ECSE 6550 COMPUTER VISION
PROJECT 5

Andrew Cunningham

May 12, 2016

1 Introduction

Tracking objects of interest in a sequence of images can be a challenging problem. This is especially true in the case of real-time object tracking or tracking an object that get occluded at some point. There are a variety of techniques that can be applied to approach this problem. One such technique is to use a Kalman filter in conjunction with some object detector to track an object of interest

2 Theory

2.1 Kalman Filter

The Kalman filter is an estimation algorithm that incorporates state dynamics and noisy measurements to produce an estimation of the system's state. It is an appealing estimator to use because it is both recursive and optimal (when certain conditions are met). To use the Kalman filter, it is necessary to define a state vector and a system model.

$$s_t = \text{State Vector} \tag{1}$$

$$s_{t+1} = \Phi s_t + w_t \tag{2}$$

Where Φ is the state transition matrix and w_t is some Gaussian distributed process noise. Once the state and state evolution are defined, Kalman filtering is performed in two steps. The first step is prediction and the second step is the update step.

2.1.1 Prediction

In the prediction step, information from the previous time step is used to predict the next time step's state.

$$\bar{s_{t+1}} = \Phi s_t + w_t \tag{3}$$

In addition, the confidence of this estimate is calculated and encapsulated in a covariance matrix which is also calculated:

$$\Sigma_{t+1}^{-} = \Phi \Sigma_t \Phi^t + Q \tag{4}$$

Symbol	Meaning
s_{t+1}^{-}	Predicted next state
Σ_t	Estimate Covariance
Q	System Error Covariance

Table 1: Meaning of symbols in Equations 3 and 4

2.1.2 Update

After the next state is estimated using information from the previous time step, a measurement is introduced and incorporated into another estimate. The measurement, z_t is described by the following model:

$$z_t = H s_t + \epsilon_t \tag{5}$$

This new measurement is incoporated into a new state estimate by:

$$s_{t+1} = s_{t+1}^{-} + k_{t+1}(z_{t+1} - H s_{t+1}^{-}) \tag{6}$$

where k_{t+1} is the Kalman gain and is calculated by:

$$k_{t+1} = (\Sigma_{t+1}^{-} H^t)(H \Sigma_{t+1}^{-} H^t + R)^{-1} \tag{7}$$

and lastly, the covariance estimate is updated with:

$$\Sigma_{t+1} = (I - k_{t+1} H) \Sigma_{t+1}^{-} \tag{8}$$

Symbol	Meaning
H	State to measurement map
k_{t+1}	Kalman Gain
ϵ_t	Measurement noise
R	Measurement Error Covariance

Table 2: Meaning of symbols in Equations 5, 6,7 and 8

2.1.3 Initializing the Kalman Filter

In order to use the Kalman filter, it is necessary to assign values to H,Q,R,Σ_0, and s_0. The values of H,Q and R are constant and will remain the same throughout the operation of the Kalman filter. s_0 is simply the starting state of the system and Σ_0 is the starting covariance. Σ_0 is typically initialized to large values due to the fact that it will converge to a stable value later on in operation.

2.2 Kalman filter for image tracking

In the context of tracking objects in images, the state can be formed to be:

$$s_t = [x_t, y_t, v_{x,t}, v_{y,t}]^t. \tag{9}$$

where x_t,y_t is the position of the pixel at time t and $v_{x,t}$, $v_{y,t}$ is the velocity of the pixel at time t. The evolution of the state can then be formed to be:

$$x_{t+1} = x_t + v_{x,t} + w_t \tag{10}$$
$$y_{t+1} = y_t + v_{y,t} + w_t \tag{11}$$
$$v_{x,t+1} = v_{x,t} + w_t \tag{12}$$
$$v_{y,t+1} = v_{y,t} + w_t \tag{13}$$

where w_t is some gaussian distributed noise. The same model can be represented in state space by:

$$s_{t+1} = \Phi s_t + w_t \text{ where } \Phi = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

To initialize the Kalman filter for image tracking where the measurement is the pixel coordinates of the object of interest, Q and R are initialized to:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{15}$$

the values of Q and R were determined experimentally. They could be computed analytically by doing noise analysis on the system in question. Lastly, the covariance matrix was initialized to be:

$$\Sigma_0 = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \tag{16}$$

which was just chosen to be an arbitrarily high value.

2.3 Object Location Measurement

In order to apply Kalman filtering to object tracking in an image, it is necessary to somehow detect or measure the location of the object. In this project, the Sum of Squared Differences and Haar classifier methods are used.

2.3.1 Sum of Squared Differences

The sum of square differences is a distance metric that can be applied to aid in matching pixels in two images. In the context of pixel matching, the sum of squared differences is applied to a two dimensional grid of values:

$$\sum_{(i,j)\in W} (I_1(i,j) - I_2(x+i, y+j))^2 \tag{17}$$

Where W is the grid, I_1 is the image segment to find, I_2 is the image candidate and x and y is the pixel around which the grid in I_2 is centered. The sum of square differences can be applied to match pixels in two images by finding an x and y in the second image that minimizes the sum of squared distances. When used in conjunction with Kalman filtering, the two images that are compared are offset by a timestep. Thus, to measure the location of an object in the next time step, the location of the object in the previous timestep is used as the target for SSD.

2.3.2 Haar Cascade

Another approach to detecting the position of an object in an image is the use of classifiers. A classifier can be applied to different regions of the image to judge whether or not an object of interest is in the region. One such classifier is the Haar cascade classifier. The Haar cascade classifier works by extracting features from the image regions four pixels at a time.

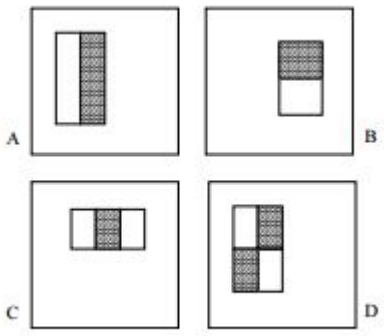

Figure 1: Haar features taken from [1]

The classifier then extracts features with these four windows over the entire region in question to classify it. When applied to face detection, the algorithm is setup as a cascading classifier in that it will check to see if a region is a face with some of the features. If it is not, discard the region and continue the search [1]. The net result of this algorithm is the capability to classify whether or not a region has an object of interest in it and to give the location of the image in pixels.

3 Problem Statement and Approach

3.1 Problem Statement

"Implement the Kalman filter (KF) to track a face over frames. You will be given a sequence of images containing a moving face and the initial face positions and their bounding boxes in the first two frames. You will then implement a Kalman filter to track the face, i.e., determine the face bound box position from frame to frame until the last frame."

3.2 Approach

The Kalman filter was implemented and used in conjunction with both the SSD detector and the Haar cascade detector. The SSD approach proved to be extremely computational costly and unreliable while the Haar cascade approach was computationaly quick and robust. To improve the computational time of the SSD measurement, suggestions from the haar cascade were used as search centers. This approach allowed for smaller search boxes and greatly reduced the computation time.

4 Code

The code for this project is organized into two main parts:

a. SSD implementation

b. Haar Cascade implementation

Where each part has slightly modified Kalman filters.

4.1 SSD

Listing 1: mainSSD.m

```
face_tracking_1 = [206, 303, 172, 172];
face_tracking_2 = [213, 303, 172, 172];

%% APPLY KALMAN FILTERING

% INITIALIZE STUFF
stateVector = zeros(100,4);
stateVector(1,:) = [213,303,7,0]; % [X,Y,Vx,Vy]
cov = zeros(4,4,100);
cov(:,:,1) = eye(4,4)*100;
z = zeros(100,2);
phi = [1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1]; % State transition matrix
R = eye(2,2)*20; % Large R because we are using SSD
Q = eye(4)*10; Q(4,4) = 4; Q(3,3) = 4;
H = eye(2,4);
% apply kalman filtering to all of the images
originalImage = rgb2gray(imread(strcat('data/face_tracking_','3','.png')));
suggestions =csvread('results.txt');

% suggestions are imported from the haar classifier.  It chops compuatation
% time down by a ton to search in a smaller region around

% Use a box centered on this pixel for ssd
originalPixel = [222 303]
for i=1:96
    % handle indices to avoid confusion
    imageIndex = i+2;
    kalmanIndex = i+1;

    % PREDICT
    pStateVector = phi*(stateVector(kalmanIndex-1,:))';
    covTp1 = phi*cov(kalmanIndex-1)*phi' + Q;

    % UPDATE
    pixelCoords = [stateVector(kalmanIndex-1,1), stateVector(kalmanIndex-1,2)];
    currentImage = rgb2gray(imread(strcat('data/face_tracking_',int2str(imageIndex),'.png')));
    z(kalmanIndex,:) = ssdCorrelate(originalImage, currentImage,originalPixel,suggestions(i,:),5)
    k = (covTp1*H')*inv(H*covTp1*H'+R);
    stateVector(kalmanIndex,:) = pStateVector + k*(z(kalmanIndex,:)'-H*pStateVector);
    cov(:,:,kalmanIndex)=(eye(4,4)-k*H)*covTp1;
end
```

```
%% GENERATE MOVIE
f = []
for i=1:96 %start from 3rd image and go from there
    imageIndex = i+2;
    kalmanIndex = i+1;
    pixels=uint16(stateVector(kalmanIndex,1:2));
    image=imread(strcat('data/face_tracking_',int2str(imageIndex),'.png'));
    image = putXonImage(image,pixels,[50,205,50]);
    image = putXonImage(image,z(kalmanIndex,:),[0,0,0]);
    f=[f;im2frame(image)];

    mov(96-i+1)=im2frame(image);
end
movie2gif(mov, 'ssdKalman.gif', 'LoopCount', 0, 'DelayTime', 0)

%% PLOT TRACE OF COVARIANCE MATRIX
x = []; y =[];
for i=1:size(cov,3)
    x = [x i];
    y = [y trace(cov(:,:,i))];
end
plot(x,y)
```

The ssdCorrelate.m function attempts to match a box specified in the original image in the new image.

Listing 2: ssdCorrelate.m

```
function [ state ] = ssdCorrelate( rectImageL, rectImageR, leftPixelsIn, offset)
% go through each leftPixel
rightOriginalPixels=[];
bestBox = [];
for k=1:size(leftPixelsIn)
    c0 = round(leftPixelsIn(k,1));
    r0 = round(leftPixelsIn(k,2));
    boxToSearchFor = rectImageL(r0-offset:r0+offset,c0-offset:c0+offset);

    % scan along the 5 rows surrounding the row of the last pixel
    minSSD = 1000000;
    bestCandidate = [-1; -1];
    for i=-30:1:30 % search along 11 different rows
        rowToSearch = r0 + i;
        for j=-20:20
            columnToSearch = c0+j;
            currentBox = rectImageR(rowToSearch-offset:rowToSearch...
                +offset,columnToSearch-offset:columnToSearch+offset);
            ssd = sumsqrd(boxToSearchFor,currentBox);

            if ssd < minSSD
                minSSD = ssd;
                bestCandidate = [columnToSearch, rowToSearch];
            end
        end
    end
end
```

```
    state = [bestCandidate(1) bestCandidate(2)];
    end
```

4.2 Haar Cascade

The bulk of the Haar Cascade detector was imported from OpenCV. The work that I did here was simply calling the appropriate functions. Unlike the previous two segments of code, this part was implemented in python.

Listing 3: mainHaar.m

```
face_tracking_1 = [206, 303, 172, 172];
face_tracking_2 = [213, 303, 172, 172];

%% APPLY KALMAN FILTERING
stateVector = zeros(100,4);
stateVector(1,:) = [213,303,7,0]; % [X,Y,Vx,Vy]
cov = zeros(4,4,100);
cov(:,:,1) = eye(4,4)*100;

% IMPORT OBSERVATIONS FROM HAAR CLASSIFIER
z = csvread('results.txt');

phi = [1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1]; % State transition matrix
R = eye(2,2);
Q = eye(4); Q(4,4) = 4; Q(3,3) = 4;
H = eye(2,4);
% apply kalman filtering to all of the images
for i=1:96
    % handle indices
    imageIndex = i+2;
    kalmanIndex = i+1;

    % predict
    pStateVector = phi*(stateVector(kalmanIndex-1,:))';
    covTp1 = phi*cov(kalmanIndex-1)*phi' + Q;

    % update
    pixelCoords = [stateVector(kalmanIndex-1,1), stateVector(kalmanIndex-1,2)];
    prevImage = rgb2gray(imread(strcat('data/face_tracking_',int2str(imageIndex-1),'.png')));
    currentImage = rgb2gray(imread(strcat('data/face_tracking_',int2str(imageIndex),'.png')));
    k = (covTp1*H')*inv(H*covTp1*H'+R);
    stateVector(kalmanIndex,:) = pStateVector + k*(z(kalmanIndex,:)'-H*pStateVector);
    cov(:,:,kalmanIndex)=(eye(4,4)-k*H)*covTp1;
end

%% GENERATE MOVIE
f = [];
for i=1:96 %start from 3rd image and go from there
    imageIndex = i+2;
    kalmanIndex = i+1;
    pixels=uint16(stateVector(kalmanIndex,1:2));
    image=imread(strcat('data/face_tracking_',int2str(imageIndex),'.png'));
    image = putXonImage(image,pixels,[50,205,50]);
```

```
    image = putXonImage(image,z(i,:),[0,0,0]);
    f=[f;im2frame(image)];

    mov(96-i+1)=im2frame(image);
end
movie2gif(mov, 'haarKalman.gif', 'LoopCount', 0, 'DelayTime', 0)

%% PLOT TRACE OF COVARIANCE MATRIX
x = []; y =[];
for i=1:size(cov,3)
    x = [x i];
    y = [y trace(cov(:,:,i))];
end
plot(x,y)
```

Listing 4: face_detect.py

```
import cv2
import sys

# Get user supplied values
imagePath = 'face_tracking_1.png'
cascPath = 'haarcascade_frontalface_default.xml'

# Create the haar cascade
faceCascade = cv2.CascadeClassifier(cascPath)

# Read the image

f=open('results.txt', 'w')

for i in range(3,100):
    #print i
    imagePath = 'face_tracking_'+str(i)+'.png'
    image = cv2.imread(imagePath)
    image = cv2.imread(imagePath)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Detect faces in the image
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=12,
        minSize=(100, 100),
        flags = cv2.CASCADE_SCALE_IMAGE
    )

    print "Found {0} faces!".format(len(faces))

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        message = str(x+w/2) + ',' + str(y+h/2) + '\n'
        #print message
        f.write(message)

f.write('\n DONE2')
```

5 Results and Analysis

The results of the tracking can be seen in ssdKalman.gif and haarKalman.gif. In general, they both performed pretty well but the Haar-Kalman approach performed better. The reason the haar-kalman approach was able to perform better is that the more reliable measurements from the haar classifier allowed for a smaller R and Q. Consequently, the measurements were given more weight and (because the measurements were really good) the face was tracked very closely. The SSD-Kalman approach followed the face pretty well, but had some significant lag and drift as the face moved around. This result also makes sense because R and Q were weighed less heavily so system dynamics movement played a larger role in moving the image around. For both cases, the trace of the covariance matrix was plotted.

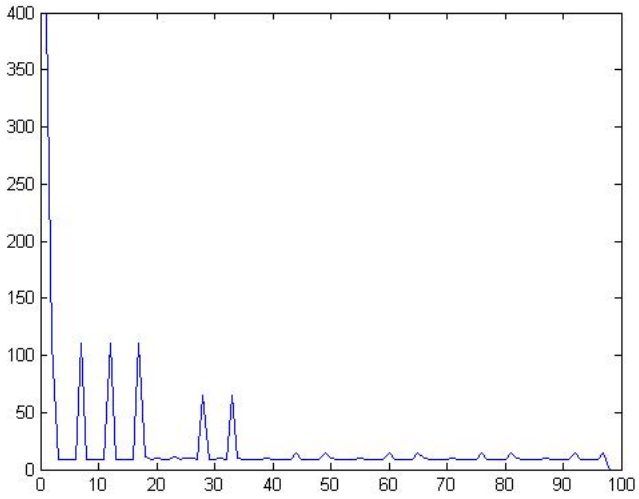

Figure 2: The results of the trace in the Haar-Kalman approach

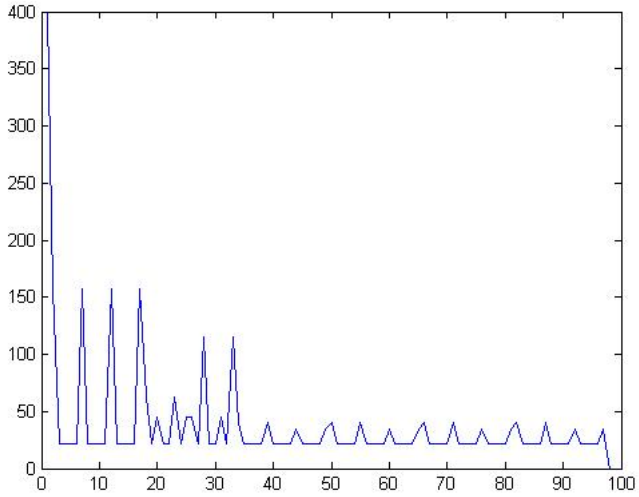

Figure 3: The results of the trace in the SSD-Kalman approach

As can be seen in Figures 5 and 5, the traces both dropped dramatically after the first couple of iterations. However, the Haar-Kalman trace dropped to a lower steady state level than the SSD-Kalman trace. This difference in trace values originates from the differing R and Q values.

6 Conclusion

This project explored image target tracking using the Kalman filter. Two different measurement techniques were explored to provide the Kalman filter with measurements. The resulting tracking performed pretty well with only small deviations from what appears to be correct. The Haar-Kalman tracker tracked slightly better than the SSD-Kalman tracker. In general target tracking applications, this may not always be the case. The classifier used in the Haar-Kalman approach was trained on forward facing faces and would be less likely to be able to find a face if the face changed orientation. In contrast, the SSD approach would be able to handle changes in orientation and even color/shape provided the changes were gradual. This improvement comes at the cost of computation time and the potential to lose a target over time.

References

[1] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001.*

CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, volume 1, pages I–511. IEEE, 2001.