{Color PDF} \\ W &= W \Big/ \text{Sum}(W) \end{aligned} \tag{65}$$

From the equation [58, 65], I will calculate the S_{mean}. However, if the following condition is satisfied, I have to implement the resampling to avoid the degeneracy.

$$N_{eff} < N_{threshold} \tag{66}$$

where N_{eff} is calculated by the following equation [5 10]:

$$N_{eff} = 1 \Big/ \sum W^2 \tag{67}$$

CHAPTER 6 - SIMULATION & RESULTS

Introduction of Simulations

To observe the effects of background on the efficacy of Kalman filter and particle filter algorithm, I carried out the experiments for object tracking with three videos by using Kalman filter with back-subtraction and particle filter. The first video is the one that shows the red color ball freely falls to the floor and bounced several time, and then stop on the floor, and the background of this video is just simply black color. The second video is the one that shows the red car moving on the road, and the background of this is the outside scene, which means this has more complex color histogram than that of the first video. The third video is the one that shows a scene on the highway. Camera faced to the backside. As initial assumption, the cameras of both first and second videos are fixed and motionless so that I can get the stable backgrounds of video scene easily, however in the third video the camera is unstable and moving because the camera is also on the car and it is moving, so obtaining the background picture was difficult (see Table 2). For these videos, applied Kalman filter and particle filter algorithms and then compared them.

Table 2: Video Information of Simulation

	Video 1	Video 2	Video 3
Object	Ball	Car	Car
Number of Objects	1	1	2
Background	Simple	Complex	Complex
Camera	Motionless	Motionless	Motion

Video 1

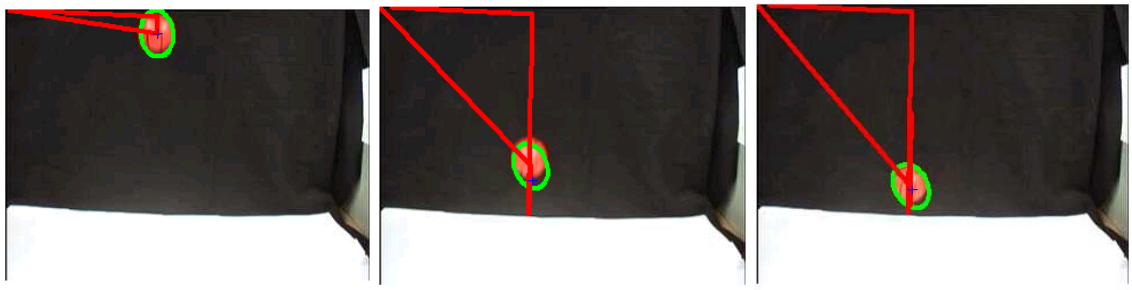

Figure 5: Kalman Filter Algorithm for Video - 1

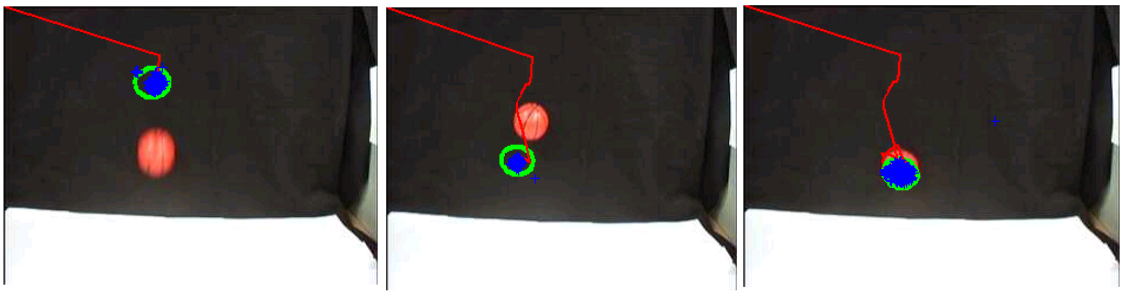

Figure 6: Particle Filter Algorithm for Video - 1

Figure 5 shows the results of the Kalman filter algorithm for the first video. From these pictures, it shows this algorithm is acceptable for object tracking. This is because the motionless camera provides a clear background and the background subtraction is very accurate, however, some errors still exist. These are caused by the actual Kalman filter error. On the other hand, in Figure 6, the particle filter algorithm is not working properly. The main reason why it lost track is that the speed of the falling object is not acceptable for this algorithm. In every frame, from the estimated center point, the particles are chosen based on the color PDF in the ellipse, however, if the object speed is increasing dramatically, the object easily goes out of the ellipse, thus the similarity of the

color PDF of target and reference no longer close. This misleads the extraction mechanism for the object.

Video 2

The results of the Kalman filter algorithm for the second video in Figure 7 is similar to that for the first video. The similarity of the situation gives acceptable results. The tracking in the second video is more accurate than that in first video because the object is not moving fast. On the other hand, the results of the particle filter algorithm for second video in Figure 8 dramatically changes. One of the reasons is the reduced speed of object. The second reason is that the color histogram of background is more complex than that of first video. This means that the algorithm can receive more information for the background.

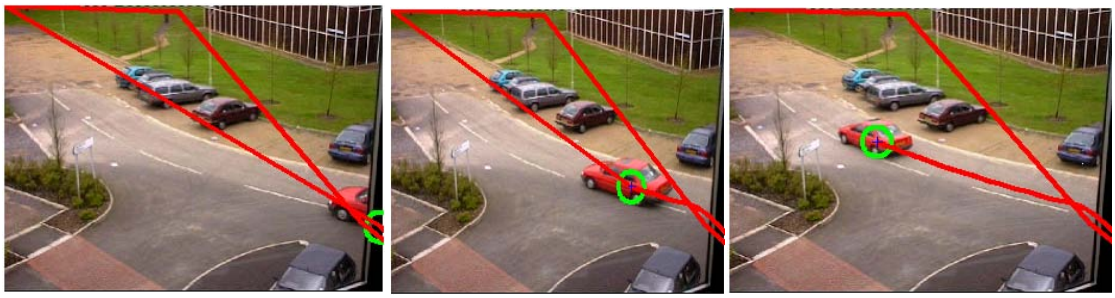

Figure 7: Kalman Filter Algorithm for Video - 2

Figure 8: Particle Filter Algorithm for Video – 2

Video 3

The results of Kalman filter algorithm of the third implementation is in Figure 9. There are two objects in which one on them is the right side big car moving from right side to center, and another is the normal car moving from center and to left side, and initially the ellipse is on the big car, however, although I wanted to track first objects, I can obtain the results coincidentally. It is because the background can be calculated just before the first frame, and first object is bigger, so coincidentally the object is extracted, however, the fact that the camera is moving make the error of the similarity between the reference background and the target background increasing as time goes by. The ellipse is moving up and down many times and loses its way.

On the other hand, in figure 10, the particle filter algorithm is different from the Kalman filter algorithm. Instead of the background subtraction, it uses the color histogram extraction. When the camera is moving, color histogram of the object does not change, and this leads to accurate results. Also there is one problem about this algorithm. I had to set the initial center point of the object manually.

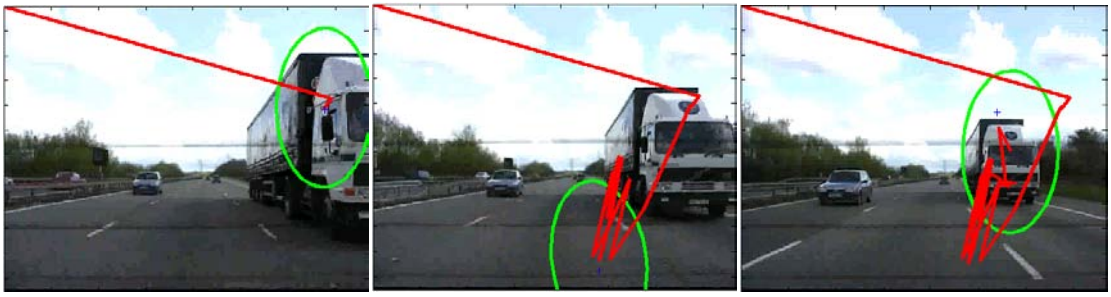

Figure 9: Kalman Filter Algorithm for Video - 3

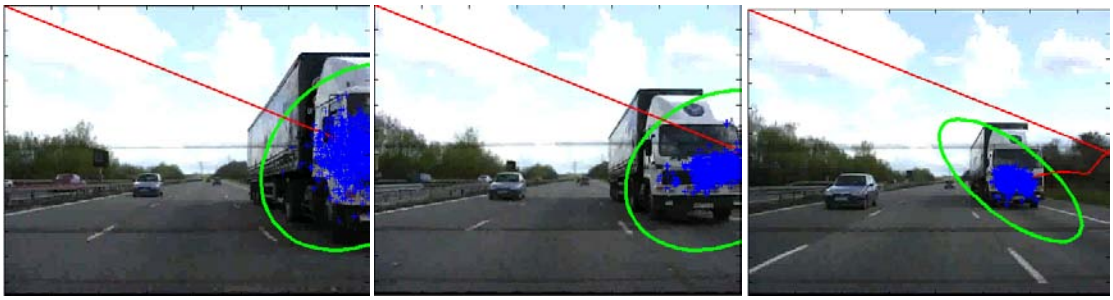

Figure 10: Particle Filter Algorithm for Video - 3

The Kalman filter with background subtraction is suitable method if the camera is motionless, and it is also fast to implement, however, because of simplicity, it is not flexible for any situation. On the other hand, the particle filter with color feature is complex to implement and speed is not so fast, but it still has an acceptable run time, although I had to set the initial center point manually and sometime the noise may cause serious error, the result can be valid, and it is adequate.

CHAPTER 7 - CONCLUSION

In this thesis, two algorithms for object tracking which are Kalman filter with background subtraction and Particle filter with color feature are compared. In order to access the performance of both methods, three different types of videos were used. From the first two videos, we can conclude that Kalman filter provides a better performance. This is due to the fact that Particle filter requires a longer transition time. Moreover, when Gaussian noise is assumed, Kalman filter is better.

On the other hand, from the results of Kalman algorithm for all the three videos we can say that the motionless camera does not provide the correct information for the background subtraction method. From those points, the study of the effects of background on the efficacy of Kalman and Particle filter algorithm are met. The background subtraction method is best for motionless camera.

For fast implementation with motionless camera, the background subtraction with Kalman filter is a better algorithm. Additionally, particle filter with color feature can be applied to any video whether the camera is moving or is motionless.

Finally, for actual implementations or applications, the Particle filter is a better method.

For future work, we will try to extract in addition to the object and background based on the color feature, the texture and the shape feature in the aim of obtaining a more accurate result. Furthermore, real time processing could be considered for surveillance and robotics applications.

APPENDIX

MATRAB CODE

The codes are originally created by Alireza Kashanipour and Sebastien Paris from the Matlab center and they are modified by me to achieve to desired results.

Kalman Filter Algorithm (Video 1)

```
clear,clc
% compute the background image
% Kalman filter initialization
Imzero = zeros(240,320,3);
for i = 1:5
Im{i} = double(imread(['DATA/',int2str(i),'.jpg']));
Imzero = Im{i}+Imzero;
end
Imback = Imzero/5;
[MR,MC,Dim] = size(Imback);
% Initial State covariance %
d = 7;
delta             = 0.7;
sigmax1           = 60;      % pixel %
sigmavx1          = 1;      % pixel / frame %
sigmay1           = 60;      % pixel  %
sigmavy1          = 1;      % pixel / frame %
sigmaHx1          = 4;    % pixel %
sigmaHy1          = 4;    % pixel %
sigmatheta1       = 8*(pi/180); % rad/frame %
% State Covariance %
% a) Position covariance %
sigmay            = 0.35;
% b) ellipse covariance %
sigmaHx           = 0.1;                % pixel %
sigmaHy           = 0.1;                % pixel %
sigmatheta        = 3.0*(pi/180);       % rad/frame %
%%%%%% State Covariance %%%%%
R=[[0.2845,0.0045]',[0.0045,0.0455]']; %[[0.01,0.001]',[0.01,0.001]']
Rk                = zeros(d , d);
Ry                = sigmay*[delta^3/3 delta^2/2 0 0 ; delta^2/2 delta 0
0 ; 0 0 delta^3/3 delta^2/2 ; 0 0 delta^2/2 delta];
Re                = [sigmaHx.^2 0 0 ; 0 sigmaHy.^2 0 ; 0 0
sigmatheta.^2];
Rk(1 : 4  , 1 : 4)= Ry;
Rk(5 : d , 5 : d) = Re;
Ck                = chol(Rk)';
pos_index         = [1 , 3];
ellipse_index     = [5 , 6 , 7];
H=[[1,0]',[0,0]',[0,1]',[0,0]',[0,0]',[0,0]',[0,0]']
Q=0.01*eye(7);
P = 100*eye(7);
dt=1;
A=[[1,0,0,0,0,0,0]',[dt,1,0,0,0,0,0]',[0,0,1,0,0,0,0]',[0,0,dt,1,0,0,0]
',[0,0,0,0,1,0,0]',[0,0,0,0,0,1,0]',[0,0,0,0,0,0,1]'];
Bu = [0,0,0,6,0,0,0]';
```

```
kfinit=0;
x=zeros(100,7);
N = randn(100,7);
Smean = zeros(1,7);
S = zeros(60,7);
yk = zeros(7,60);
distance = zeros(2,1);
error = zeros(1,60);
aaa = 0;
% loop over all images
for i = 1 : 60
  % load image
  Im = (imread(['DATA/',int2str(i), '.jpg']));
  imshow(Im)
  imshow(Im)
  Imwork = double(Im);
  %extract ball
  [cc(i),cr(i),radius,flag] = extractball(Imwork,Imback,i);
  if flag==0
    continue
  end

  % Kalman update
  if kfinit==0
    xp = [MC/2,0,MR/2,0,14,20,pi]';
  else
    xp=A*x(i-1,:)' + Ck*N(i-1,:)'; %Bu
  end
  kfinit=1;
  PP = A*P*A' + Q;
  K = PP*H'*inv(H*PP*H'+R);
  yk(:,i) = K*([cc(i),cr(i)]' - H*xp); %measurement residual
  x(i,:) = (xp + yk(:,i))';%[cc(i),0,cr(i),0,0,0,0]'
  distance = H*x(i,:)' - [cc(i),cr(i)]';
  error(1,i) = sqrt(distance(1,1)^2 + distance(2,1)^2);
  P = (eye(7)-K*H)*PP;
  Smean = x(i,:);
  S(i,:) = Smean;
  hold on
  ind_i             = (1 : i);
    ykmean            = Smean(pos_index);
    ekmean            = Smean(ellipse_index);
    [xmean , ymean]   = ellipse(ykmean' , ekmean');
    plot(xmean , ymean , 'g' , 'linewidth' , 3)
    plot(S(:,pos_index(1)) , S(:,pos_index(2)) , 'r' , 'linewidth' , 3)
    plot(xp(pos_index(1) , :) , xp(pos_index(2) , :) , 'b+');
    hold off
    pause(0.3)
    aaa =  aaa + 1
end
```

Kalman Filter Algorithm (Video 2)

```
clear,clc
% compute the background image
% Kalman filter initialization
Imzero = zeros(576,768,3);
for i = 1:18
Im{i} = double(imread(['DATA2/',int2str(i),'.jpg']));
Imzero = Im{i}+Imzero;
end
Imback = Imzero/18;
[MR,MC,Dim] = size(Imback);
% Initial State covariance %
d = 7;
delta             = 0.7;
sigmax1           = 60;      % pixel %
sigmavx1          = 1;      % pixel / frame %
sigmay1           = 60;      % pixel  %
sigmavy1          = 1;      % pixel / frame %
sigmaHx1          = 4;    % pixel %
sigmaHy1          = 4;    % pixel %
sigmatheta1       = 8*(pi/180); % rad/frame %
% State Covariance %
% a) Position covariance %
sigmay            = 0.35;
% b) ellipse covariance %
sigmaHx           = 0.1;                % pixel %
sigmaHy           = 0.1;                % pixel %
sigmatheta        = 3.0*(pi/180);       % rad/frame %
%%%%%% State Covariance %%%%%
R=[[0.2845,0.0045]',[0.0045,0.0455]']; %[[0.01,0.001]',[0.01,0.001]']
Rk                = zeros(d , d);
Ry                = sigmay*[delta^3/3 delta^2/2 0 0 ; delta^2/2 delta 0
0 ; 0 0 delta^3/3 delta^2/2 ; 0 0 delta^2/2 delta];
Re                = [sigmaHx.^2 0 0 ; 0 sigmaHy.^2 0 ; 0 0
sigmatheta.^2];
Rk(1 : 4  , 1 : 4)= Ry;
Rk(5 : d , 5 : d) = Re;
Ck                = chol(Rk)';
pos_index         = [1 , 3];
ellipse_index     = [5 , 6 , 7];
H=[[1,0]',[0,0]',[0,1]',[0,0]',[0,0]',[0,0]',[0,0]']
Q=0.01*eye(7);
P = 100*eye(7);
dt=1;
A=[[1,0,0,0,0,0,0]',[dt,1,0,0,0,0,0]',[0,0,1,0,0,0,0]',[0,0,dt,1,0,0,0]
',[0,0,0,0,1,0,0]',[0,0,0,0,0,1,0]',[0,0,0,0,0,0,1]'];
Bu = [0,0,0,6,0,0,0]';
kfinit=0;
```

```
x=zeros(100,7);
N = randn(100,7);
Smean = zeros(1,7);
S = zeros(60,7);
yk = zeros(7,60);
distance = zeros(2,1);
error = zeros(1,60);aaa = 0;
% loop over all images
for i = 1 : 102
  % load image
  Im = (imread(['DATA2/',int2str(i), '.jpg']));
  imshow(Im)
  imshow(Im)
  Imwork = double(Im);
  %extract ball
  [cc(i),cr(i),radius,flag] = extractball(Imwork,Imback,i);
  if flag==0
    continue
  end

  % Kalman update
  if kfinit==0
    xp = [MC/2,0,MR/2,0,30,30,pi]';
  else
    xp=A*x(i-1,:)' + Ck*N(i-1,:)'; %Bu
  end
  kfinit=1;
  PP = A*P*A' + Q;
  K = PP*H'*inv(H*PP*H'+R);
  yk(:,i) = K*([cc(i),cr(i)]' - H*xp); %measurement residual
  x(i,:) = (xp + yk(:,i))';%[cc(i),0,cr(i),0,0,0,0]'
  distance = H*x(i,:)' - [cc(i),cr(i)]';
  error(1,i) = sqrt(distance(1,1)^2 + distance(2,1)^2);
  P = (eye(7)-K*H)*PP;
  Smean = x(i,:);
  S(i,:) = Smean;
  hold on
  ind_i             = (1 : i);
    ykmean            = Smean(pos_index);
    ekmean            = Smean(ellipse_index);
    [xmean , ymean]   = ellipse(ykmean' , ekmean');
    plot(xmean , ymean , 'g' , 'linewidth' , 3)
    plot(S(:,pos_index(1)) , S(:,pos_index(2)) , 'r' , 'linewidth' , 3)
    plot(xp(pos_index(1) , :) , xp(pos_index(2) , :) , 'b+');
    hold off
    pause(0.3)
    aaa =  aaa + 1
end
```

Kalman Filter Algorithm (Video 3)

```
clear,clc
% compute the background image
% Kalman filter initialization
             = 'camera2.avi';
info              = aviinfo()
%offset_frame      = 80;%80
Imzero = zeros(576,768,3);
for i = 320:339
    mov               = aviread( , i);
    Im{i}               = double(mov.cdata);
    Imzero = Im{i}+Imzero;
end
Imback = Imzero/20;
[MR,MC,Dim] = size(Imback);
% Initial State covariance %
d = 7;
delta             = 0.7;
sigmax1           = 60;      % pixel %
sigmavx1          = 1;      % pixel / frame %
sigmay1           = 60;      % pixel  %
sigmavy1          = 1;      % pixel / frame %
sigmaHx1          = 4;    % pixel %
sigmaHy1          = 4;    % pixel %
sigmatheta1       = 8*(pi/180); % rad/frame %
% State Covariance %
% a) Position covariance %
sigmay            = 0.35;
% b) ellipse covariance %
sigmaHx           = 0.1;                % pixel %
sigmaHy           = 0.1;                % pixel %
sigmatheta        = 3.0*(pi/180);       % rad/frame %
%%%%%% State Covariance %%%%%
R=[[0.2845,0.0045]',[0.0045,0.0455]']; %[[0.01,0.001]',[0.01,0.001]']
Rk                = zeros(d , d);
Ry                = sigmay*[delta^3/3 delta^2/2 0 0 ; delta^2/2 delta 0
0 ; 0 0 delta^3/3 delta^2/2 ; 0 0 delta^2/2 delta];
Re                = [sigmaHx.^2 0 0 ; 0 sigmaHy.^2 0 ; 0 0
sigmatheta.^2];
Rk(1 : 4  , 1 : 4)= Ry;
Rk(5 : d , 5 : d) = Re;
Ck                = chol(Rk)';
pos_index         = [1 , 3];
ellipse_index     = [5 , 6 , 7];
H=[[1,0]',[0,0]',[0,1]',[0,0]',[0,0]',[0,0]',[0,0]']
Q=0.01*eye(7);
P = 100*eye(7);
dt=1;
```

```
A=[[1,0,0,0,0,0,0]',[dt,1,0,0,0,0,0]',[0,0,1,0,0,0,0]',[0,0,dt,1,0,0,0]
',[0,0,0,0,1,0,0]',[0,0,0,0,0,1,0]',[0,0,0,0,0,0,1]'];
Bu = [0,0,0,6,0,0,0]';
kfinit=0;
x=zeros(400,7);
N = randn(400,7);
Smean = zeros(1,7);
S = zeros(100,7);
yk = zeros(7,100);
distance = zeros(2,1);
error = zeros(1,100);
% loop over all images
aaa = 0;
xp = [300,0,350,0,100,160,pi]' %300 ; 350
for i = 340 : 400
  % load image
  mov              = aviread( , i);
  Im              = double(mov.cdata);
  title(sprintf('Frames = %d/%d' , i , 390 ));
  image(mov.cdata);
  Imwork = double(Im);
%   sub = Imwork - Imback;
%   imshow(sub)
  %extract ball
  [cc(i),cr(i),radius,flag] = extractball(Imwork,Imback,i);
  if flag==0
    continue
  end
  % Kalman update
  if kfinit==0
    xp = [300,0,350,0,100,160,pi]'; %480,640
  else
    xp=A*x(i-1,:)' + Ck*N(i-1,:)'; %Bu
  end
  kfinit=1;
  PP = A*P*A' + Q;
  K = PP*H'*inv(H*PP*H'+R);
  yk(:,i) = K*([cc(i),cr(i)]' - H*xp); %measurement residual
  x(i,:) = (xp + yk(:,i))';%[cc(i),0,cr(i),0,0,0,0]'
  distance = H*x(i,:)' - [cc(i),cr(i)]';
  error(1,i) = sqrt(distance(1,1)^2 + distance(2,1)^2);
  P = (eye(7)-K*H)*PP;
  Smean = x(i,:);
  S(i,:) = Smean;
  hold on
  ind_i             = (1 : i);
    ykmean            = Smean(pos_index);
    ekmean            = Smean(ellipse_index);
    [xmean , ymean]   = ellipse(ykmean' , ekmean');
    plot(xmean , ymean , 'g' , 'linewidth' , 3)
    plot(S(:,pos_index(1)) , S(:,pos_index(2)) , 'r' , 'linewidth' , 3)
    plot(xp(pos_index(1) , :) , xp(pos_index(2) , :) , 'b+');
    hold off
    pause(0.3)
```

```
    aaa =  aaa + 1
end
```

Particle Filter Algorithm (Video 1)

```
clear all
close all
Image = imread(['DATA/',int2str(17),'.jpg']);
[MR,MC,Dim] = size(Image);
nb_frame          = 120;
dim_x             = MC;
dim_y             = MR;
N                 = 1000;        % Number of particules
N_threshold       = 6.*N/10;    % Redistribution threshold
delta             = 0.8;
%%%%% Color Cue parameters %%%%%%%%
Npdf              = 800;       % Number of samples to draw inside
ellipse to evaluate color histogram
Nx                = 6;         % Number of bins in first color
dimension (R or H)
Ny                = 6;         % Number of bins in second color
dimension (G or S)
Nz                = 6;         % Number of bins in third color
dimension (B or V)
sigma_color       = 0.20;      % Measurement Color noise
nb_hist           = 256;
range             = 1;
pos_index         = [1 , 3];
ellipse_index     = [5 , 6 , 7];
d                 = 7;
M                 = Nx*Ny*Nz;
vect_col          = (0:range/(nb_hist - 1):range);
%%%%%% Target Localization for computing the target distribution %%%%
yq                = [133.87 ; 21.79];
eq                = [16 ; 16 ; pi];
%%%%%% Initialization distribution initialization %%%%
Sk                = zeros(d , 1);
Sk(pos_index)     = yq;
Sk(ellipse_index) = eq;
% Initial State covariance %
sigmax1           = 60;      % pixel %
sigmavx1          = 2;      % pixel / frame %
sigmay1           = 60;      % pixel  %
sigmavy1          = 2;      % pixel / frame %
sigmaHx1          = 2;    % pixel %
sigmaHy1          = 2;    % pixel %
sigmatheta1       = 8*(pi/180); % rad/frame %
% State Covariance %
% a) Position covariance %
sigmay            = 0.35;
% b) ellipse covariance %
sigmaHx           = 0.1;                % pixel %
```

```
sigmaHy           = 0.1;                % pixel %
sigmatheta        = 3.0*(pi/180);       % rad/frame %
%%%%%%%%%%%%%%%%%%%% State transition matrix %%%%%%%%%%%%%%%%%%%%
A                 = [1 delta 0 0 0 0 ; 0 1 0 0 0 0 ; 0 0 1 delta 0
0 0; 0 0 0 1 0 0 ; 0 0 0 0 1 0 0 ; 0 0 0 0 0 1 0 ; 0 0 0 0 0 0 1];
By                = [1 0 0 0 0 0 ; 0 0 1 0 0 0 0];
Be                = [0 0 0 0 1 0 0 ; 0 0 0 0 0 1 1 ; 0 0 0 0 0 0 1 ];
%%%%%% Initial State Covariance %%%%%
R1                = diag([sigmax1 , sigmavx1 , sigmay1 , sigmavy1 ,
sigmaHx1 , sigmaHy1 , sigmatheta1].^2);
%%%%%% State Covariance %%%%%
Rk                = zeros(d , d);
Ry                = sigmay*[delta^3/3 delta^2/2 0 0 ; delta^2/2 delta 0
0 ; 0 0 delta^3/3 delta^2/2 ; 0 0 delta^2/2 delta];
Re                = [sigmaHx.^2 0 0 ; 0 sigmaHy.^2 0 ; 0 0
sigmatheta.^2];
Rk(1 : 4  , 1 : 4)= Ry;
Rk(5 : d , 5 : d) = Re;
Ck                = chol(Rk)';
%%%%%%%% Memory Allocation %%%%%%%
ON                = ones(1 , N);
Od                = ones(d , 1);
Smean             = zeros(d , nb_frame);
Pcov              = zeros(d , d , nb_frame);
N_eff             = zeros(1 , nb_frame);
cte               = 1/N;
cteN              = cte(1 , ON);
w                 = cteN;
compteur          = 0;
cte1_color        = 1/(2*sigma_color*sigma_color);
cte2_color        = (1/(sqrt(2*pi)*sigma_color));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Target Distribution %%%%%%%%%%%%%%
Image = imread(['DATA/',int2str(17),'.jpg']);
Z                 = double(Image);
im                = rgb2hsv_mex(Z);
C1                = cumsum(histc(reshape(im(: , : , 1) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
C2                = cumsum(histc(reshape(im(: , : , 2) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
C3                = cumsum(histc(reshape(im(: , : , 3) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
i1                = sum(C1(: , ones(1 , Nx)) < repmat((0:1/(Nx - 1) :
1) , nb_hist  , 1));
i2                = sum(C2(: , ones(1 , Ny)) < repmat((0:1/(Ny - 1) :
1) , nb_hist  , 1));
i3                = sum(C3(: , ones(1 , Nz)) < repmat((0:1/(Nz - 1) :
1) , nb_hist  , 1));
edge1             = [0 , vect_col(i1(2 : end)) , range];
edge2             = [0 , vect_col(i2(2 : end)) , range];
edge3             = [0 , vect_col(i3(2 : end)) , range];
q                 = pdfcolor_ellipserand(im , yq , eq , Npdf , edge1 ,
edge2 , edge3);
Q                 = q(: , ON);
fig1              = figure(1);
```

```
imshow (Image);
aaa = 0;
%%%%%%%%%%%% Particle initialisation %%%%%%%%
Sk                = Sk(: , ON) + chol(R1)'*randn(d , N);
%%%%%%%%%%%% Main Loop %%%%%%%%%%%%%%%%%%%%%%%
for k = 20: nb_frame ;
    disp(sprintf('Frames = %d/%d' , k , nb_frame ));
    Sk                = A*Sk + Ck*randn(d , N);
    Image = imread(['DATA/',int2str(k),'.jpg']);
    %imshow(Image)
    Z                 = double(Image);
    im                = rgb2hsv_mex(Z);
    yk                = Sk(pos_index , :);     %yk                 =
By*Sk;
    ek                = Sk(ellipse_index , :); %ek                 =
Be*Sk;
    %%%%%%%%%%  Color Likelihood %%%%%%%%%%
    [py , zi , yi]    = pdfcolor_ellipserand(im , yk , ek , Npdf , edge1
, edge2 , edge3);
    rho_py_q          = sum(sqrt(py.*Q));
    likelihood_color = cte2_color*exp((rho_py_q - 1)*cte1_color);
    w                 = w.*likelihood_color;
    w                 = w/sum(w);
    %--------------------------- 6) MMSE estimate & covariance --------
-----------------
    [Smean(: , k) , Pcov(: , : , k)] = part_moment(Sk , w);
    %--------------------------- 7) Particles redistribution ? if N_eff
< N_threshold -------------------------
    N_eff(k)                          = 1./sum(w.*w);
    if (N_eff(k) < N_threshold)
        compteur              = compteur + 1;
        indice_resampling     = particle_resampling(w);
        % Recopie des particules selon le tirage des indices précédents
        Sk                    = Sk(: , indice_resampling);
        w                     = cteN;
    end
    %%%%%%%%%%%%% Display %%%%%%%%%%%%%%%
    fig1              = figure(1);
    imshow(Image)
    title(sprintf('N = %6.3f/%6.3f, Frame = %d, Redistribution =%d' ,
N_eff(k) , N_threshold , k , compteur))
    ind_k             = (1 : k);
    hold on
    ykmean            = Smean(pos_index , k);
    ekmean            = Smean(ellipse_index, k);
    [xmean , ymean]   = ellipse(ykmean , ekmean);
    plot(xmean , ymean , 'g' , 'linewidth' , 3)
    plot(Smean(pos_index(1) , ind_k) , Smean(pos_index(2) , ind_k) ,
'r' , 'linewidth' , 2)
    plot(Sk(pos_index(1) , :) , Sk(pos_index(2) , :) , 'b+');
    hold off
    aaa =  aaa + 1
end
```

Particle Filter Algorithm (Video 2)

```
clear all
close all
Image = imread(['DATA2/',int2str(40),'.jpg']);
[MR,MC,Dim] = size(Image);
nb_frame          = 102;
dim_x             = MC;
dim_y             = MR;
N                 = 1500;         % Number of particules
N_threshold       = 6.*N/10;    % Redistribution threshold
delta             = 0.8;
%%%%% Color Cue parameters %%%%%%%%
Npdf              = 1000;       % Number of samples to draw inside
ellipse to evaluate color histogram
Nx                = 6;          % Number of bins in first color
dimension (R or H)
Ny                = 6;          % Number of bins in second color
dimension (G or S)
Nz                = 6;          % Number of bins in third color
dimension (B or V)
sigma_color       = 0.20;       % Measurement Color noise
nb_hist           = 256;
range             = 1;
pos_index         = [1 , 3];
ellipse_index     = [5 , 6 , 7];
d                 = 7;
M                 = Nx*Ny*Nz;
vect_col          = (0:range/(nb_hist - 1):range);
%%%%%% Target Localization for computing the target distribution %%%%
yq                = [520.1592 ; 357.4856];
eq                = [100 ; 50 ; -7*pi/8];
%%%%%% Initialization distribution initialization %%%%
Sk                = zeros(d , 1);
Sk(pos_index)     = yq;
Sk(ellipse_index) = eq;
% Initial State covariance %
sigmax1           = 60;       % pixel %
sigmavx1          = 4;       % pixel / frame %
sigmay1           = 60;       % pixel  %
sigmavy1          = 4;       % pixel / frame %
sigmaHx1          = 7;     % pixel %
sigmaHy1          = 7;     % pixel %
sigmatheta1       = 8*(pi/180); % rad/frame %
% State Covariance %
% a) Position covariance %
sigmay            = 0.35;
% b) ellipse covariance %
sigmaHx           = 2;                % pixel %
sigmaHy           = 0.6;                % pixel %
sigmatheta        = 3.0*(pi/180);       % rad/frame %
%%%%%%%%%%%%%%%%%%%% State transition matrix %%%%%%%%%%%%%%%%%%%%%
```

```
A                = [1 delta 0 0 0 0 0 ; 0 1 0 0 0 0 0 ; 0 0 1 delta 0
0 0; 0 0 0 1 0 0 0 ; 0 0 0 0 1 0 0 ; 0 0 0 0 0 1 0 ; 0 0 0 0 0 0 1];
By               = [1 0 0 0 0 0 0 ; 0 0 1 0 0 0 0];
Be               = [0 0 0 0 1 0 0 ; 0 0 0 0 0 1 1 ; 0 0 0 0 0 0 1 ];
%%%%%% Initial State Covariance %%%%%
R1               = diag([sigmax1 , sigmavx1 , sigmay1 , sigmavy1 ,
sigmaHx1 , sigmaHy1 , sigmatheta1].^2);
%%%%%% State Covariance %%%%%
Rk               = zeros(d , d);
Ry               = sigmay*[delta^3/3 delta^2/2 0 0 ; delta^2/2 delta 0
0 ; 0 0 delta^3/3 delta^2/2 ; 0 0 delta^2/2 delta];
Re               = [sigmaHx.^2 0 0 ; 0 sigmaHy.^2 0 ; 0 0
sigmatheta.^2];
Rk(1 : 4  , 1 : 4)= Ry;
Rk(5 : d , 5 : d) = Re;
Ck               = chol(Rk)';
%%%%%%%% Memory Allocation %%%%%%%
ON               = ones(1 , N);
Od               = ones(d , 1);
Smean            = zeros(d , nb_frame);
Pcov             = zeros(d , d , nb_frame);
N_eff            = zeros(1 , nb_frame);
cte              = 1/N;
cteN             = cte(1 , ON);
w                = cteN;
compteur         = 0;
cte1_color       = 1/(2*sigma_color*sigma_color);
cte2_color       = (1/(sqrt(2*pi)*sigma_color));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Target Distribution %%%%%%%%%%%%%%%
Image = imread(['DATA2/',int2str(40),'.jpg']);
Z                = double(Image);
im               = rgb2hsv_mex(Z);
C1               = cumsum(histc(reshape(im(: , : , 1) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
C2               = cumsum(histc(reshape(im(: , : , 2) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
C3               = cumsum(histc(reshape(im(: , : , 3) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
i1               = sum(C1(: , ones(1 , Nx)) < repmat((0:1/(Nx - 1) :
1) , nb_hist  , 1));
i2               = sum(C2(: , ones(1 , Ny)) < repmat((0:1/(Ny - 1) :
1) , nb_hist  , 1));
i3               = sum(C3(: , ones(1 , Nz)) < repmat((0:1/(Nz - 1) :
1) , nb_hist  , 1));
edge1            = [0 , vect_col(i1(2 : end)) , range];
edge2            = [0 , vect_col(i2(2 : end)) , range];
edge3            = [0 , vect_col(i3(2 : end)) , range];
q                = pdfcolor_ellipserand(im , yq , eq , Npdf , edge1 ,
edge2 , edge3);
Q                = q(: , ON);
fig1             = figure(1);
imshow (Image);
% image(mov.cdata);
%
```

```
% set(gca , 'drawmode' , 'fast');
% set(gcf , 'doublebuffer','on');
%
% set(gcf , 'renderer' , 'zbuffer');
aaa = 0;
%%%%%%%%%%%% Particle initialisation %%%%%%%%
Sk                 = Sk(: , ON) + chol(R1)'*randn(d , N);
%%%%%%%%%%%% Main Loop %%%%%%%%%%%%%%%%%%%%%%%%
for k = 40: nb_frame ;
    disp(sprintf('Frames = %d/%d' , k , nb_frame ));
    Sk               = A*Sk + Ck*randn(d , N);
    Image = imread(['DATA2/',int2str(k),'.jpg']);
    %imshow(Image)
    Z                = double(Image);
    im               = rgb2hsv_mex(Z);
    yk               = Sk(pos_index , :);     %yk                =
By*Sk;
    ek               = Sk(ellipse_index , :); %ek                =
Be*Sk;
    %%%%%%%%%%  Color Likelihood %%%%%%%%%
    [py , zi , yi]   = pdfcolor_ellipserand(im , yk , ek , Npdf , edge1
, edge2 , edge3);
    rho_py_q         = sum(sqrt(py.*Q));
    likelihood_color = cte2_color*exp((rho_py_q - 1)*cte1_color);
    w                = w.*likelihood_color;
    w                = w/sum(w);
    %--------------------------- 6) MMSE estimate & covariance --------
-----------------
    [Smean(: , k) , Pcov(: , : , k)] = part_moment(Sk , w);
    %--------------------------- 7) Particles redistribution ? if N_eff
< N_threshold -------------------------
    N_eff(k)                         = 1./sum(w.*w);
    if (N_eff(k) < N_threshold)
        compteur             = compteur + 1;
        indice_resampling    = particle_resampling(w);
        % Recopie des particules selon le tirage des indices précédents
        Sk                   = Sk(: , indice_resampling);
        w                    = cteN;
    end
    %%%%%%%%%%%%% Display %%%%%%%%%%%%%%%
    fig1             = figure(1);
    imshow(Image)
    title(sprintf('N = %6.3f/%6.3f, Frame = %d, Redistribution =%d' ,
N_eff(k) , N_threshold , k , compteur))
    ind_k            = (1 : k);
    hold on
    ykmean           = Smean(pos_index , k);
    ekmean           = Smean(ellipse_index, k);
    [xmean , ymean]  = ellipse(ykmean , ekmean);
    plot(xmean , ymean , 'g' , 'linewidth' , 3)
    plot(Smean(pos_index(1) , ind_k) , Smean(pos_index(2) , ind_k) ,
'r' , 'linewidth' , 2)
    plot(Sk(pos_index(1) , :) , Sk(pos_index(2) , :) , 'b+');
    hold off
```

```
    aaa =  aaa + 1
end
```

Particle Filter Algorithm (Video 3)

```
clear all
close all
             = 'camera2.avi';
info              = aviinfo()
offset_frame      = 339;%80
%nb_frame          = info.NumFrames
%nb_frame          = info.NumFrames - offset_frame -10;
nb_frame          = 500;
dim_x             = info.Width;
dim_y             = info.Height;
N                 = 2000;         % Number of particules
N_threshold       = 6.*N/10;    % Redistribution threshold
delta             = 0.7;
%%%%% Color Cue parameters %%%%%%%%
Npdf              = 800;       % Number of samples to draw inside
ellipse to evaluate color histogram
Nx                = 6;         % Number of bins in first color
dimension (R or H)
Ny                = 6;         % Number of bins in second color
dimension (G or S)
Nz                = 6;         % Number of bins in third color
dimension (B or V)
sigma_color       = 0.20;      % Measurement Color noise
nb_hist           = 256;
range             = 1;
pos_index         = [1 , 3];
ellipse_index     = [5 , 6 , 7];
d                 = 7;
M                 = Nx*Ny*Nz;
vect_col          = (0:range/(nb_hist - 1):range);
%%%%%% Target Localization for computing the target distribution %%%%
% yq                = [dim_x ; dim_y/2];
yq                = [300 ; 350];
% eq                = [100 ; 160 ; pi/3];
eq                = [30 ; 30 ; pi/3];
%%%%%% Initialization distribution initialization %%%%
Sk                = zeros(d , 1);
Sk(pos_index)     = yq;
Sk(ellipse_index) = eq;
% Initial State covariance %
sigmax1           = 60;      % pixel %
sigmavx1          = 1;      % pixel / frame %
sigmay1           = 60;      % pixel  %
sigmavy1          = 1;      % pixel / frame %
sigmaHx1          = 40;    % pixel %
sigmaHy1          = 40;    % pixel %
sigmatheta1       = 30*(pi/180); % rad/frame %
% State Covariance %
% a) Position covariance %
```

```
sigmay            = 0.35;
% b) ellipse covariance %
sigmaHx           = 0.1;                % pixel %
sigmaHy           = 0.1;                % pixel %
sigmatheta        = 3.0*(pi/180);       % rad/frame %
%%%%%%%%%%%%%%%%%%%% State transition matrix %%%%%%%%%%%%%%%%%%%%
A                 = [1 delta 0 0 0 0 0 ; 0 1 0 0 0 0 0 ; 0 0 1 delta 0
0 0; 0 0 0 1 0 0 0 ; 0 0 0 0 1 0 0 ; 0 0 0 0 0 1 0 ; 0 0 0 0 0 0 1];
By                = [1 0 0 0 0 0 0 ; 0 0 1 0 0 0 0];
Be                = [0 0 0 0 1 0 0 ; 0 0 0 0 0 1 1 ; 0 0 0 0 0 0 1 ];
%%%%%% Initial State Covariance %%%%%
R1                = diag([sigmax1 , sigmavx1 , sigmay1 , sigmavy1 ,
sigmaHx1 , sigmaHy1 , sigmatheta1].^2);
%%%%%% State Covariance %%%%%
Rk                = zeros(d , d);
Ry                = sigmay*[delta^3/3 delta^2/2 0 0 ; delta^2/2 delta 0
0 ; 0 0 delta^3/3 delta^2/2 ; 0 0 delta^2/2 delta];
Re                = [sigmaHx.^2 0 0 ; 0 sigmaHy.^2 0 ; 0 0
sigmatheta.^2];
Rk(1 : 4  , 1 : 4)= Ry;
Rk(5 : d , 5 : d) = Re;
Ck                = chol(Rk)';
%%%%%%%% Memory Allocation %%%%%%%
ON                = ones(1 , N);
Od                = ones(d , 1);
Smean             = zeros(d , nb_frame);
Pcov              = zeros(d , d , nb_frame);
N_eff             = zeros(1 , nb_frame);
cte               = 1/N;
cteN              = cte(1 , ON);
w                 = cteN;
compteur          = 0;
cte1_color        = 1/(2*sigma_color*sigma_color);
cte2_color        = (1/(sqrt(2*pi)*sigma_color));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Target Distribution %%%%%%%%%%%%%%
mov               = aviread( , offset_frame );
im                = mov.cdata;
Z                 = double(im);
im                = rgb2hsv_mex(Z);
C1                = cumsum(histc(reshape(im(: , : , 1) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
C2                = cumsum(histc(reshape(im(: , : , 2) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
C3                = cumsum(histc(reshape(im(: , : , 3) , dim_x*dim_y ,
1) , vect_col))/(dim_x*dim_y);
i1                = sum(C1(: , ones(1 , Nx)) < repmat((0:1/(Nx - 1) :
1) , nb_hist  , 1));
i2                = sum(C2(: , ones(1 , Ny)) < repmat((0:1/(Ny - 1) :
1) , nb_hist  , 1));
i3                = sum(C3(: , ones(1 , Nz)) < repmat((0:1/(Nz - 1) :
1) , nb_hist  , 1));
edge1             = [0 , vect_col(i1(3 : end)) , range];
edge2             = [0 , vect_col(i2(3 : end)) , range];
edge3             = [0 , vect_col(i3(2 : end)) , range];
```

```
q                = pdfcolor_ellipserand(im , yq , eq , Npdf , edge1 ,
edge2 , edge3);
Q                = q(: , ON);
fig1             = figure(1);
image(mov.cdata);
set(gca , 'drawmode' , 'fast');
set(gcf , 'doublebuffer','on');
set(gcf , 'renderer' , 'zbuffer');
aaa = 0;
%%%%%%%%%%%% Particle initialisation %%%%%%%%
Sk               = Sk(: , ON) + chol(R1)'*randn(d , N);
%%%%%%%%%%%% Main Loop %%%%%%%%%%%%%%%%%%%%%%%
for k = 340 : nb_frame ;
    disp(sprintf('Frames = %d/%d' , k , nb_frame ));
    Sk               = A*Sk + Ck*randn(d , N);
    mov              = aviread( , k);
    im               = (mov.cdata);
    Z                = double(im);
    im               = rgb2hsv_mex(Z);
    yk               = Sk(pos_index , :);     %yk                =
By*Sk;
    ek               = Sk(ellipse_index , :); %ek                =
Be*Sk;
    %%%%%%%%%%  Color Likelihood %%%%%%%%%%
    [py , zi , yi]   = pdfcolor_ellipserand(im , yk , ek , Npdf , edge1
, edge2 , edge3);
    rho_py_q         = sum(sqrt(py.*Q));
    likelihood_color = cte2_color*exp((rho_py_q - 1)*cte1_color);
    w                = w.*likelihood_color;
    w                = w/sum(w);
    %---------------------------- 6) MMSE estimate & covariance --------
-----------------
    [Smean(: , k) , Pcov(: , : , k)] = part_moment(Sk , w);
    %--------------------------- 7) Particles redistribution ? if N_eff
< N_threshold ------------------------
    N_eff(k)                         = 1./sum(w.*w);
    if (N_eff(k) < N_threshold)
        compteur             = compteur + 1;
        indice_resampling    = particle_resampling(w);
        % Recopie des particules selon le tirage des indices précédents
        Sk                   = Sk(: , indice_resampling);
        w                    = cteN;
    end
    %%%%%%%%%%%%% Display %%%%%%%%%%%%%%%
    fig1             = figure(1);
    image(mov.cdata);
    title(sprintf('N = %6.3f/%6.3f, Frame = %d, Redistribution =%d' ,
N_eff(k) , N_threshold , k , compteur))
    ind_k            = (1 : k);
    hold on
    ykmean           = Smean(pos_index , k);
    ekmean           = Smean(ellipse_index, k);
    [xmean , ymean]  = ellipse(ykmean , ekmean);
    plot(xmean , ymean , 'g' , 'linewidth' , 3)
```

```
    plot(Smean(pos_index(1) , ind_k) , Smean(pos_index(2) , ind_k) ,
'r' , 'linewidth' , 2)
    plot(Sk(pos_index(1) , :) , Sk(pos_index(2) , :) , 'b+');
    hold off
     aaa =  aaa + 1
end
```

REFERENCES

[1] Bonnie Danette Allen, Gary Bishop, and Greg Welch. *Tracking: Beyond 15 Minutes of Thought*, SIGGRAPH 2001 course 11. In Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques. ACM Press, Addison-Wesley, Los Angeles, CA, USA (August 12–17), SIGGRAPH 2001 course pack edition, 2001.

[2] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*, SIGGRAPH 2001 course 8. In Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques. ACM Press, Addison-Wesley, Los Angeles, CA, USA (August 12--17), SIGGRAPH 2001 course pack edition, 2001.

[3] Yilmaz, A., Javed, O., and Shah, M. 2006. *Object tracking: A survey*. ACM Comput. Surv. 38, 4, Article 13 (Dec. 2006), 45 pages. DOI = 10.1145/1177352.1177355

[4] Yao Shen, Parthasarathy Guturu, Thyagaraju Damarla, and Bill P. Buckles. *Particle Filter Based Object Tracking with discriminative Feature Extraction and Fusion*, Lecture Notes In Computer Science; Vol. 5359 Proceedings of the 4th International Symposium on Advances in Visual Computing, Part II Las Vegas, NV Section: Motion Pages: 246 - 256 Year of Publication: 2008

[5] Katja Nummiaro, Esther K Meier, Luc J Van Gool, *Object Tracking with an Adaptive Color – Based Particle Filter,* In Proceedings of the 24th DAGM Symposium on Pattern Recognition (2002), pp. 353-360.

[6] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp: *A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking*. IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 50, NO. 2, FEBRUARY 2002

[7] Jaco Vermaak, Patrick P´erez, Michel Gangnet, and Andrew Blake, *Towards Improved Observation Models for Visual Tracking: Selective Adaptation*, Lecture Notes In Computer Science; Vol. 2350. Proceedings of the 7th European Conference on Computer Vision-Part I Pages: 645 - 660
Year of Publication: 2002

[8] Emilio Maggio, Student Member, IEEE, Fabrizio Smerladi, and Andrea Cavallaro, Member, IEEE. *Adaptive Multifeature Tracking in a Particle Filtering Framework*: IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR TECHNOLOGY, VOL. 17, NO. 10, OCTOBER 2007:

[9] Changjiang Yang, Ramani Duraiswami and Larry Davis. *Fast Multiple Object Tracking via a Hierarchical Particle Filter*, This paper appears in: Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on. Publication Date: 17-21 Oct. 2005 Volume: 1, On page(s): 212- 219 Vol. 1

[10] Z. Chen, *Bayesian filtering: From kalman filters to particle filters, and beyond*, McMaster University, Tech. Rep., 2003. [Online]. Available: http://soma.crl.mcmaster.ca/~zhechen/ieee_bayes.ps

[11] Dorin Comaniciu, Visvanathan Ramesh, Peter Meer, *Kernel-Based Object Tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence Volume 25 , Issue 5 (May 2003) Pages: 564 – 575, Year of Publication: 2003

[12]f. Aherne, N. Thacker and P. Rockett, *The Bhattacharyya Metric as an Absolute Similarity Measure for Frequency Coded Data*, Kybernetika, pp. 1-7, Vol. 32(4), 1997

[13] M. Tuceryan and A. K. Jain, *Texture Analysis*, In The Handbook of Pattern Recognition and Computer Vision (2nd Edition), by C. H. Chen, L. F. Pau, P. S. P. Wang (eds.), pp. 207-248, World Scientific Publishing Co., 1998. (Abstract) (Book Chapter)

[14] Julier, S.J.; Uhlmann, J.K. (1997), *A new extension of the Kalman filter to nonlinear systems*, Int. Symp. Aerospace/Defense Sensing, Simul. and Controls 3. http://www.cs.unc.edu/~welch/kalman/media/pdf/Julier1997_SPIE_KF.pdf Retrieved on 2008-05-03.

[15]Peter S.MAYBECK, Stochastic models, estimation, and control vol 1 Department of electrical engineering.Air Force Institute of Technology. Wright-Patterson Air Force Base Ohio

[16] Mallick, M. Maskell, S. Kirubarajan, T. Gordon, N. , *Littoral tracking using particle filter*, Information Fusion, 2002. Proceedings of the Fifth International Conference on Publication Date: 2002 Volume: 2, On page(s): 935- 942 vol.2