Kalman Filter and Particle Filter for Real time Object Tracking

Lucas Rodés Guirao (partner *Diego Morín*)
lucasrg@kth.se

***Abstract*—The objective of this project is to compare the performance of the Kalman Filter and the Particle Filter for real time object tracking. In addition, we explore possible combinations of both techniques. Furthermore, we will use a pre-filtered video sample as input. With this project, we expect to gain a deep understanding of both approaches. Advantages and drawbacks from both techniques will be also highlighted. Finally, a demo of the implementation will be provided.**

I. Introduction

We are living in an era in which massive amounts of data are generated everyday. Currently, 2.5 quintillion bytes of data are created daily, according to [1]. Within this sea of data, there is place for visual content (such as images, videos), where powerful algorithms and processing techniques can provide the tools to make inferences from this data. In this regard, *computer vision* is an interdisciplinary field which focuses on the understanding of computers of digital images. Speaking in terms of engineering, computer vision aims to design systems that automatize tasks that could be easily done by humans but are infeasible due to the yet increasing amount of data. Within this field, we find *object tracking*, which has gained momentum due to the progress done in the field of supercomputing hardware and the increasing affordability of HD cameras. Object tracking has a wide range of applications, such as surveillance, sports, medical imaging etc [2].

In this regard, this report analyses and discusses different object tracking approaches in the context of video analysis. In particular, we study two different methods, namely (i) the *Kalman Filter* (KF) and (ii) the *Particle Filter* (PF), which have been exhaustively used in several Object Tracking problems, for instance in [3], [4], [5] and [6], [7], [8], respectively. Object tracking might be complex due to several factors, such as noise in the image frames, scene illumination changes, complex image motion, object occlusions, real-time image processing etc. This paper addresses the three later. Additionally, we also inspect some object detection techniques involving image processing, which are essential for a successful object tracking.

A. Contribution

In this paper, we provide the following contributions:

- We explore a combination of KF and PF, which will be referred as *Kalmanized Particle Filter* (KPF) throughout this paper. The initial idea was to obtain a more robust method, however it presents some drawbacks.
- We compare all implemented methods, i.e. KF, PF and KPF, in terms of the Mean Square Error (MSE) between the estimated- and the real object position.
- We propose *Kernel Density Extraction* (KDE) as an efficient method to solve multiple problems in Object tracking such as: Tuning of noise covariance matrix Q in KF and state estimation and particle weighting in PF.

B. Outline

Section II briefly reviews important aspects of the theory used in this project (i.e. KF and PF). Next, Section III explains all different implementations. Furthermore, IV comments the results obtained from the different approaches. Finally, Section V finishes the report with the conclusions.

II. Object Tracking

Object tracking in videos is the task of locating a specific object in a sequence of frames. First, a detection algorithm is required in order to obtain possible object locations. Next, the object tracker is then responsible for following the trajectory of the object in subsequent frames. There exist several approaches for object tracking, which can be classified in different categories [9]:

- **Point tracking**: It represents the object by a point and estimates the current object location based on estimations in previous frames. That is, there is a dependence between estimations at time k and time $k+1$. Moreover, these estimations usually are the pixel coordinates of the object position and its motion (e.g. velocity, acceleration). This information is stored in a vector which is referred to as *state* vector.
- **Kernel tracking**: It represents the object by a certain shape (e.g. rectangular or elliptic) together with a RGB color histogram [10].
- **Silhouette tracking**: In this approach, the object region is estimating in each frame using image processing methods.

In this project we only consider the point tracking category, which can be further divided into two subcategories, (i) deterministic approaches and (ii) probabilistic approaches. We only focus on the later, which uses randomness techniques to estimate state vectors. In particular, we briefly present the KF and the PF.

Advisor: Prof. John Folkesson, EL2320 Applied Estimation, KTH Royal Institute, HT 2016.

A. Kalman Filter

The KF implements a Bayes filter and was invented by Rudolph Emil Kalman in the 1950s [11]. In this approach, the belief at time step k, i.e. $bel(x_k)$, is assumed to be Gaussian distributed, which allows it to be uniquely described by a mean vector μ_k and a covariance matrix Σ_k. It consists of two steps, namely *prediction* and *update*, each of them making important assumptions.

1) Prediction: In the prediction step, the KF uses previous state to make a first prediction of the current state. In particular, it *assumes* that the state x_k is related with x_{k-1} through a linear function for all $1 \leqslant k \leqslant K$, where K is the duration of the tracking task. This linear function is given by

$$x_k = A_k x_{k-1} + B_k u_k + \varepsilon_k, \tag{1}$$

where we have used the following notation:

x_k: $n \times 1$ vector representing the state vector at time step k. Typically contains data of interest, e.g. position, velocity, acceleration etc.
u_k: $m \times 1$ vector denoting the control vector. Typically contains control input data, e.g. steering angle.
A_k: $n \times n$ matrix state transition matrix which establishes the relation between two consecutive states.
B_k: $n \times m$ control matrix. Relates the control vector and the estimated state.
ε_k: $n \times 1$ Gaussian random vector modelling the randomness in this system. It is assumed to be zero-mean with covariance matrix R_k (often referred to as the *process noise covariance matrix*)

In the context of object tracking, where there is no external influence, both the control matrix and control vector can be omitted. Hence, (1) is rewritten as

$$x_k = A_k x_{k-1} + \varepsilon_k. \tag{2}$$

As aforementioned, the state is fully characterized by a mean vector and a covariance matrix. Thus, the predicted state is defined by the predicted mean $\overline{\mu}_k$ and predicted covariance $\overline{\Sigma}_k$. Using (2) we write the prediction equations as

$$\begin{aligned} \overline{\mu}_k &= A_k \mu_{k-1}, \\ \overline{\Sigma}_k &= A_k \Sigma_{k-1} A_k^T + R_k. \end{aligned} \tag{3}$$

The system performance depends on the value of A_k and R_k, whose impact will be later studied.

2) Update: Next, in the update step the KF uses the current measurement, i.e. z_k, to correct (or update) the object's state. It is *assumed* that the relation between the measurement z_k and the state x_k is also linear. In particular we have that

$$z_k = C_k x_k + \delta_k, \tag{4}$$

where z_k is the $l \times 1$ measurement vector, C_k is the $l \times n$ transformation matrix mapping the vector state into the measurement domain and δ_k is a $l \times 1$ Gaussian vector representing the measurement noise term. Consequently we have that the likelihood of the measurement is $p(z_k|x_k) = N(z_k|C_k x_k, Q_k)$, where Q_k is the *measurement noise covariance matrix*.

In this step, the predicted parameters $\overline{\mu}_k$ and $\overline{\Sigma}_k$ are corrected. For this purpose, the Kalman gain K_k is defined, which quantifies the relative importance between the update and prediction step. That is, the higher it is, the more relevant is the measurement for the state estimation, which might occur when the measurement is very certain. Using (4) we write the update equation as

$$\begin{aligned} K_k &= \overline{\Sigma}_k C_k^T (C_k \overline{\Sigma}_k C_k^T + Q_k)^{-1}, \\ \mu_k &= \overline{\mu}_k + K_k (z_k - C_k \overline{\mu}_k), \\ \Sigma_k &= \overline{\Sigma}_k - K_k C_k \overline{\Sigma}_k. \end{aligned} \tag{5}$$

B. Particle Filter

The PF is a non-parametric implementation of the Bayes Filter. In contrast to the KF, it does not make any assumption on the belief distribution. This, makes it very useful for representing multi-modal posteriors. Moreover, it does not assume the motion model to be linear, making it suitable for modelling non-linear systems.

The main idea of the PF is to represent the posterior $bel(x_k)$ by a finite number of randomly drawn state samples from this posterior. Therefore, we can assert that it is an numerical approximation rather than an analytical. These samples are known as *particles*, which have an associated weighting factor. At time step k we denote a particle by $x_k^{[m]}$ for $m = 1, \ldots, M$, where M is the number of particles. These particles are obtained by sampling from the so called proposal distribution $p(x_k|u_k, x_{k-1})$, for instance (2). Along with each particle, we define the *importance factors* $w_k^{[m]}$. They are used to incorporate the influence from the measurement z_k into the posterior estimation. They are obtained as

$$w_k^{[m]} \propto p(z_k|x_k^{[m]}), \qquad \text{for } m = 1, \ldots, M. \tag{6}$$

Moreover, they are normalized so that $\sum_{m=1}^{M} w_k^{[m]} = 1$. The set of particles

$$\mathcal{X}_k = \{x_k^{[1]}, \ldots, x_k^{[M]}\}$$

weighted by the corresponding importance factors represent the belief at time step k. Nonetheless, after obtaining this set the PF performs its key trick, the *Resampling*. It consists in sampling, with replacement, from the current belief estimate. That is, particles with higher weights will be more likely to "survive". However, re-sampling to often increases the risk of losing diversity.

The larger is M, the more accurate will be the posterior, but this comes at the expense of high computational cost.

III. Implementation

We have selected 1 minute recording of the Nintendo Pinball arcade game as our test video and have focussed on tracking the ball. In addition, we deleted some regions from the original video in order to create occlusions. This allowed us for testing the performance of our implemented methods when the measurements were very poor. Prior to any tracking algorithm, we pre-process each input video frame k in order to generate the corresponding binary matrix $I_B(k)$ of the same size of the original picture with '1's in pixels that are likely to contain ball and '0' in the rest. This was done using an RGB color interval of the ball, giving the results shown in Fig. 1 illustrates.

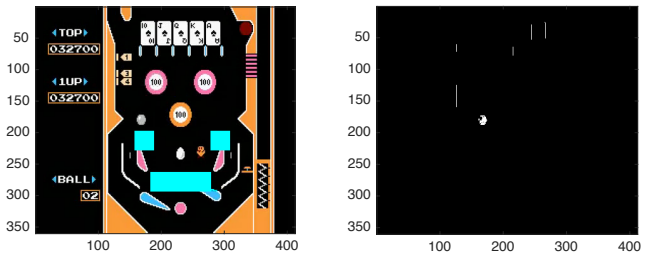


Fig. 1: The left image, shows the input frame with the artificially added occlusions (cyan filled rectangles). The right image illustrates the filtered k:th frame, i.e. $I_B(k)$, which shows likely regions for the ball highlighted in white. This particular result is for frame $k = 315$.

Next, we applied the 10×10 kernel matrix

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 8 & 8 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 8 & 27 & 27 & 8 & 1 & 0 & 0 \\ 0 & 1 & 8 & 27 & 81 & 81 & 27 & 8 & 1 & 0 \\ 1 & 8 & 27 & 81 & 253 & 253 & 81 & 27 & 8 & 1 \\ 1 & 8 & 27 & 81 & 253 & 253 & 81 & 27 & 8 & 1 \\ 0 & 1 & 8 & 27 & 81 & 81 & 27 & 8 & 1 & 0 \\ 0 & 0 & 1 & 8 & 27 & 27 & 8 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 8 & 8 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

to the binary image as

$$I_K(k) = K * I_B(k), \tag{8}$$

where '$*$' stands for the convolution operator and we removed the outer rows and columns of the $I_K(k)$ in order for it to match the dimensions of the original frame. Note that K is a circular kernel (with cubic coefficients) which is what we need when the tracking object is round. Next, $I_K(k)$, which is illustrated in Fig 2, was used to include the measurement information in the posterior estimation on both KF and PF. We shall see this later.

In the following, let us detail our implementations.

A. Kalman Filter

In this project, we have considered two different motion models: (i) *Constant Velocity* and (ii) *Constant Acceleration*.

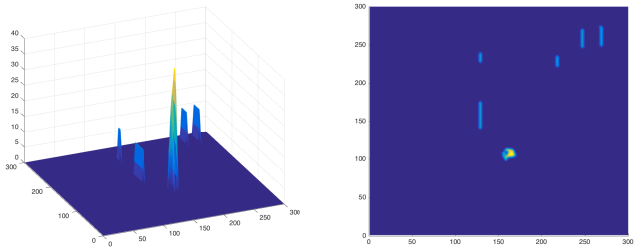


Fig. 2: Result after applying a given Kernel K on the binary image, i.e. we computed as convolution $I_K(k) = K * I_B(k)$, where $I_B(k)$ is the filtered image frame from Fig. 1. This particular output is for frame $k = 315$.

Moreover, for simplicity we have used stationary matrices for the state transition and the state-to-measurement mapping, i.e.

$$\begin{aligned} A_k &= A, \\ C_k &= C. \end{aligned} \tag{9}$$

In this regard, the algorithm of the Kalman Filter is summarized in Fig. 3.

kalman_filter $(\mu_{k-1}, \Sigma_{k-1}, z_k)$
Step 1: Prediction
$\overline{\mu}_k \leftarrow A\mu_{k-1}$
$\overline{\Sigma}_k \leftarrow A\Sigma_{k-1}A^T + R_k$
Step 2: Update
$K_k \leftarrow \overline{\Sigma}_k C^T (C\overline{\Sigma}_k C^T + Q_k)^{-1}$
$\mu_k \leftarrow \overline{\mu}_k + K_k(z_k - C\overline{\mu}_k)$
$\Sigma_k \leftarrow \overline{\Sigma}_k - K_k C\overline{\Sigma}_k$
return μ_k, Σ_k

Fig. 3: Kalman Filter Algorithm

Furthermore, the measurement z_k for the KF was simply acquired by obtaining the coordinates of the mode of $I_K(k)$ for all frames k .

We shall highlight that we keep the measurement noise covariance Q_k variable. We tune it using the variance of the coefficients of $I_K(k)$. In particular, the higher is the variance (which means that we might have a clear mode in $I_K(k)$) the lower valued we set Q_k. In contrast, when the variance is low we set a high valued Q_k. This way, we have a mechanism to put more weight on the motion model whenever the measurements are poor and vice-versa.

1) Constant Velocity: For this approach we used a state vector of size 4×1, shown in (10). The first two positions are the pixel coordinates of the object ($p_{k,1}$ and $p_{k,2}$) and the two last are the discrete velocity of the object (v_{k-1} and $v_{k,2}$),

$$x_k = \begin{bmatrix} p_{k,1} \\ p_{k,2} \\ v_{k,1} \\ v_{k,2} \end{bmatrix}. \tag{10}$$

Note that we only consider two dimensions of motion, since we are working with images which are two-dimensional. Moreover, we have $v_{k,1} = v_1$ and $v_{k,2} = v_2$ for all $k = 1, \ldots, K$.

From physics, for a constant velocity linear model the position p_k can be obtained using the position $p_{k-\Delta k}$, the velocity v and the incremental time difference Δk, namely

$$p_k = p_{k-\Delta k} + \Delta k v.$$

In our case, we have that $\Delta k = 1$, which means that $p_k = p_{k-1} + v$. Thus, the state transition matrix is given by

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{11}$$

Since we only observe the position of the object, the size of the measurement vector z_k is 2×1, i.e.

$$z_k = \begin{bmatrix} z_{k,1} \\ z_{k,2} \end{bmatrix} \tag{12}$$

and the mapping is given by the 2×4-sized matrix

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{13}$$

The initial state, i.e. x_0, is characterized by a mean vector μ_0 and a covariance matrix Σ_0. The first is estimated using the first two measurements z_{-1} and z_0. In particular

$$\mu_0 = \begin{bmatrix} p_{0,1} \\ p_{0,2} \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} z_{0,1} \\ z_{0,2} \\ z_{0,1} - z_{-1,1} \\ z_{0,2} - z_{-1,2} \end{bmatrix}$$

and the second is initialized with high coefficients

$$\Sigma_0 = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}.$$

Using the algorithm from Fig. 3 the object tracking task can be performed.

2) Constant Acceleration: This approach considers variability in the velocity according to a constant acceleration value a. The dynamics are now given by

$$\begin{aligned} p_k &= p_{k-\Delta k} + v_{k-\Delta k}\Delta k + .5a\Delta k^2, \\ v_k &= v_{k-\Delta k} + a\Delta k. \end{aligned} \tag{14}$$

In this regard, the transition matrix is now

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The initial estimate is obtained similarly as in the previous case. However, we now need the first three measurements

$$\mu_0 = \begin{bmatrix} p_{0,1} \\ p_{0,2} \\ v_{0,1} \\ v_{0,2} \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} z_{0,1} \\ z_{0,2} \\ z_{0,1} - z_{-1,1} \\ z_{0,2} - z_{-1,2} \\ (z_{0,1} - z_{-1,1}) - (z_{-1,1} - z_{-2,1}) \\ (z_{0,2} - z_{-1,2}) - (z_{-1,2} - z_{-2,2}) \end{bmatrix}.$$

The initial covariance matrix remains the same, i.e.

$$\Sigma_0 = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 1 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix}.$$

B. Particle Filter

We propose a very simple implementation of the PF. In particular, we only use the process noise to inspect new states. In particular, we use the following motion model

$$x_k^{[m]} = x_{k-1}^{[m]} + \eta^{[m]}, \tag{15}$$

where $\eta^{[m]} \sim N(0, \Sigma_\eta)$ is the diffusion noise for particle m and we have defined $\Sigma_\eta = \text{diag}(25, 25)$.

Next, we assigned the weights to the particles proportionally to the values in $I_K(k)$ corresponding to their location. That is, higher weight was given to those particles that are located in highly likely regions for the ball. This way, we were able to include the information given by the measurements and update the state belief. Formally, we can write this as

$$w_k^{[m]} \propto I_K(k)\Big|_m, \tag{16}$$

where $I_K(k)\Big|_m$ denotes the kernel value of the particle $x_k^{[m]}$ at time step k, respectively. As aforementioned, we need to ensure normalization of the weighting factors. At each iteration k, we can estimate the state computing the weighted sum of the particles coordinates, i.e.

$$x_k^{est} = \sum_{m=1}^{M} w_k^{[m]} x_k^{[m]} \tag{17}$$

As of the re-sampling method, we have used systematic re-sampling, since it only generates one random number (instead of M random numbers, as the multinomial sampling would).

particle_filter $(\mathcal{X}_{k-1}, z_k)$
Step 1: Diffusion
$\bar{\mathcal{X}}_k = \mathcal{X}_k = 0$
for $m = 1$ to M **do do**
sample $x_k^{[m]} \sim p(x_k|x_{k-1}^{[m]})$
$w_k[m] = p(z_k|x_k^{[m]})$
$\bar{\mathcal{X}}_k + < x_k^{[m]}, w_k^{[m]} >$
end for
Step 2: Resampling
for $m = 1$ to M **do do**
draw particle i with probability $\propto w_k^{[i]}$
add $x_k^{[i]}$ to $\mathcal{X}_k$
end for return $\mathcal{X}_k$

Fig. 4: Particle Filter Algorithm

C. Combination

As the reader might have noticed, so far we have presented traditional approaches for object tracking, namely KF and PF. The later, in our implementation, did not even include any motion model. In the following we present our combination of KF and PF which aims to combine both approaches to generate an even more robust model. Hence, our goal here is not to provide a very complex model but a simple robust model that takes advantage of the aforementioned models.

We first use the Particle Filter, as previously explained, to obtain an estimation of the position of the object, i.e. x_k^{est} as proposed in (17). We then use it as the measurement for KF at time step k. We use the motion models previously explained (constant velocity and constant acceleration).

Whenever there are occlusions, i.e. the ball is occluded, the PF stops working properly, since it has no good measurements and only the motion model is insufficient (it only spreads the particle cloud, remember (15)). This means that the PF can not provide the KF with good measurements (or estimations). To overcome this, we alert the KF whenever no white regions are detected after the image processing. This is done by setting the measurement noise covariance matrix Q_k to a very high value, which ensures that the KF will rely on its motion model to estimate the position of the object rather than on the measurements, which are poor. As explained in the description of the KF method, we set Q_k by using the variance of the matrix $I_K(k)$, obtained after filtering the original image and applying the circular kernel from (7).

The algorithm for this combined approach is shown in Fig. 5.

kalmanized_particle_filter $(\mu_{k-1}, \Sigma_{k-1}, z_k)$

$z_k^{PF} \leftarrow$ filtered frame at time step k
if no white region in z_k^{PF} **then**
Increase Q in KF
end if
$\mathcal{X}_k \leftarrow$ **particle_filter** $(\mathcal{X}_{k-1}, z_k^{PF})$
$z_k^{KF} \leftarrow 1/M \sum_{x_k^{[m]} \in \mathcal{X}_k} x_k^{[m]}$
$\mu_k, \Sigma_{k-1} \leftarrow$ **kalman_filter** $(\mu_{k-1}, \Sigma_{k-1}, z_k^{KF})$
return μ_k

Fig. 5: Kalmanized Particle Filter Algorithm

IV. Results

In this section we show and briefly discuss the results obtained when using the methods explained so far for object tracking. As already highlighted in the previous section, we will use a 1 minute length video which captures a Nintendo Pinball game play. However, we restrict the analysis to the first 15 seconds, since it is enough to gain some insights about the performances of the respective filters.

In the following, we attach links to YouTube videos where you can see how each method performed:

- KF constant velocity (video)
- KF constant acceleration (vide)
- PF
- KPF constant velocity (video)
- KPF constant acceleration (video)

Table I lists the MSE of the estimations for the first 15 seconds where no particle deprivation happened. To compute the MSE, we used as a reference image processing tools comparing pixel values and removed the artificially added occlusion regions. Furthermore, Fig. 6a and Fig. 6b illustrate the squared error for all frames within the 15 second sequence.

TABLE I: Results when there is no particle deprivation.

Method	Mean square error
KF (constant velocity)	41.70
KF (constant acceleration)	48.08
PF	110.15
KPF (constant velocity)	84.32
KPF (constant acceleration)	88.68

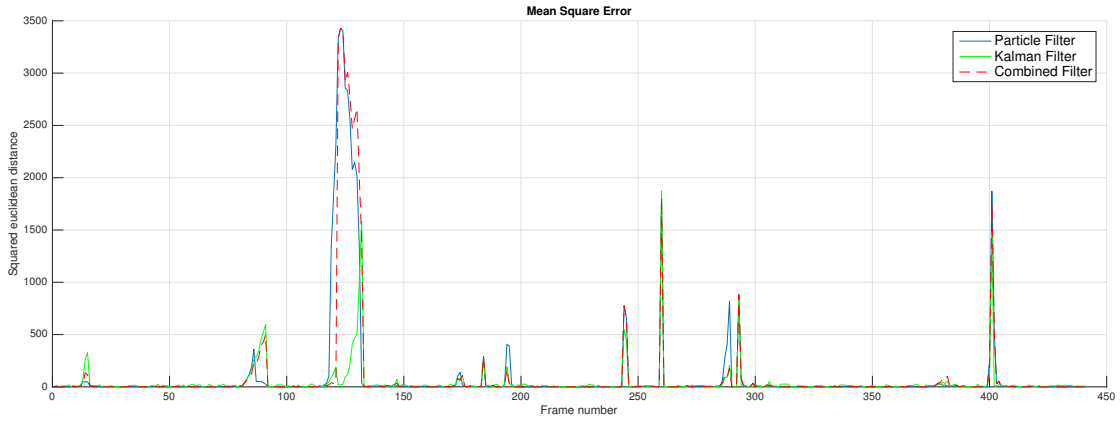
In addition, Table I shows the MSE results when particle deprivation happens. As expected, we observe a notable decrease in the performance of of the PF and KPF methods. Additionally, Fig. 6c illustrates the time evolution of the squared error.

TABLE II: Results when particle deprivation happens.

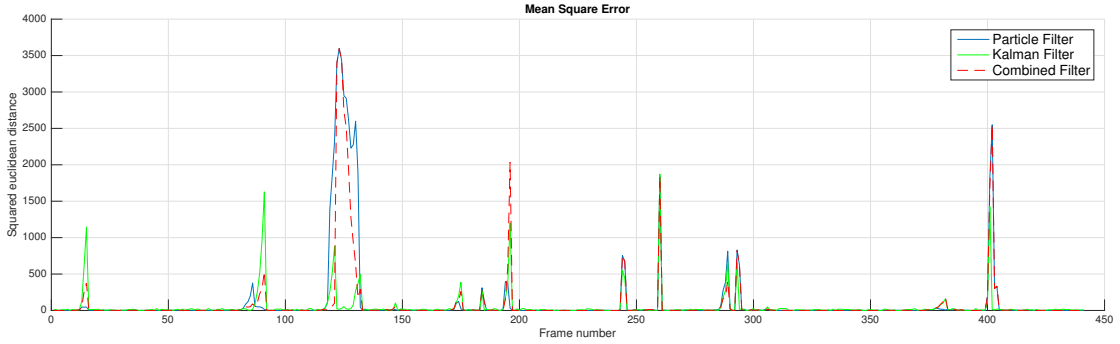
Method	Mean square error
KF (constant velocity)	41.7
KF (constant acceleration)	48.08
PF	8983.22
KPF (constant velocity)	3568.71
KPF (constant acceleration)	9003.51

A. Discussion

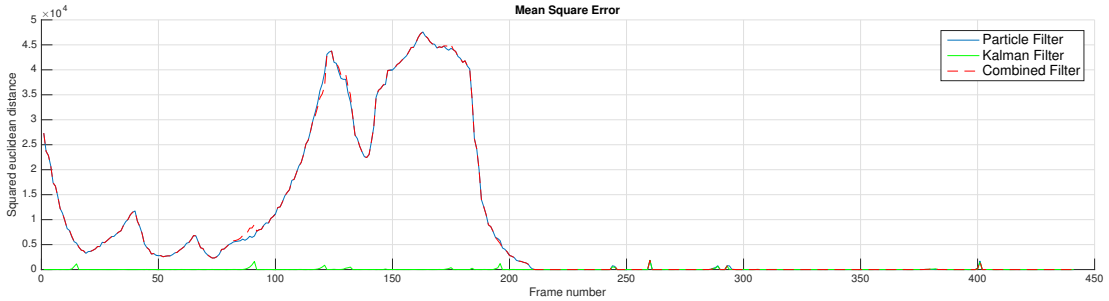
From the results, we observe that the performance of the PF and the KPF highly depends on whether particle deprivation



(a) Squared error for each frame for a constant velocity model. Kalman Filter turned out to perform at best



(b) Squared error for each frame for a constant acceleration model. Kalman Filter turned out to perform at best



(c) In this execution, we observe the phenomena of particle deprivation. This is seen in the high values of squared error for some frame intervals, which ultimately correspond to instances when the ball is occluded and hence its track is lost by the particle cloud.

Fig. 6: MSE for each frame for different executions of the 15 second frame sequence. Peaky values are usually associated with frame intervals where the ball is occluded.

occurs or not. If that is the case, i.e. particle depletion occurs, then their performance is extremely poor. This is due to the fact that the particle cloud loses track of the ball and is not able to obtain new accurate measurements. Find an example of this in our pinball video in thislink. A possible solution to this is to modify the diffusion step, by increasing the coefficients of the noise matrix which ultimately allows the PF to explore more states. In this regard, we have to bear in mind that, as shown by (15), we are using a very simplistic PF with no motion model at all.

When there is no particle deprivation, from the results in Tables I and II we can assert that the combination of PF and KF slightly increases the performance of the PF alone.

Occlusion intervals are shown in Fig. (6) as peaks in the squared error. In this regard, we observe that the KF is much less affected by them. One of the keys here has been the tuning of Q_k when there are occlusions, previously explained.

V. Conclusion

We conclude that using KF after PF can slightly improve the performance of the PF alone. However, this has been proved for a very simple PF and should be verified in the future with a more complete PF. In particular, we observe that the combination, i.e. KPF, is still very sensitive to particle deprivation when there are occlusions. Hence, it might be interesting to explore techniques, such as randomly adding new particles or increasing M to reduce its effects.

In addition, we can assert that kernels can be very helpful in the filtering of the image in order to obtain suitable measurements. Nonetheless, in the future further Kernels should be tested and a more detailed analysis should be done about it.

Furthermore, we find it interesting to remark the fact that we have used a simple linear Kalman Filter and not any of its variants (EKF, UKF, iEKF). This might be due to the simplicity of the ball movements in the video. In the future, it can be enriching to probe with videos of different nature, where such variants might be helpful.

VI. Acknowledgement

We have partially used some of the code of LAB2, in particular those functions related with the re-sampling. For this, we acknowledge course Teacher Prof. John Folkersson and course TAs as its authors.

References

[1] Bringing big data to the enterprise. https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html. Accessed: 2016-27-12.

[2] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

[3] Ted J Broida and Rama Chellappa. Estimation of object motion parameters from noisy images. *IEEE transactions on pattern analysis and machine intelligence*, (1):90–99, 1986.

[4] David Beymer and Kurt Konolige. Real-time tracking of multiple people using continuous detection. In *IEEE Frame Rate Workshop*, page 53. Citeseer, 1999.

[5] Romer Rosales and Stan Sclaroff. 3d trajectory recovery for tracking multiple objects and trajectory guided recognition of actions. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

[6] Jun S Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998.

[7] N Shepard and MK PITT. Filtering via simulation: auxiliary particle filter. *Journal of the American Statistical Association*, 94:590–599, 1999.

[8] B-N Vo, Sumeetpal Singh, and Arnaud Doucet. Sequential monte carlo methods for multitarget filtering with random finite sets. *IEEE Transactions on Aerospace and electronic systems*, 41(4):1224–1245, 2005.

[9] Duc Phu Chau, François Bremond, and Monique Thonnat. Object tracking in videos: Approaches and issues. *arXiv preprint arXiv:1304.5212*, 2013.

[10] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Transactions on pattern analysis and machine intelligence*, 25(5):564–577, 2003.

[11] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.