# Kalman Filters Explanation Note

**Document Control Information**

| Version | Revision Description | Date | Author | Reviewer |
|---|---|---|---|---|
| 1.0 | Initial Draft | 10th October | Tilak | Charu K Bharadwaj |
| | | | | |

## Table of Contents

# 1. Introduction

This document mainly tries to summarize the learnings of Kalman filter theory and the practical implementations done to prove that learnt theory.

*Note: Many of the images attached in the theoretical section will be image excerpts extracted from the writer's hand written notes.*

# 2. Theory:

Kalman filters help to estimate the unknown parameter from a known quantity/parameter of the same system.

- Kalman filters are basically inspired from the Recursive Bayesian Probability concept.
- Kalman filter process revolves around the calculation of an important factor called Kalman Gain (K), through which it tells whether to trust a real measurement or an estimate made for the same

## 2.1 Recursive Bayesian Probability

**Statement:**
Given some prior data, the likelihood/probability of a future event can be guessed can be guessed accurately.

- As prior data becomes more better (i.e the feedback accuracy), the better will be the likelihood/ future event prediction

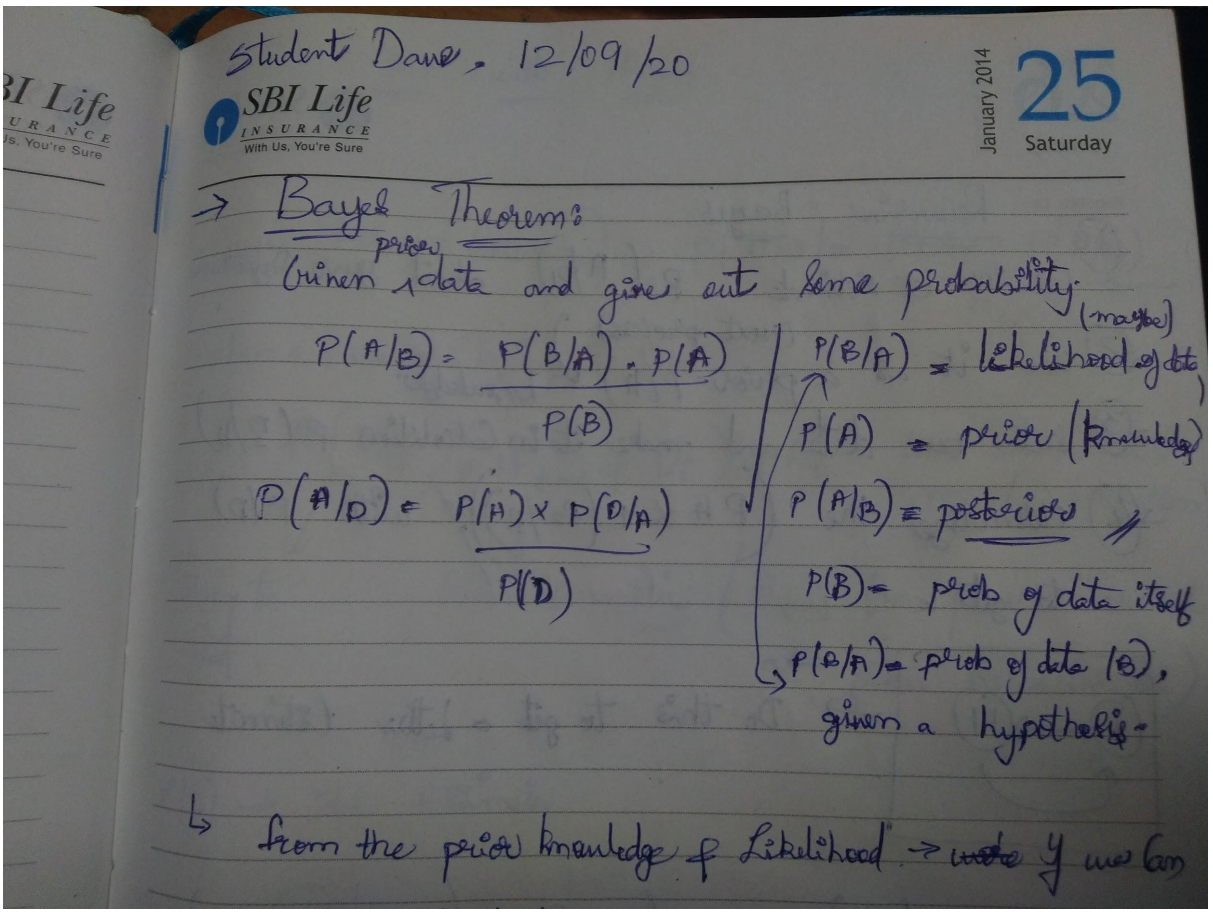



**Image1: Bayes theorem equation**

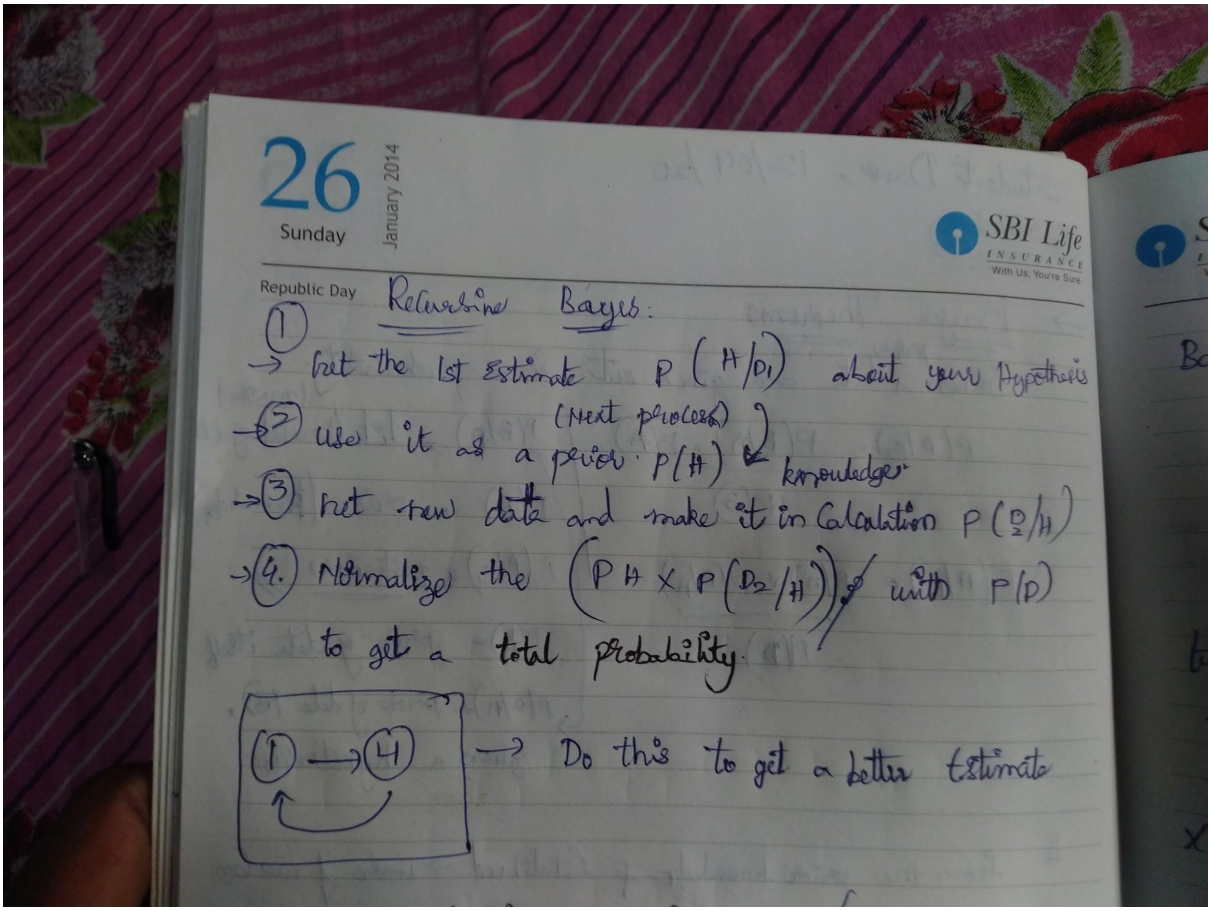



**Image2: Recursive Bayesian Process (building upon the previous image equations)**

## 2.2 Kalman Filter

### 2.2.1 Introduction:

Below steps should be followed for realizing the Kalman filter:

1. **Establish a Mathematical model for your system**: A Mathematical model for the system were trying to analyse should be made (system should be linear)
2. **Derive the parameters for your Mathematical Model:** The parameters of the derived Math model are calculated with some assumptions in place
3. **State Prediction (/Time) Update & Measurement Update Stages**: These two stages are the important part of Kalman filters concept. They Steer the estimation vs measurement decision process using the calculated Kalman Filter gain (K)
    a. Kalman gain - tells whether to use Measurement or the Prediction in the final value know how

**Note**: Extended Kalman Filter is used for state estimation of a Nonlinear system, but the scope of this document is only limited to Kalman filters (i.e considering only the linear filters)

## 2.2.2 System Mathematical Model Derivation:

The practical implementation phase of this document has a moving body on a paper
Hence we will try to apply the universal equations of motion to deduce the mathematical model of this moving body system.

### 2.2.2.1 Basic Laws of Motion:

**$S = ut + ½ at^2$**
(S = Distance Covered, u - initial velocity, t - time, a - acceleration/Control input)

**$V = u + aT$**
(V - Velocity, a - acceleration, T - time)

**$X_t = X_{t-1} + U_{t-1}*T + ½ aT^2$ ---------- (1)**
(Predicted displacement/Position = Initial/Previous displacement ($X_{t-1}$) + (equations which describe the new occurred movement - $U_{t-1}*T + ½ aT^2$ ))

**$V_t = V_{t-1} + aT$ ----------(2)**
**(**Predicted Velocity = Initial/Previous Velocity($V_{t-1}$) + newly added velocity component (aT) **)**

### 2.2.2.2 Math Model derivation for the system:

**System:** A moving body (a 2D space is considered using image processing techniques)

**Math Model:**

- **$Q\_Estimate_t = A (Q\_Estimate_{t-1}) + B(U_t) + Ex$ ---------- (3)**
  Q_Estimate - Present State Estimate Vector Matrix
  $Q\_Estimate_{t-1}$ - Previous state Estimate Matrix
  U - Control input i.ee Given Acceleration magnitude
  Ex - process noise in the system,
  A - State Transition Matrix
  B - Control input Matrix

**It says Present state estimate is sum of Previous state estimate, Given control i/p or acceleration in our system and the process noise**

- **$Q\_Measured_t = C (Q\_Estimate_{t-1}) + Ez$ ---------- (4)**
  Q_Measured - Measured tracking data of the object,
  C - Transformation Matrix (Tells the area of interest (Or) O/p)
  Ez- Measurement Noise

**It says the Measured value will be some factor (C) based Estimated value plus the Measurement noise. (or) Measured value O/p = Estimate transformed by a factor of C + Measurement Noise (Ez)**

- [As generally the real measured output **$Q\_Measured_t$** , will be near to previously calculated Estimate **$Q\_Estimate_{t-1}$** ]

**Note**:

1. As we see Noise signals, define the change/the way a system will turn out to be
2. Hence, the better we estimate the noise, the better we estimate the real signal from Kalman Filter.

### 2.2.3 Calculation of Parameters for the derived System Mathematical model:

Equations 3 & 4 described the Math model for our moving body.

From the above equations (1) & (2), below equations are derived for X & Y axis directions of the moving body/system considered

$X(p)_t$ (Position in X direction) = $X(p)_{t-1} + X(v)\,T + (T^2/2)\,a$ ---------- **(5)**

$X(v)_t$ (Velocity in X direction) = $X(v)_{t-1} + Ta$ ---------- **(6)**

$Y(p)_t$ (Position in Y direction) = $Y(p)_{t-1} + Y(v)T + (T^2/2)\,a$ ---------- **(7)**

$Y(v)_t$ (Velocity in Y direction) = $Y(v)_{t-1} + Ta$ ----------**(8)**

**[Tells that the present Position/Velocity is Sum of previous value & the newer component governed by the Eqns (1) & (2) ]**

### 2.2.3.1 Realized relations:

(Position prop to $T^2/2$) **----------(9)**
(Velocity prop to T) **----------(10)**

The above relations are realized from equation 7 & 8

### 2.2.3.2 State Position/Prediction Matrix

By comparing above X & Y direction P&V equations (5-8) with that of the System model Eqn (3) & (4) gives the below matrix derivations of A & B.

- If the matrix is multiplied, we get back the 4 equations from 5 to 8.

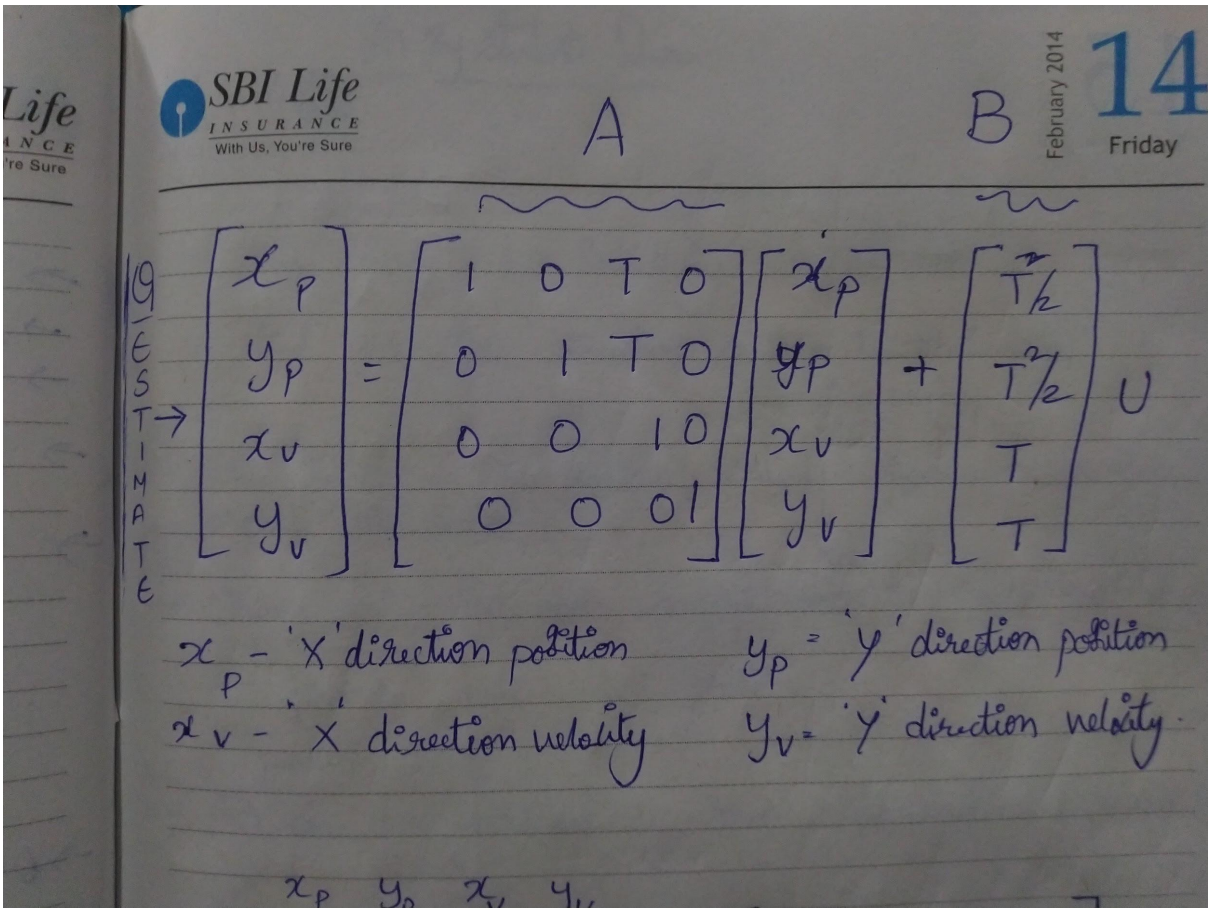



**Image3: A&B Matrices Derivation**

Parameters in above Image:

- **U (**control input**)** = a (acceleration given to the body to move)
- State Estimation Matrix **($Q\_Estimate_t$ )** =

[Xp]
[Yp]

$[Xv]$
$[Yv]_{4x1}$

- Notation $A_b$ : A - axis X or Y & b - Position (p) or Velocity (v)

### 2.2.3.3 Transformation Matrix Derivation:

We multiply our wanted outputs (Xp & Yp - Position of both X & Y axis) matrix with that of the created State Estimation Vector (**Q_Estimate$_t$**) matrix.

C - Transformation Matrix (this transforms the Estimate by )

- Here we need only Position in X & Y only. Hence the remaining terms (X & Y direction Velocity terms) will go to 0.

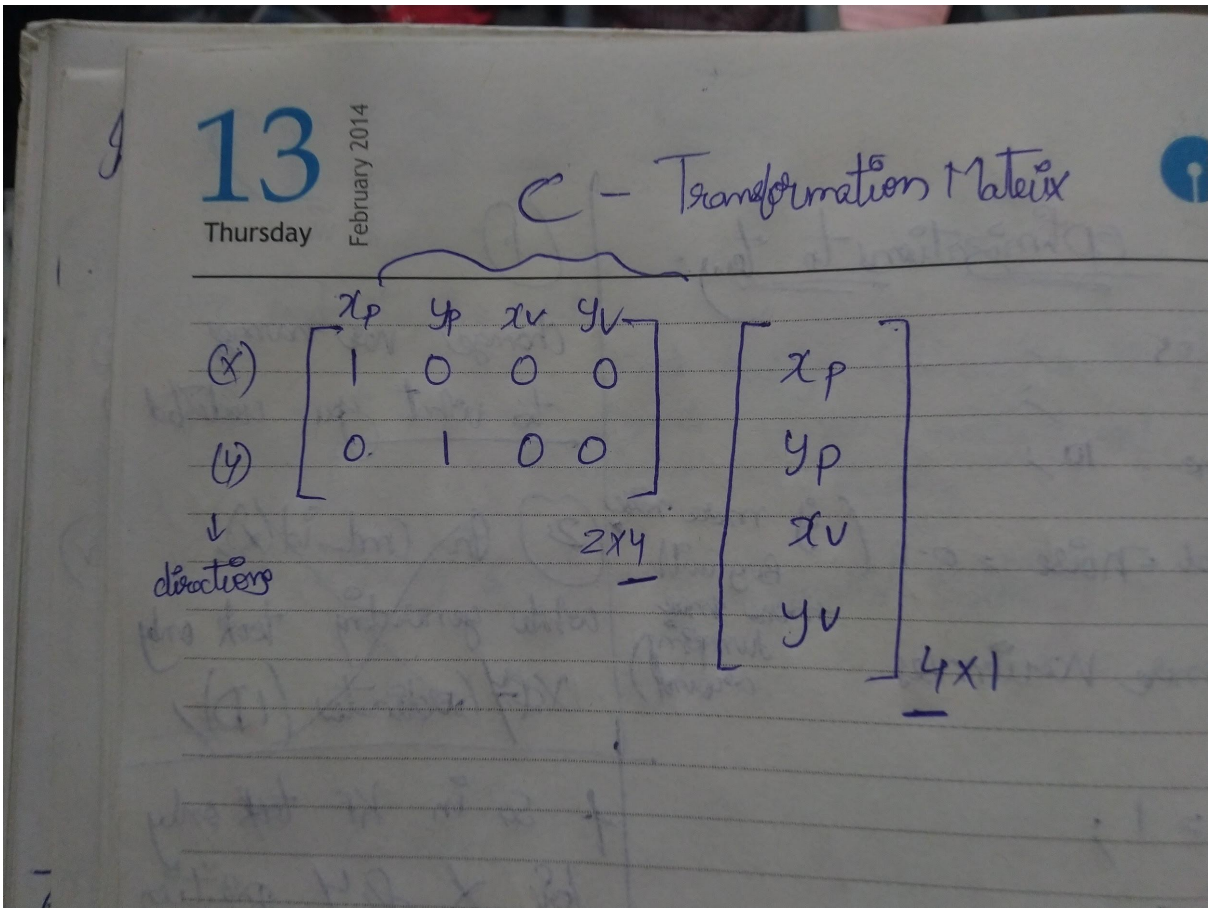



**Image4: Matrix C derivation**

Thus Matrix C is derived.

The above matrix Multiplication gives us the required outputs wrt the created State Estimation Vector **Q_Estimate$_t$**

**Note**: A,B,C are matrices from the sec 2.2.3.2 & 2.2.3.3 respectively

### 2.2.3.4 Covariance & Variance Concept:

**Variance**: Measurement of uncertainty

**Covariance**: Measurement of uncertainty between two variables/quantities/subjects
**Covariance Matrix** = [Cov(A,A) Cov(A,B)]
[Cov(B,A) Cov(B,B)] $_{2x2}$
Cov (A,A) = variance of that Same quantity) itself
Cov (A,B) = E(A,B) - E(A)E(B)
[E - Expectation = weighted average (Or) **Mean** ]
Cov (B,A) = Cov (A,B) (Both the Variance measurements between two variables will be the same)

### 2.2.3.5 Ex and Ez (Process and Measurement Noise Covariance matrix) Derivation:

**Ex Process Noise Covariance Matrix :**

From the equations of 9 & 10, we derive the Ex covariance matrix. As the dependent terms of (X,Y) (Y,X) will be zero as we have an assumption that noises are all independent of each other (See Assumptions point no 1)

**Ez - measurement noise covariance matrix:**

Noise of only X & Y direction is included. Since remaining terms are dependent, (See Assumptions point no 1)

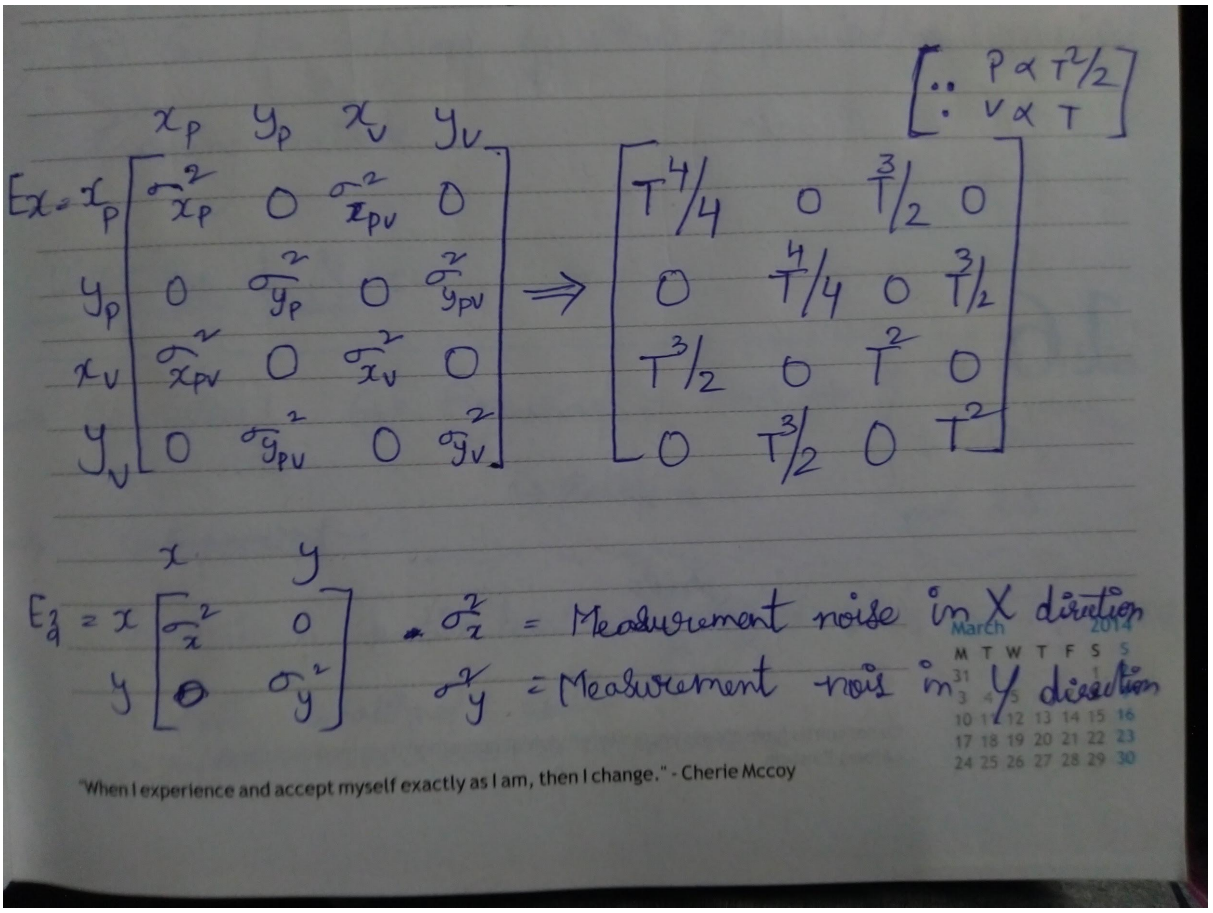



**Image5: Ex and Ez Matrices Derivation**

## 2.2.4 State Prediction (/Time) Update & Measurement Update:

The below Two stages of equations are the Crux of Kalman filter. They iteratively continue one stage after another ① -> ② **->** ① ---- ….

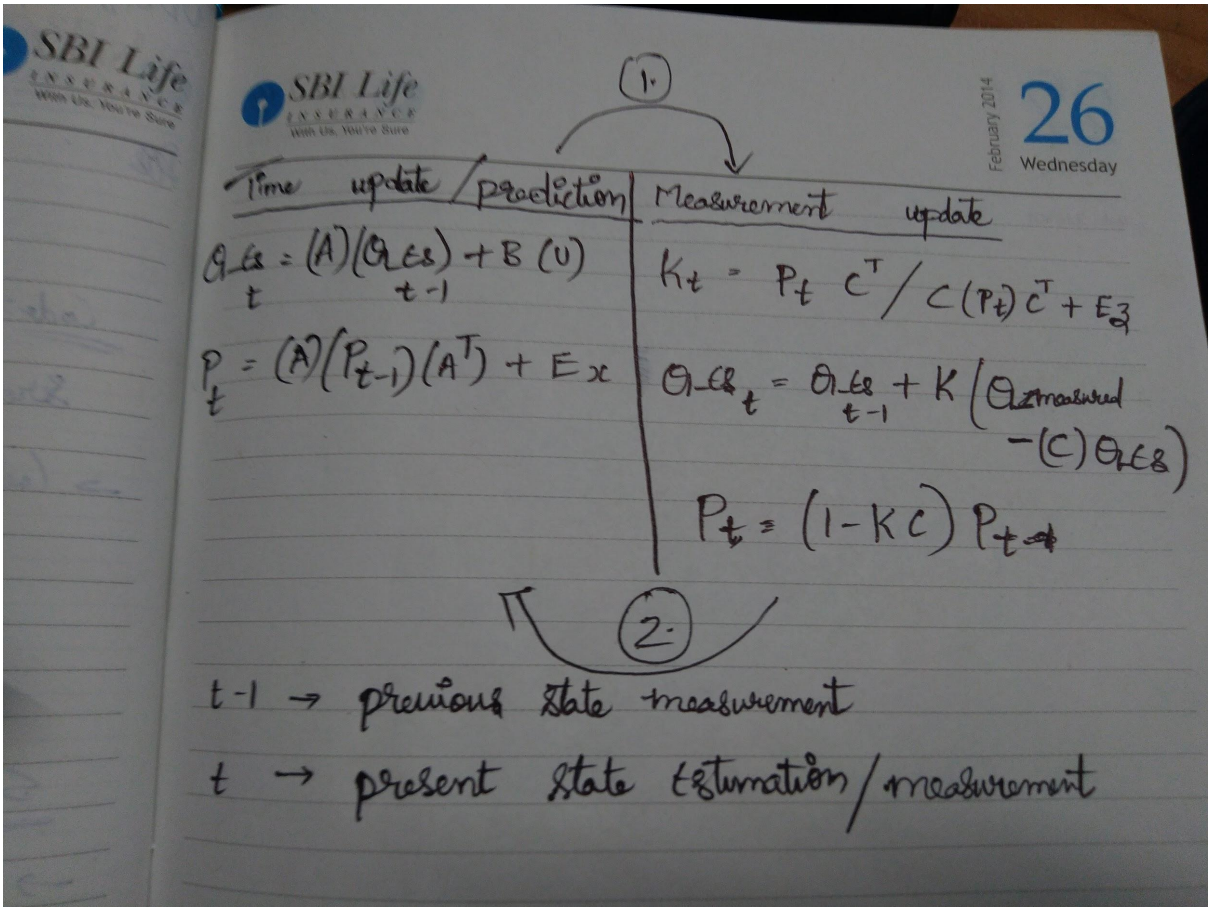



**Image6: Prediction & Measurement Stages Equations**

① **State prediction/Time Update stage**: All the estimation calculations are done, namely:

- **Q_Estimate$_t$** vector
- **$P_t$** - Error Covariance Estimation

② **Measurement Update Stage:** In this stage the following are calculated:

1. **Kalman gain (K) =** Based predominantly on Process Error Covariance ($P_t$) in Numerator and Measurement Noise (Ez) in denominator.

2. Here the real measurement is made from the system and **$Q\_Measured_t$** (real measurement of the sensors giving position in X & Y) and again (**$Q\_Estimate_t$**) & **$P_t$** is calculated.

**$Q\_Estimate_t = Q\_Estimate_{t-1} + K(Q\_Measured_t - C*Q\_Estimate_t)$ ---------- (11)**

**$K = P_t*C^T / C*P_t*C^T*Ez$ ----------(12)**

Note:

1. *This stage essentially corrects the estimation of the first stage by using Kalman gain, as a deciding factor on whom (Estimation $Q\_Estimate_t$ or Measurement $Q\_Measured_t$ ) to contribute to the ultimate output*
2. ② stage **equations are got** by taking the partial derivative of ① (for a min estimation of error) - More should be researched on how these equations are mathematically derived.

**Iteration of Steps** ① to ② & ② to ①

- All the updated values which are obtained in this stage become the (t-1) terms for the ① and vice-versa
- **Example**: $Q\_Estimate_t$ and $P_t$ of the Measurement update stage will become t-1 terms for the two equations in ① - Time update/State Prediction stage.
    - Using those terms $Q\_Estimate_{t-1}$ & $P_{t-1}$ from ②, prediction stage ①, now predicts $Q\_Estimate_t$ & $P_t$ more accurately.

#### 2.2.4.1 Importance of K (Kalman Gain):

Kalman gain ultimately helps in combining both measurement ② and prediction ①stages to find optimal estimate

- K value tells - to how seriously it should consider the Measurement ($Q\_Measured_t$ ) vs approximation ($Q\_Estimate_t$ ) terms in calculation of the final estimate value ($Q\_Estimate_t$)
- **Example**: Consider Equation 11 & 12 cases:
    - **Case 1**: If the Measurement noise is very high, K will be very less .. Hence measurement will be not given a big role to play in the final Estimate & vice-versa
    - **Case 2**: If error in Prior estimate is less, then estimate contribution is taken more for the final calculation of the estimate

**Note:** Hence the noises are very important in making the system to work efficiently. Hence the Noises should be calculated or estimated very accurately. As they define, how the Kalman filter is going to work.

## 2.3 Assumptions:

1. All the noises are independent of each other/ each is of white (gaussian Glossary [1]) noise in nature (property used in Covariance matrix calculation).

    a. Hence if any dependent covariance terms (X,Y) are calculated, they will become zero (Because of their independence nature, no covariance value exists between them)
2. The taken system is considered to be linear (For nonlinear system a Extended Kalman Filter solves the issue -- see Sec ‘Next Steps’ )

# 3. Practical Implementation Phase(Proving the theory):

**Full Target Statement:**To prove the Kalman Filter working, generate an (Moving object) Images based mat binary files, which when used as an input to the Kalman filter system of equations, should give an accurate & better position tracking estimation than a normal averaging (/image processing) method based position tracking.

The above target will be achieved by explaining in two steps.

1. Image Recognition based Binary File generation
2. Kalman Filter Implementation.

## 3.1 Image Recognition based Binary File generation

**Aim** :
Applying image processing techniques to a bunch of moving object images, to ultimately generate a Matlab Binary File (.mat files)

**Procedure**:

- Before generating the above Image based Matlab binary files, below steps are made to tackle the noisy images data we got.

1. **Averaged background subtraction:** Getting the Common background of all the images (Averaged Background) and subtracting that from the whole image gives the area of interest -> Averaged Moving object.
2. **Noise reduction via image smoothing using 2-d gaussian filter:** Convolution of the generated Averaged Moving object with Gaussian Filter (to get an overall smoothened image)
    a. Used ‘fspecial’ & ‘filter2’ functions to obtain the gaussian filter & Gaussian smoothed images respectively.
3. **Threshold detection:** Set the threshold (for the core moving body identification) for filtering all the images (from step2 ) and copy only them into a Image Binary (CMD_Idx) file.
4. Different noise parameters (No noise ,with Noise ,No Tracking data at all) based Image binary files are generated:

a. **No Noise Mat file**: CM_idx_easier.mat - Actual Tracking data as it is recorded
b. **With Noise:** CM_idx_harder.mat - Some images motion data is not recorded in the Image binary files, instead some random noise is intentionally added
c. **No tracking At all:** CM_idx_harder.mat - Some images motion data (say from image 200-300) is not recorded in the ultimate image binary file)

**Code:**

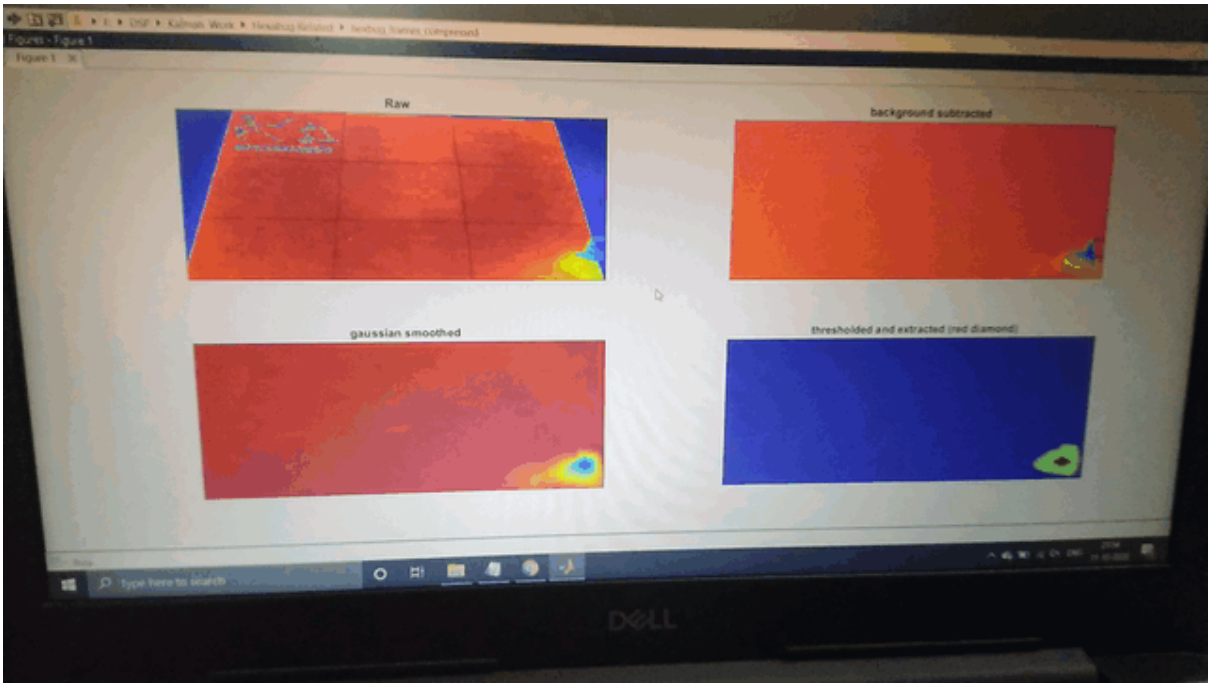
```
%the code finds the hexbug buy
% 1) Averaged background subtraction
% 2) Noise reduction via image smoothing using 2-d gaussian filter.
% 3) Threshold and point detection in binary image.

clear all;
close all;
set(0,'DefaultFigureWindowStyle','docked');

base_dir = 'F:\DSP\Kalman  Work\Hexabug Related\hexbug_frames_compressed\';

cd(base_dir);

%% get listings of frames so that you can cycle through them easily.
f_list =  dir('*png');


%% make average of background images (i.e. images with no objects of interest)
% Here we just read in a set of images (N) and then take the average of
% them so that we are confident we got a good model of what the background
% looks like (i.e. a template free from any potential weird image artifacts

N = 20; % num of frames to use to make an averaged background...that is, images with no bug!
img = zeros(288,352,N); %define image stack for averaging (if you don't know what this is, just load the image and  check it with size())
for i = 1:N
    img_tmp = imread(f_list(i).name); %read in the given image
     img(:,:,i) = img_tmp(:,:,1); % we don't really care about the rgb image values, so we just take the first dimension of the image
end
bck_img = (mean(img,3)); %take the average across the image stack..and bingo! there's your background template!
subplot(121);imagesc(bck_img)
subplot(122);imagesc(img(:,:,1))
colormap(gray)
clear img; % free up memory.

%initialize gaussian filter
```

```
%using fspecial, we will make a gaussian template to convolve (pass over)
%over the image to smooth it.
hsize = 20;
sigma = 10;
gaus_filt = fspecial('gaussian',hsize , sigma);
subplot(122); imagesc(gaus_filt)
subplot(122); mesh(gaus_filt)
colormap(jet)

%this one is just for making the coordinate locations more visible.
SE = strel('diamond', 9); %another tool make for making fun matrice :) this one makes a matrice object for passing into imdilate())



%% iteratively (frame by frame) find bug!
CM_idx = zeros(length(f_list),2); % initize the variable that will store the bug locations (x,y) -- I.ee for each frame/img - xy are stored in CMD_IDX [m,n]

for i = 1:2:length(f_list)

      img_tmp = double(imread(f_list(i).name)); %load in the image and convert to double too allow for computations on the image
     img = img_tmp(:,:,1); %reduce to just the first dimension, we don't care about color (rgb) values here, since it is a normal grey scale image -- first dimension is enough
   subplot(221);imagesc(img);
   title('Raw');

   %{
   %VERY HARD TRACKING
   %for frames 230:280, make the bug very hard to track
   if (i > 230) && (i < 280) && (mod(i,3) == 0 )
      J = imnoise(img,'speckle');
      img = img+J*200;
   end
   %}



   %subtract background from the image
   sub_img = (img - bck_img);
   subplot(222);imagesc(sub_img);
   title('background subtracted');
   %gaussian blurr the image
   gaus_img = filter2(gaus_filt,sub_img,'same');
   subplot(223);imagesc(gaus_img);
   title('gaussian smoothed');
```

```
    %threshold the image...here i just made a quick histogram to see what
    %value the bug was below
    subplot(224);hist(gaus_img(:));
    thres_img = (gaus_img < -15);
    subplot(224);imagesc(thres_img);
    title('thresholded');


    %% TRACKING! (i.e. get the coordinates of the bug )
    %quick solution for finding center of mass for a BINARY image
    %basically, just get indices of all locations above threshold (1) and
    %take the average, for both the x and y directions. This will give you
    %the average location in each dimension, and hence the center of the
    %bug..unless of course, something else (like my hand) passes threshold
    %:P
    %if doesn't find anything, it randomly picks a pixel
    [x,y] = find (thres_img);
    if ~isempty(x)
        CM_idx(i,:) = ceil([mean(x) mean(y)]+1); % i used ceiling to avoid zero indices,
but it makes the system SLIGHTLY biased, meh, no biggie, not the point here :).
    else
       CM_idx(i,:) = ceil([rand*200 rand*200]);
    end

   %,CM_idx(i,2)

    %{
    %NOT SO HARD TRACKING
     %for frames 230:280, make the bugtracking just a lil noisy by randomly sampling
    %around the bugtracker
    if (i > 230) && (i < 280) && (mod(i,2) == 0 )
      CM_idx(i,:) = [round(CM_idx(i,1) + randn*10) round(CM_idx(i,2) + randn*10)];
    end
    %}

     %{
    %NO  TRACKING
     %for frames 230:280, make the bugtracking just a lil noisy by randomly sampling
    %around the bugtracker
    if (i > 230) && (i < 280)
      CM_idx(i,:) = [NaN NaN];
    end
    %}


    %% now, we visualize everything :)
```

```
    %create a dilated dot at this point for visualize
    %make a binary image with a single coordinate of bug = 1 and rest zeros.
    %then dilate that point to make a more visible circle.
    bug_img = zeros(size(thres_img));
    bug_img(CM_idx(i,1),CM_idx(i,2)) = 1;

    %{
    % if you are running the "no tracking segment above, you'll need to
    % skip over that  segment, and thus use this code
    if ~((i > 230) && (i < 280))
        bug_img(CM_idx(i,1),CM_idx(i,2)) = 1;
    end
    %}

     bug_img = imdilate(bug_img, SE); % SE is the structural element object we set
previously
    subplot(224);imagesc(thres_img + bug_img);
    title('thresholded and extracted (red diamond)');
    axesHandles = get(gcf,'children');
    set(axesHandles, 'XTickLabel', [], 'XTick', []);
    set(axesHandles, 'YTickLabel', [], 'YTick', [])  ;

    pause(.01)
end

%save out the hexbug coordinates

%save('CM_idx_easier.mat', 'CM_idx')
```

**Results:**




**Gif1: A rough Object tracking achieved by using image processing techniques**

## 3.2 Kalman Filter Implementation

**Aim:** KF equations discussed in the theory section will be applied here to get better position tracking estimates in both X & Y Direction.

**Procedure**:

- Use the state model equations (3) & (4) derived in Sec 2.2.2.2
- As we intend to track the position of the moving object in both the X & Y direction (2D space), our state estimation vector will be with 4x1 size as follows (Including the velocity components)
- Initial estimated values are given for all variables in equations of ① Stage & ② Stage (See Sec 2.2.4 )
- Fill the variables with the derived values from Sec 2.2.3 (With a sampling rate t = 1 )
- Apply the Kalman equations stage & and the recursively estimated result **Q_Estimate**$_t$ improves the tracking, as we go ahead with the for loop in the code.

**Code:**

```
clear all;
close all;
clc;
set(0,'DefaultFigureWindowStyle','docked')
base_dir = 'F:\DSP\Kalman  Work\Hexabug Related\hexbug_frames_compressed\';
%replace the above directory with your local images folder path
cd(base_dir);
```

```
%% get listing of frames
f_list =  dir('*png');

%% load tracking data
load('CM_idx_easier.mat') %simple tracking: continuously monitored bug
%load('CM_idx_harder.mat') %hard tracking:  segment of very noisy images and very bad tracking
%load('CM_idx_no.mat'); %missing data tracking: segment with no tracking



%% define main variables
dt = 1;  %our sampling rate
S_frame = 10; %starting frame
u = .005; % define acceleration magnitude
Q= [CM_idx(S_frame,1); CM_idx(S_frame,2); 0; 0]; %initized state--it has four components: [positionX; positionY; velocityX; velocityY] of the hexbug
Q_estimate = Q;  %estimate of initial location estimation of where the hexbug is (what we are updating)
HexAccel_noise_mag = .1; %process noise: the variability in how fast the Hexbug is speeding up (stdv of acceleration: meters/sec^2)
tkn_x = 1;  %measurement noise in the horizontal direction (x axis).
tkn_y = 1;  %measurement noise in the horizontal direction (y axis).
Ez = [tkn_x 0; 0 tkn_y];
Ex = [dt^4/4 0 dt^3/2 0; ...
    0 dt^4/4 0 dt^3/2; ...
    dt^3/2 0 dt^2 0; ...
     0 dt^3/2 0 dt^2].*HexAccel_noise_mag^2; % Ex convert the process noise (stdv) into covariance matrix
P = Ex; % estimate of initial body position variance (covariance matrix)

%% Define update equations in 2-D! (Coefficient matrices): A physics based model for where we expect the HEXBUG to be [state transition (state + velocity)] + [input control (acceleration)]
A = [1 0 dt 0; 0 1 0 dt; 0 0 1 0; 0 0 0 1]; %state update matrice
B = [(dt^2/2); (dt^2/2); dt; dt];
C = [1 0 1 0; 0 1 0 1];  %this is our measurement function C, that we apply to the state estimate Q to get our expect next/new measurement



%% initize result variables
% Initialize for speed
Q_loc = []; % ACTUAL hexbug motion path
vel = []; % ACTUAL hexbug velocity
Q_loc_meas = []; % the hexbug path extracted by the tracking algo

%% initialize estimation variables
Q_loc_estimate = [];%  position estimate
vel_estimate = []; % velocity estimate
P_estimate = P;
r = 5; % r is the radius of the plotting circle
j=0:.01:2*pi; %to make the plotting circle
```

```
for t = S_frame:length(f_list)

    % load the image
    img_tmp = double(imread(f_list(t).name));
    img = img_tmp(:,:,1);
    % load the given tracking
    Q_loc_meas(:,t) = [ CM_idx(t,1); CM_idx(t,2)];

    %% do the kalman filter

    % Predict next state of the Hexbug with the last state and predicted motion.
    Q_estimate = A * Q_estimate + B * u;
    %predict covariance
    P = A * P * A' + Ex;
    % predicted Ninja measurement covariance
    % Kalman Gain
    K = P*C'*inv(C*P*C'+Ez);
    % Update the state estimate.
    if ~isnan(Q_loc_meas(:,t))
        Q_estimate = Q_estimate + K * (Q_loc_meas(:,t) - C * Q_estimate);
    end
    % update covariance estimation.
    P =  (eye(4)-K*C)*P;

    %% Store data
    Q_loc_estimate = [Q_loc_estimate; Q_estimate(1:2)];
    vel_estimate = [vel_estimate; Q_estimate(3:4)];

    %% plot the images with the  tracking
    imagesc(img);
    axis off
    colormap(gray);
    hold on;
    plot(r*sin(j)+Q_loc_meas(2,t),r*cos(j)+Q_loc_meas(1,t),'.g'); % the actual tracking
        plot(r*sin(j)+Q_estimate(2),r*cos(j)+Q_estimate(1),'.r');  %  the  kalman  filtered
tracking
    hold off
    pause(0.1)
end
```

### 3.2.1 Matlab/Octave Setup:

- **CMD_Id & Frames Drive Link:** https://drive.google.com/file/d/1s6clpi4xp73zFt6V7zRUu6Wg2pB4wvP6/view?usp=sharing

**Note:** Set your folder path to the above downloaded folder, in the ‘base_dir ‘ variable in code

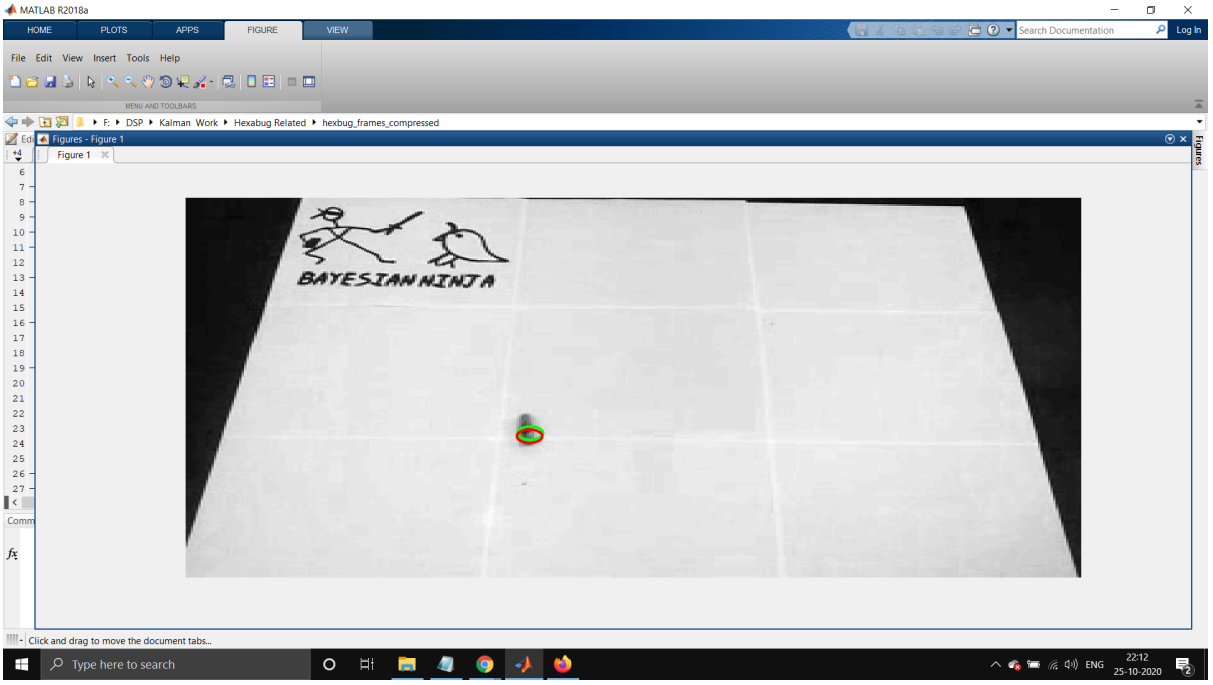
## 3.2.2 Results:

**Note:**

1. Green Circle - Image recognition based tracking
2. Red Circle - Kalman Filter estimation & measurement included tracking

### 3.2.2.1 With complete tracking data

- **Using** CM_idx_easier.mat (All tracking data is there, no anomaly expected)



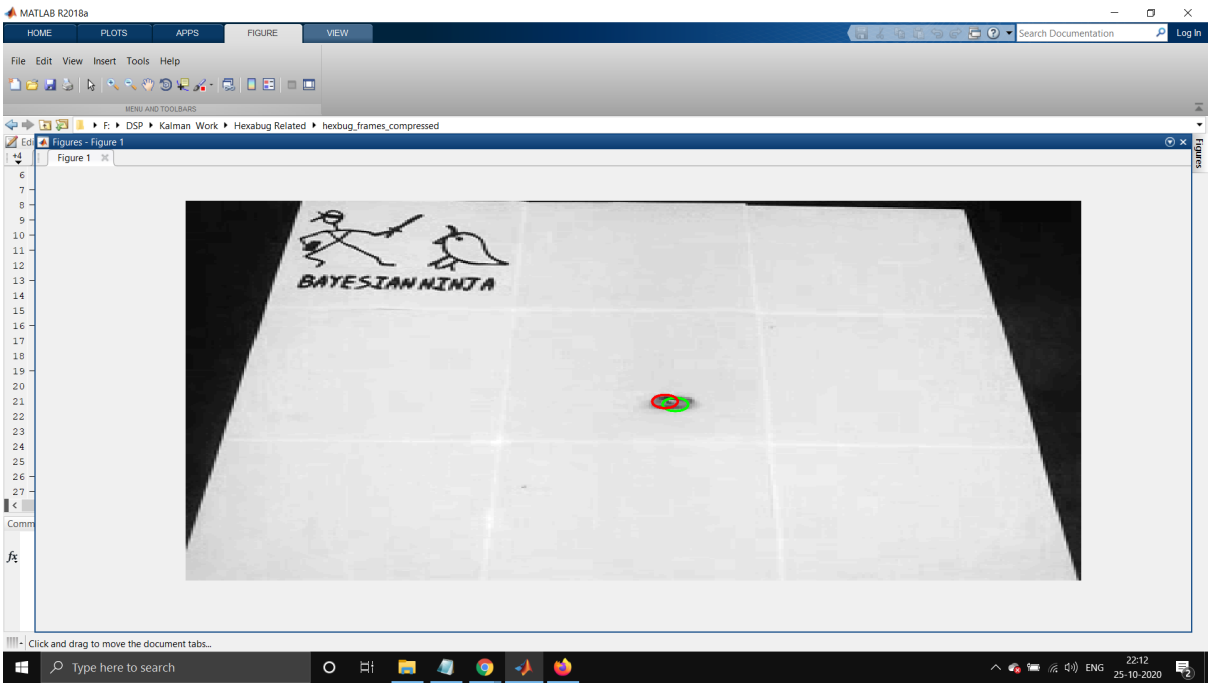
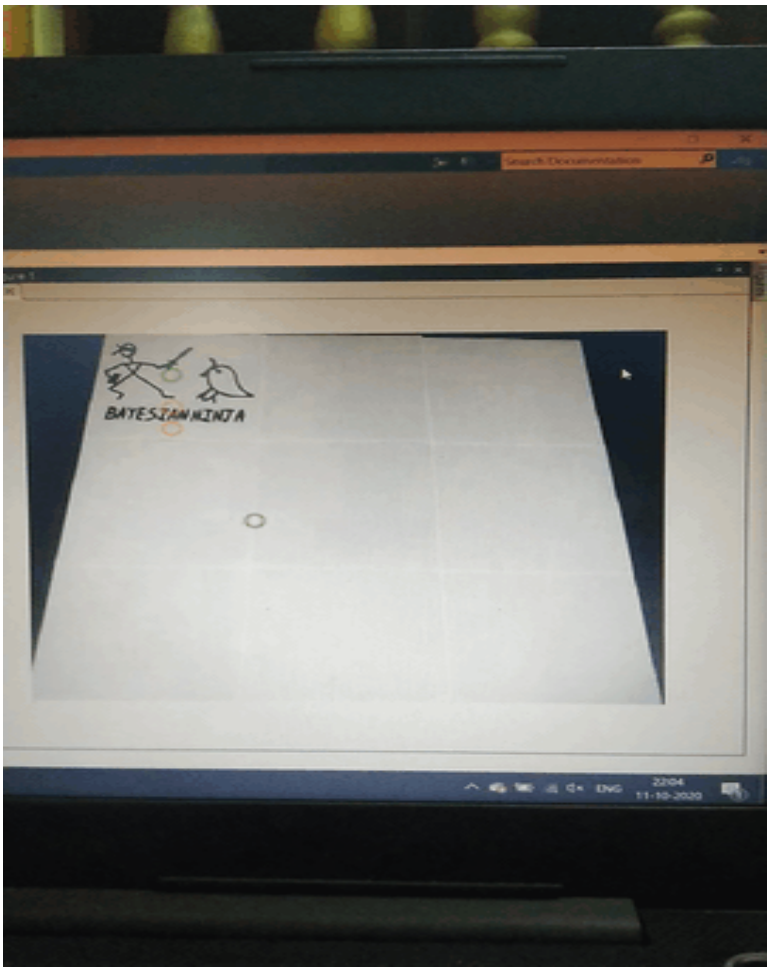



**Image 7&8: Body moving with both tracking Image processing and Kalamn Based with almost no difference**

### 3.2.2.2 Without Complete tracking data

- Using CM_idx_harder.mat (some frames Tracking data is removed from the whole )
    - Since no real measurement is there in that segment (Green Circle), Kalman just goes with the estimation it has from previous results and almost successfully tracks the object (Red circle).
        - In Sec 2.2.4.1, consider the Case 1 where no measurement data is present, then only with prediction estimates, (with the prior t-1 estimates & measurements) it can estimate the moving body.
        - This greatly explains the application of Kalman Filter use (See below images (fig) & **Gif2** which exactly depict the discussed behaviour). This can be compared to Kalman Filters application of Car's GPS approximation when it goes through a tunnel and doesn't get the GPS sensor readings, but via Kalman filter application it can approximate the moving cars postipon.




**Gif2: The moving object's position is shown by both normal tracking(G) and Kalman Prediction (R)**

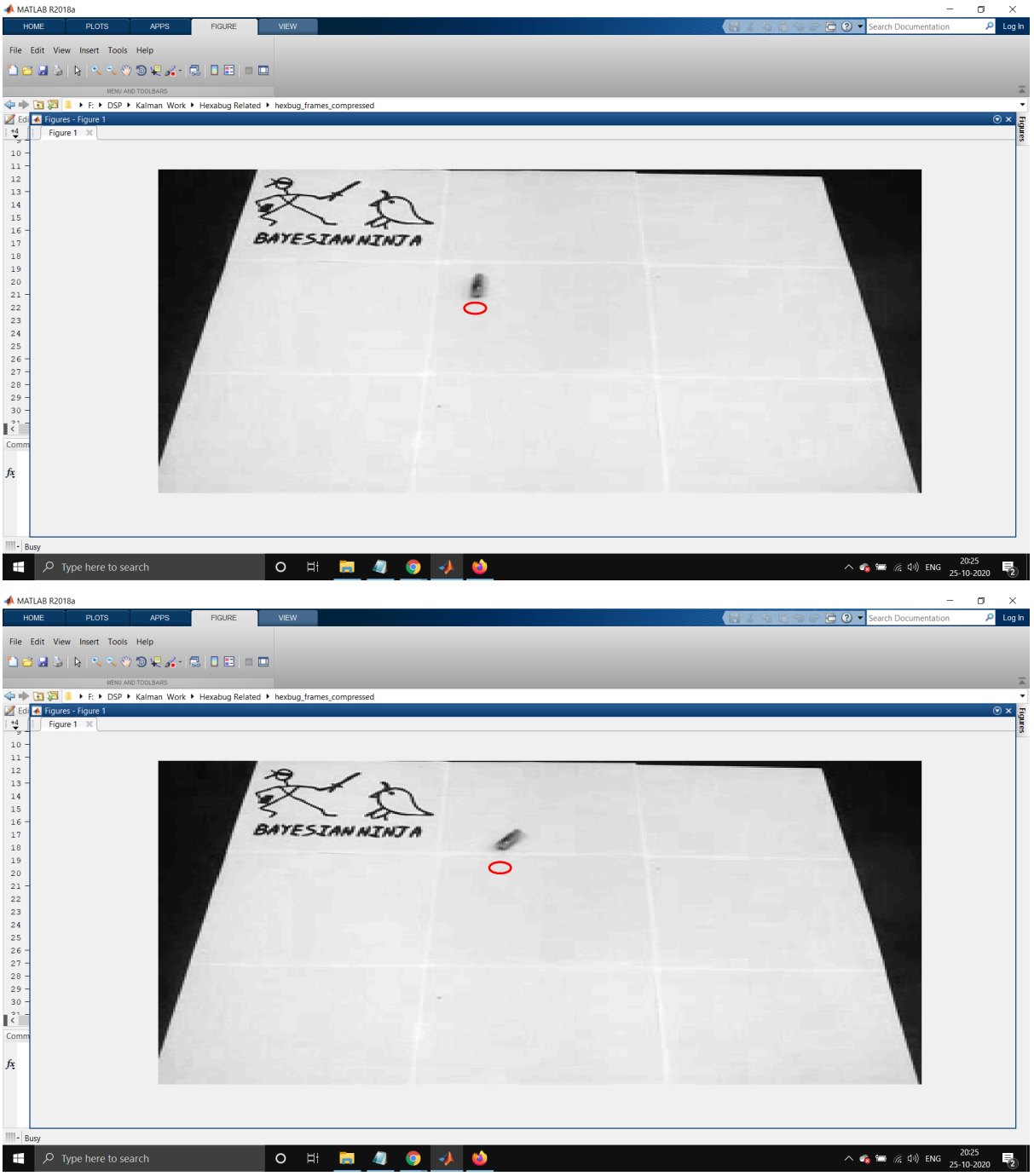



**Image 9 & 10: Images of only the Red circle/Kalman Estimation following the moving body**

**Observations for tweaks made:**

1. Increase in the process/system noise
    a. **System Observation**: Kalman Estimation won't be that great. Because P = Ex gets affected and it affects the Kalamna gain and hence the whole deviation from the original estimation.


**Image 11**

2. Decrease in the Measurement noise (Ez)
    a. **System Observation**: Wrt above case, the Kalman gain deviates from the standard again and we get a worst estimation as shown below


**Image 12**

3. Increase in Acceleration (Control Input): Value increased from 0.005 to 0.05
    a. **System Observation**: Tracking is lost very drastically - Because a great value is being added to Equation No.3, instead of the recursive corrected estimate **(Q_estimate = A * Q_estimate + B * u;)**


**Image 13**

# Next Steps to Explore:

- Find the velocity Xv & Yv
- Extended Kalman filters for nonlinear system
- Try to understand the Math derivation of Measurement update part - applying partial differential equations.

# Other Explored Applications:

**Examples of KF:**

**1.** For example, radio communication signals are corrupted with noise. A good filtering algorithm can remove the noise from electromagnetic signals while retaining the useful information.

**2.** Another example is power supply voltages. Uninterruptible power supplies are devices that filter line voltages in order to smooth out undesirable fluctuations that might otherwise shorten the lifespan of electrical devices such as computers and printers.

Hungarian Filters

**3**. Kalman Filters user in Stock prediction - Quantopian Website link

# References:

1. Student dave Website
2. Coursera / octave sensor based Kalman Filter Implementation

3. Dan SImon Kalman Filters paper
4. Kalman Filter for Dummies

# Glossary:

1. Gaussian signal - where most of the signal is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean
2.

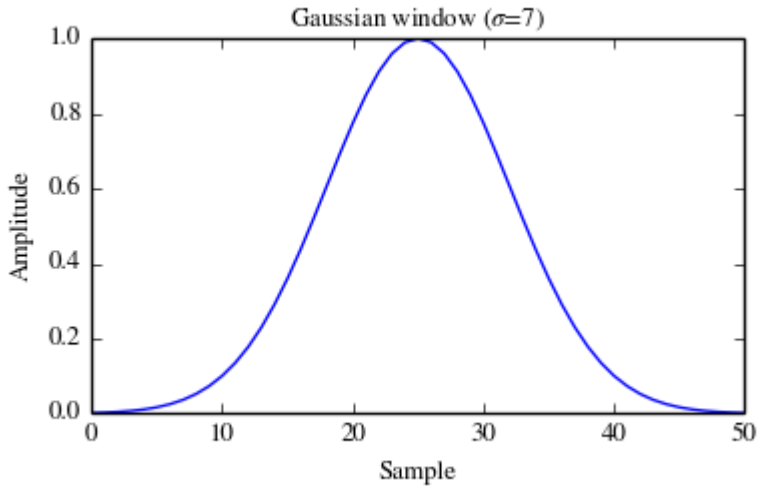

Gaussian window (σ=7)
1.0
0.8
0.6
0.4
0.2
0.0
Amplitude
0
10
20
30
40
50
Sample

**Image 14**