Programmable Embedded Systems
Assignment 2

Submitted by
Pratyush Jaiswal
18EE35014

Design the android app to classify the motion of cellphones in the x-axis, y-axis, and z-axis direction, and circular motion using Kalman filter and SVM. Submit the codes, results, and a small report in a pdf (Demo will be taken for everyone later on).

Introduction

Support Vector machines can be defined as systems that use hypothesis space of a linear function in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. Support vector machine was initially popular with the NIPS community and now is an active part of machine learning research around the world.
SVM becomes famous when using pixel maps as input; it gives accuracy comparable to sophisticated neural networks with elaborated features in a handwriting recognition task. It is also being used for many applications, such as handwriting analysis, face analysis and so forth, especially for pattern classification and regression-based applications.
The foundations of Support Vector Machines (SVM) have been developed by Vapnik and gained popularity due to many promising features such as better empirical performance. The formulation uses the Structural Risk Minimization (SRM) principle, which has been shown to be superior, to the traditional Empirical Risk Minimization (ERM) principle, used by conventional neural networks. SRM minimizes an upper bound on the expected risk, whereas ERM minimizes the error on the training data. It is this difference that equips SVM with a greater ability to generalize, which is the goal in statistical learning. SVMs were developed to solve the classification problem, but recently they have been extended to solve regression problems.

Statistical Learning Theory

The statistical learning theory provides a framework for studying the problem of gaining knowledge, making predictions, making decisions from a set of data. In simple terms, it enables the choosing of the hyper plane space such a way that it closely represents the underlying function in the target space.

In statistical learning theory the problem of supervised learning is formulated as follows. We are given a set of training data $\{(\mathbf{x}_1,y_1)... (\mathbf{x}_l,y_l)\}$ in $R^n \times R$ sampled according to unknown probability distribution $P(\mathbf{x},y)$, and a loss function $V(y,f(\mathbf{x}))$ that measures the error, for a given **x**, f(**x**) is "predicted" instead of the actual value y. The problem consists in finding a function f that minimizes the expectation of the error on new data that is, finding a function f that minimizes the expected error: $\int V(y,f(\mathbf{x}))\, P(\mathbf{x},y)\, d\mathbf{x}\, dy$ [6] In statistical modeling we would choose a model from the hypothesis space, which is closest (with respect to some error measure) to the underlying function in the target space. More on statistical learning theory can be found on introduction to statistical learning theory [7].

Learning and Generalization

Early machine learning algorithms aimed to learn representations of simple functions. Hence, the goal of learning was to output a hypothesis that performed the correct classification of the training data and early learning algorithms were designed to find such an accurate fit to the data [8]. The ability of a hypothesis to correctly classify data not in the training set is known as its generalization. SVM performs better in term of not over generalization when the neural networks might end up over generalizing easily [11]. Another thing to observe is to find where to make the best trade-off in trading complexity with the number of epochs; the illustration brings to light more information about this.
The below illustration is made from the class notes.

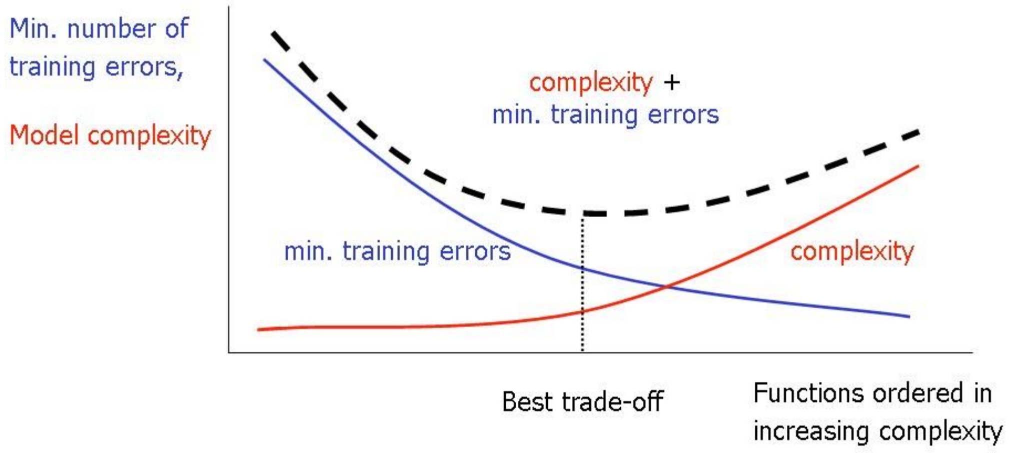


Figure 1: Number of Epochs Vs Complexity. [8][9][11]

Introduction to SVM: Why SVM?

Firstly working with neural networks for supervised and unsupervised learning showed good results while used for such learning applications. MLP's uses feed forward and recurrent networks. Multilayer perceptron (MLP) properties include universal approximation of continuous nonlinear functions and include learning with input-output patterns and also involve advanced network architectures with multiple inputs and outputs [10].

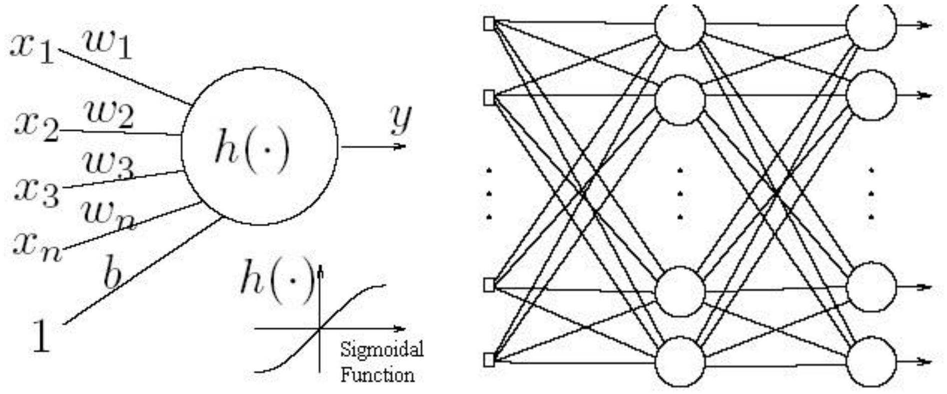


Figure 2: a] Simple Neural Network b]Multilayer Perceptron. [10][11]. These are simple visualizations just to have a overview as how neural network looks like.

There can be some issues noticed. Some of them are having many local minima and also finding how many neurons might be needed for a task is another issue which determines whether optimality of that NN is reached. Another thing to note is that even if the neural network solutions used tends to converge, this may not result in a unique solution [11]. Now let us look at another example where we plot the data and try to classify it and we see that there are many hyper planes which can classify it. But which one is better?

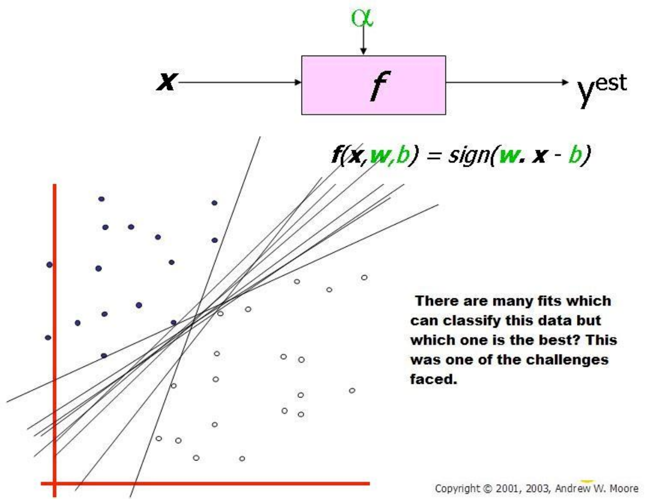


Figure 3: Here we see that there are many hyper planes which can be fit in to classify the data but which one is the best is the right or correct solution. The need for SVM arises. (Taken Andrew W. Moore 2003) [2]. Note the legend is not described as they are sample plotting to make understand the concepts involved.

From above illustration, there are many linear classifiers (hyper planes) that separate the data. However only one of these achieves maximum separation. The reason we need it is because if we use a hyper plane to classify, it might end up closer to one set of datasets compared to others and we do not want this to happen and thus we see that the concept of maximum margin classifier or hyper plane as an apparent solution. The next illustration gives the maximum margin classifier example which provides a solution to the above mentioned problem [8].

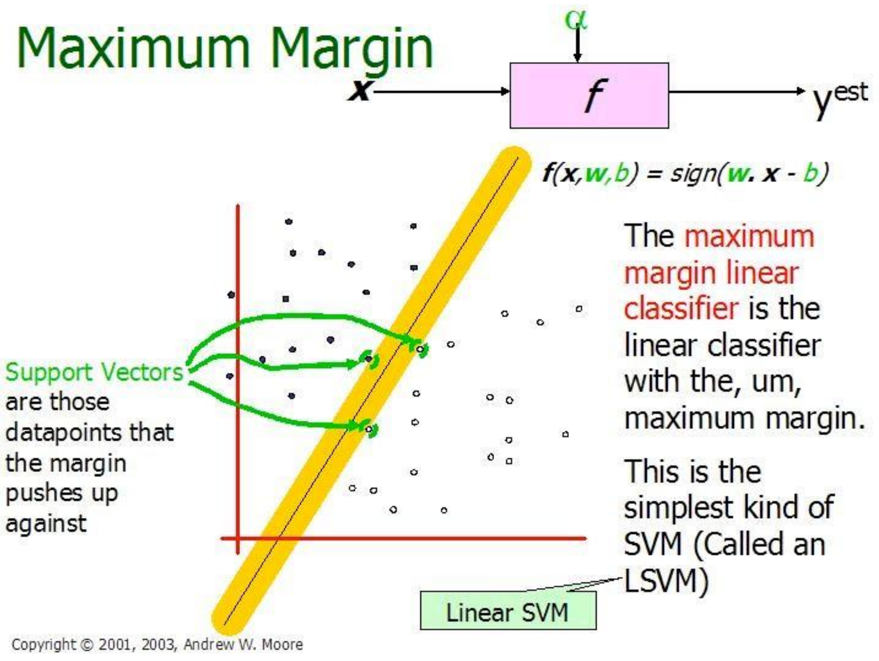

Maximum Margin
α
x
f
y^est
f(x,w,b) = sign(w. x - b)
The maximum margin linear classifier is the linear classifier with the, um, maximum margin.
This is the simplest kind of SVM (Called an LSVM)
Support Vectors are those datapoints that the margin pushes up against
Linear SVM
Copyright © 2001, 2003, Andrew W. Moore

Figure 4: Illustration of Linear SVM. (Taken from Andrew W. Moore slides 2003) [2]. Note the legend is not described as they are sample plotting to make understand the concepts involved.

Expression for Maximum margin is given as [4][8] (for more information visit [4]):

$$\text{margin} \equiv \underset{\mathbf{x}\in D}{\arg\min}\, d(\mathbf{x}) = \underset{\mathbf{x}\in D}{\arg\min} \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

The above illustration is the maximum linear classifier with the maximum range. In this context it is an example of a simple linear SVM classifier. Another interesting question is why maximum margin? There are some good explanations which include better empirical performance. Another reason is that even if we've made a small error in the location of the boundary this gives us least chance of causing a misclassification. The other advantage would be avoiding local minima and better classification. Now we try to express the SVM mathematically and for this tutorial we try to present a linear SVM. The goals of SVM are separating the data with hyper plane and extend this to non-linear boundaries using kernel trick [8] [11]. For calculating the SVM we see that the goal is to correctly classify all the data. For mathematical calculations we have,

[a] If Y_i= +1; $wx_i + b \geq 1$

[b] If Y_i= -1; $wx_i + b \leq 1$

[c] For all i; $y_i (w_i + b) \geq 1$

In this equation x is a vector point and w is weight and is also a vector. So to separate the data [a] should always be greater than zero. Among all possible hyper planes, SVM selects the one where the distance of hyper plane is as large as possible. If the training data is good and every test vector is located in radius r from training vector. Now if the chosen hyper plane is located at the farthest possible from the data [12]. This desired hyper plane which maximizes the margin also bisects the lines between closest points on convex hull of the two datasets. Thus we have [a], [b] & [c].

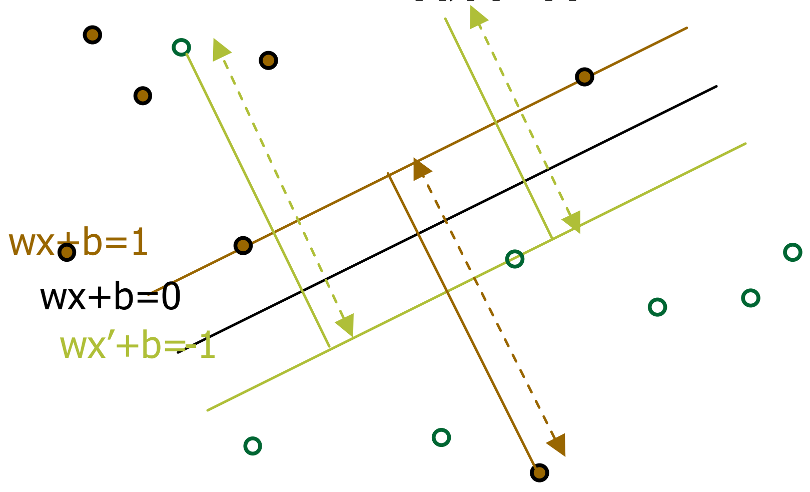


Figure 5: Representation of Hyper planes. [9]

Distance of closest point on hyperplane to origin can be found by maximizing the x as x is on the hyper plane. Similarly for the other side points we have a similar scenario. Thus solving and subtracting the two distances we get the summed distance from the separating hyperplane to nearest points. Maximum Margin = M = 2 / ||w||

Now maximizing the margin is same as minimum [8]. Now we have a quadratic optimization problem and we need to solve for w and b. To solve this we need to optimize the quadratic function with linear constraints. The solution involves constructing a dual problem and where a Langlier's multiplier α_i is associated. We need to find w and b such that Φ (w) =½ |w'||w| is minimized; And

for all $\{(x_i, y_i)\}$: $y_i (w * x_i + b) \geq 1$.

Now solving: we get that $w = \Sigma\alpha_i * x_i$; $b = y_k - w * x_k$ for any x_k such that $\alpha k \neq 0$

Now the classifying function will have the following form: $f(x) = \Sigma\alpha_i y_i x_i * x + b$

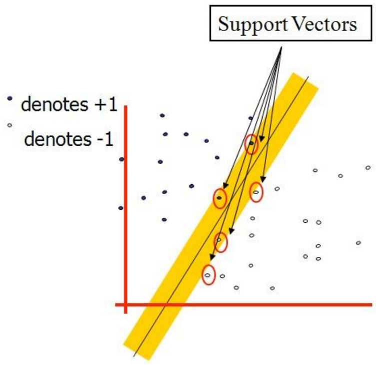


Figure 6: Representation of Support Vectors (Copyright © 2003, Andrew W. Moore)[2]

SVM Representation

In this we present the QP formulation for SVM classification [4][8][12][13]. This is a simple representation only.

SV classification:

$$\min_{f,\xi_i} \|f\|_K^2 + C\sum_{i=1}^{l}\xi_i \qquad y_i f(\mathbf{x}_i) \geq 1 - \xi_i, \text{ for all } i \quad \xi_i \geq 0$$

SVM classification, Dual formulation:

$$\min_{\alpha_i} \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \qquad 0 \leq \alpha_i \leq C, \text{ for all } i; \qquad \sum_{i=1}^{l}\alpha_i y_i = 0$$

Variables ξ_i are called slack variables and they measure the error made at point $(\mathbf{x}_i, y_i)$. Training SVM becomes quite challenging when the number of training points is large. A number of methods for fast SVM training have been proposed [4][8][13].

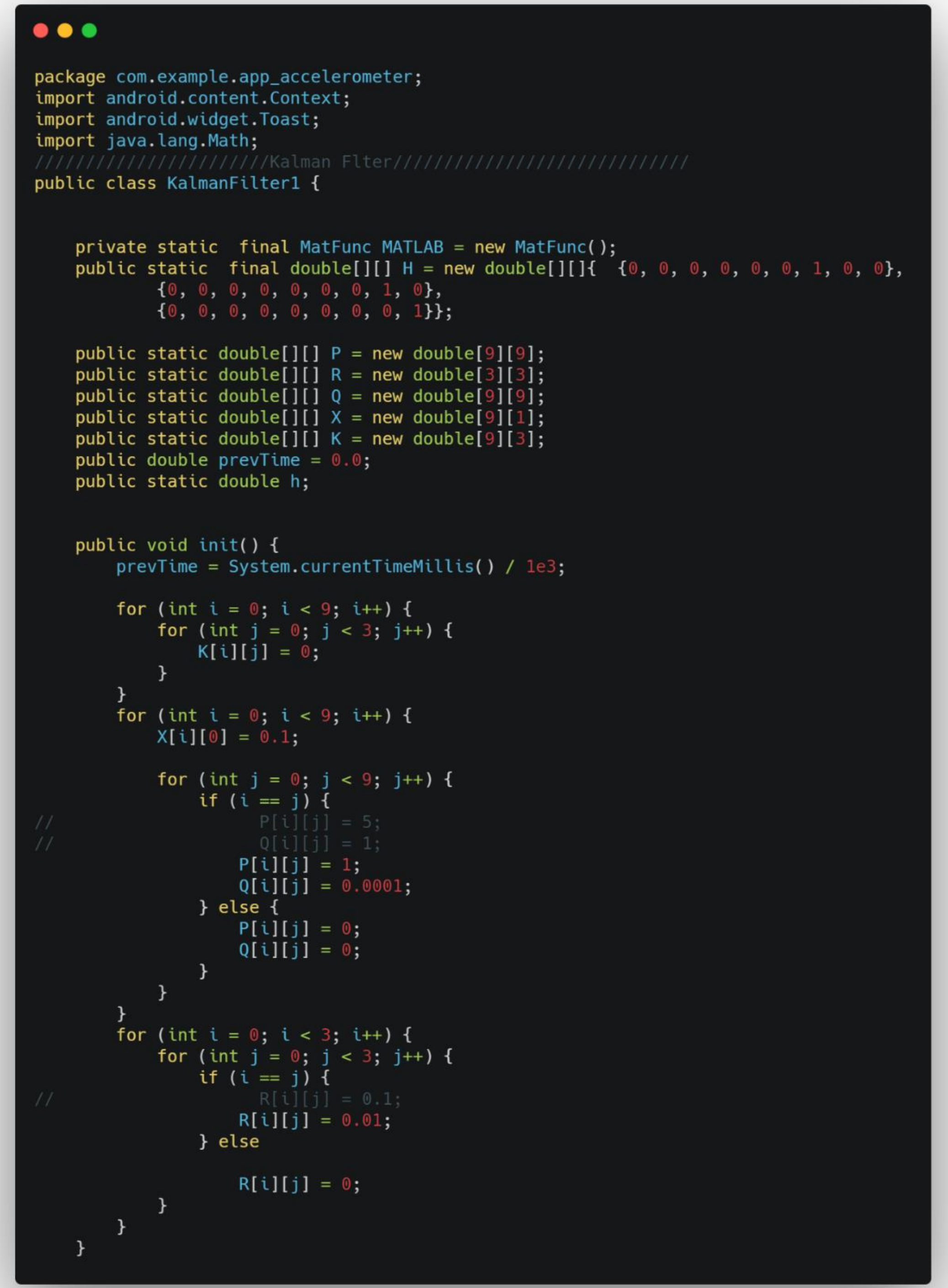
Soft Margin Classifier

In real world problem it is not likely to get an exactly separate line dividing the data within the space. And we might have a curved decision boundary. We might have a hyperplane which might exactly separate the data but this may not be desirable if the data has noise in it. It is better for the smooth boundary to ignore few data points than be curved or go in loops, around the outliers. This is handled in a different way; here we hear the term slack variables being introduced. Now we have, $y_i(w'x + b) \geq 1 - S_k$ [4] [12]. This allows a point to be a small distance S_k on the wrong side of the hyper plane without violating the constraint. Now we might end up having huge slack variables which allow any line to separate the data, thus in such scenarios we have the Lagrangian variable introduced which penalizes the large slacks.

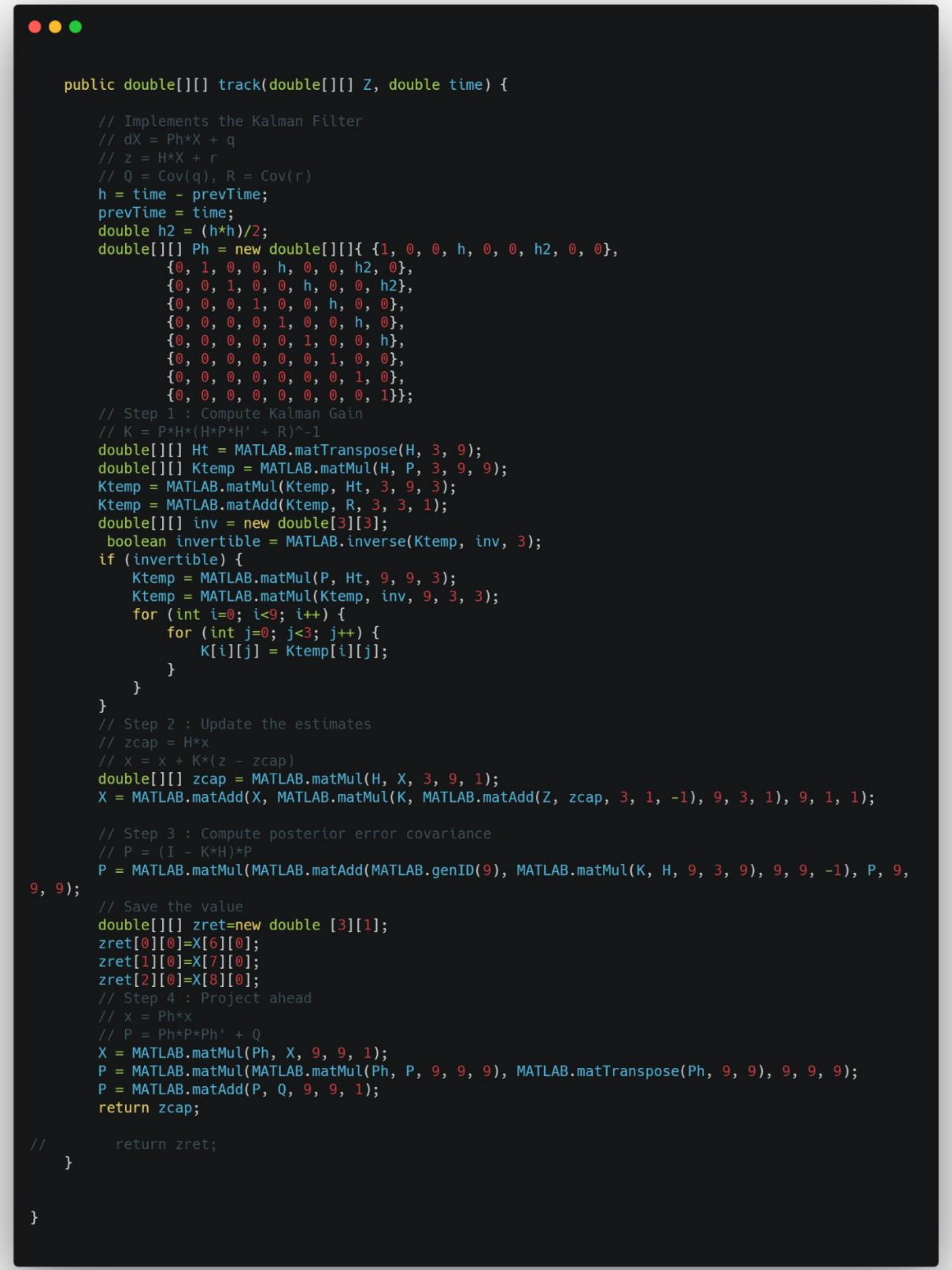
$$\min L = \tfrac{1}{2} w'w - \sum \lambda_k (y_k (w'x_k + b) + s_k - 1) + \alpha \sum s_k$$

Where reducing α allows more data to lie on the wrong side of hyper plane and would be treated as outliers which give smoother decision boundary [12].

Kalman Filter Code

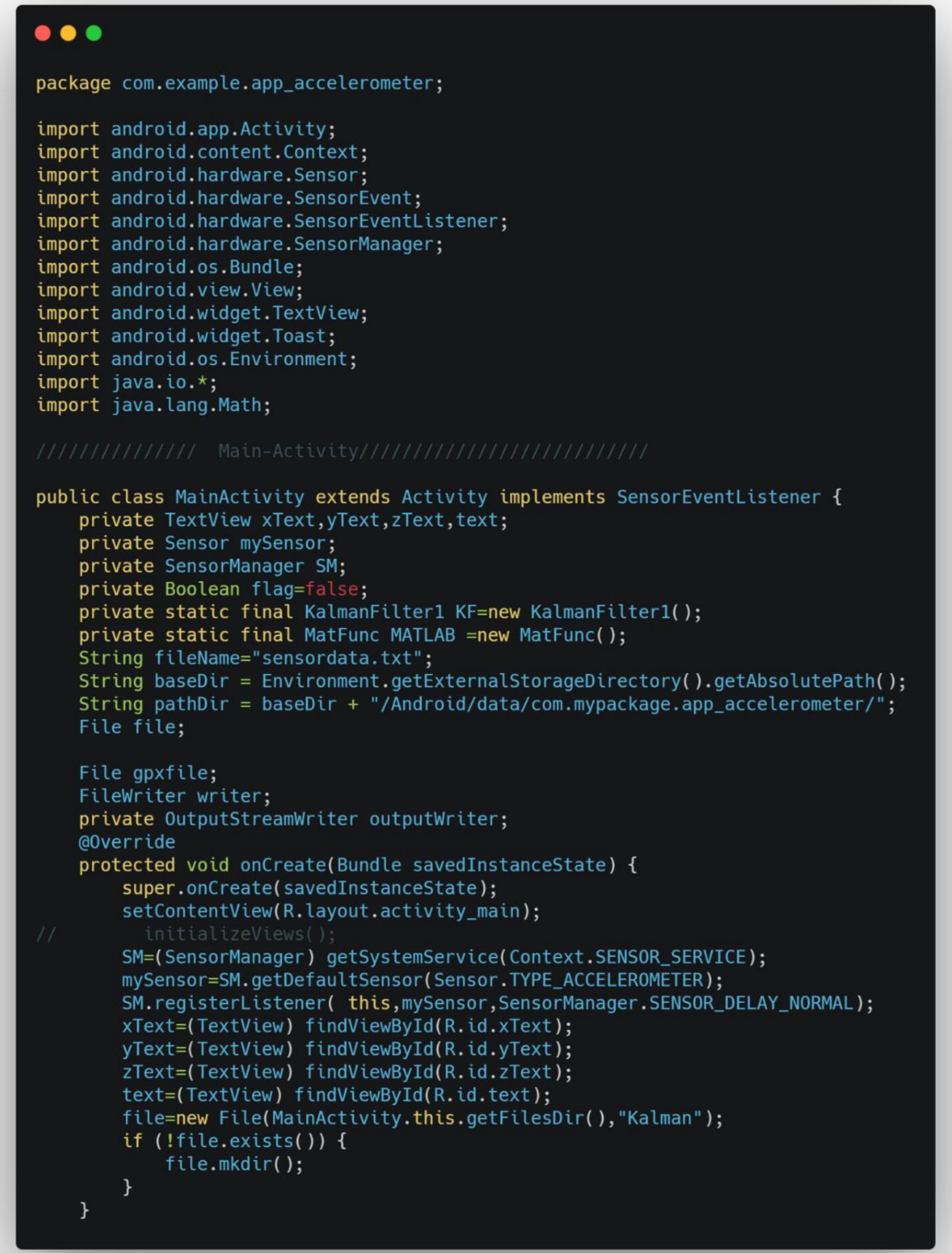


```java
package com.example.app_accelerometer;
import android.content.Context;
import android.widget.Toast;
import java.lang.Math;
/////////////////////////Kalman Flter/////////////////////////////
public class KalmanFilter1 {


    private static  final MatFunc MATLAB = new MatFunc();
    public static  final double[][] H = new double[][]{  {0, 0, 0, 0, 0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 1}};

    public static double[][] P = new double[9][9];
    public static double[][] R = new double[3][3];
    public static double[][] Q = new double[9][9];
    public static double[][] X = new double[9][1];
    public static double[][] K = new double[9][3];
    public double prevTime = 0.0;
    public static double h;


    public void init() {
        prevTime = System.currentTimeMillis() / 1e3;

        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 3; j++) {
                K[i][j] = 0;
            }
        }
        for (int i = 0; i < 9; i++) {
            X[i][0] = 0.1;

            for (int j = 0; j < 9; j++) {
                if (i == j) {
//                    P[i][j] = 5;
//                    Q[i][j] = 1;
                    P[i][j] = 1;
                    Q[i][j] = 0.0001;
                } else {
                    P[i][j] = 0;
                    Q[i][j] = 0;
                }
            }
        }
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (i == j) {
//                    R[i][j] = 0.1;
                    R[i][j] = 0.01;
                } else

                    R[i][j] = 0;
            }
        }
    }
```

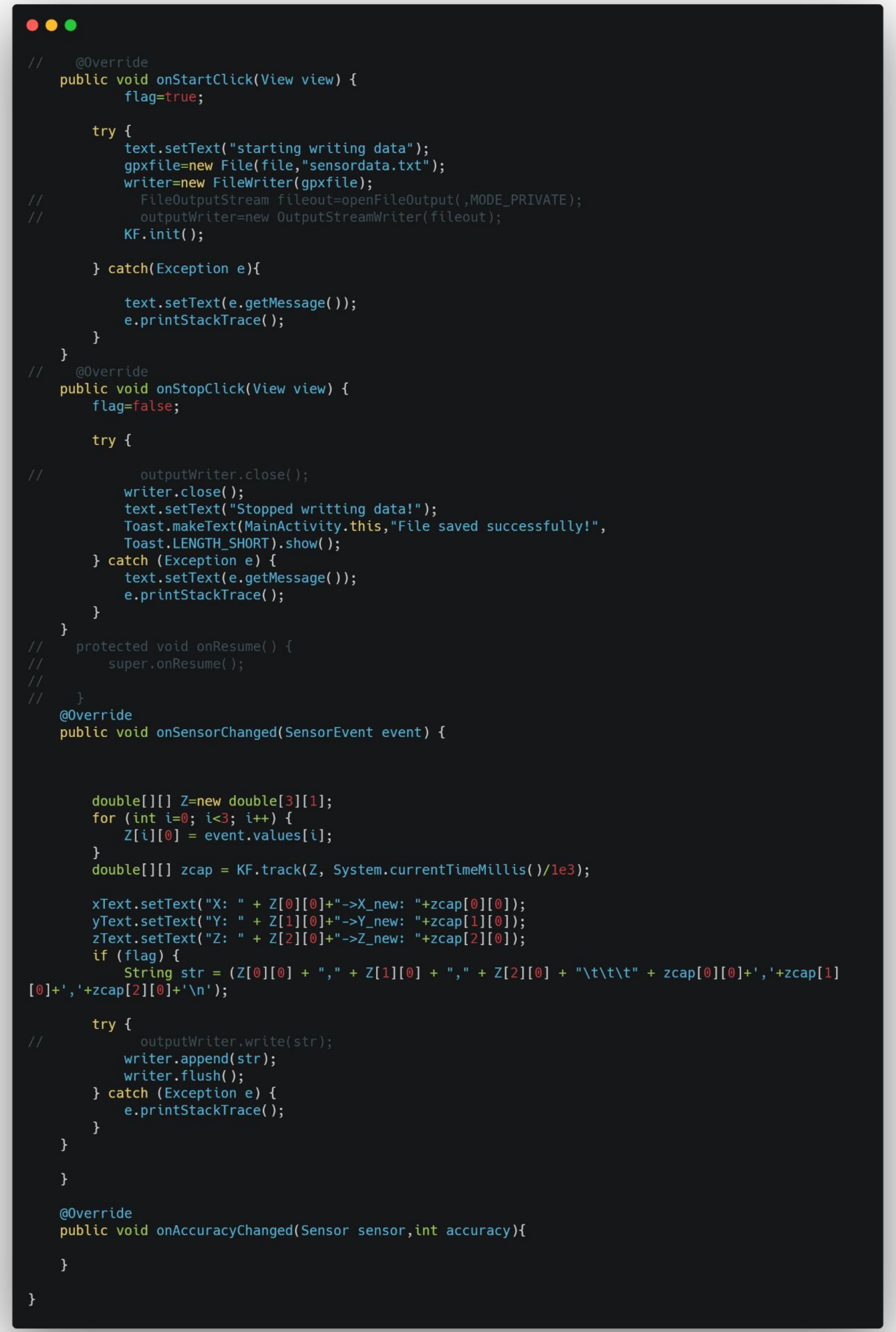


```java
    public double[][] track(double[][] Z, double time) {

        // Implements the Kalman Filter
        // dX = Ph*X + q
        // z = H*X + r
        // Q = Cov(q), R = Cov(r)
        h = time - prevTime;
        prevTime = time;
        double h2 = (h*h)/2;
        double[][] Ph = new double[][]{ {1, 0, 0, h, 0, 0, h2, 0, 0},
                {0, 1, 0, 0, h, 0, 0, h2, 0},
                {0, 0, 1, 0, 0, h, 0, 0, h2},
                {0, 0, 0, 1, 0, 0, h, 0, 0},
                {0, 0, 0, 0, 1, 0, 0, h, 0},
                {0, 0, 0, 0, 0, 1, 0, 0, h},
                {0, 0, 0, 0, 0, 0, 1, 0, 0},
                {0, 0, 0, 0, 0, 0, 0, 1, 0},
                {0, 0, 0, 0, 0, 0, 0, 0, 1}};
        // Step 1 : Compute Kalman Gain
        // K = P*H*(H*P*H' + R)^-1
        double[][] Ht = MATLAB.matTranspose(H, 3, 9);
        double[][] Ktemp = MATLAB.matMul(H, P, 3, 9, 9);
        Ktemp = MATLAB.matMul(Ktemp, Ht, 3, 9, 3);
        Ktemp = MATLAB.matAdd(Ktemp, R, 3, 3, 1);
        double[][] inv = new double[3][3];
         boolean invertible = MATLAB.inverse(Ktemp, inv, 3);
        if (invertible) {
            Ktemp = MATLAB.matMul(P, Ht, 9, 9, 3);
            Ktemp = MATLAB.matMul(Ktemp, inv, 9, 3, 3);
            for (int i=0; i<9; i++) {
                for (int j=0; j<3; j++) {
                    K[i][j] = Ktemp[i][j];
                }
            }
        }
        // Step 2 : Update the estimates
        // zcap = H*x
        // x = x + K*(z - zcap)
        double[][] zcap = MATLAB.matMul(H, X, 3, 9, 1);
        X = MATLAB.matAdd(X, MATLAB.matMul(K, MATLAB.matAdd(Z, zcap, 3, 1, -1), 9, 3, 1), 9, 1, 1);

        // Step 3 : Compute posterior error covariance
        // P = (I - K*H)*P
        P = MATLAB.matMul(MATLAB.matAdd(MATLAB.genID(9), MATLAB.matMul(K, H, 9, 3, 9), 9, 9, -1), P, 9,
9, 9);
        // Save the value
        double[][] zret=new double [3][1];
        zret[0][0]=X[6][0];
        zret[1][0]=X[7][0];
        zret[2][0]=X[8][0];
        // Step 4 : Project ahead
        // x = Ph*x
        // P = Ph*P*Ph' + Q
        X = MATLAB.matMul(Ph, X, 9, 9, 1);
        P = MATLAB.matMul(MATLAB.matMul(Ph, P, 9, 9, 9), MATLAB.matTranspose(Ph, 9, 9), 9, 9, 9);
        P = MATLAB.matAdd(P, Q, 9, 9, 1);
        return zcap;

//          return zret;
    }


}
```

MainActivity Code



```java
package com.example.app_accelerometer;

import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
import android.os.Environment;
import java.io.*;
import java.lang.Math;

///////////////  Main-Activity//////////////////////////

public class MainActivity extends Activity implements SensorEventListener {
    private TextView xText,yText,zText,text;
    private Sensor mySensor;
    private SensorManager SM;
    private Boolean flag=false;
    private static final KalmanFilter1 KF=new KalmanFilter1();
    private static final MatFunc MATLAB =new MatFunc();
    String fileName="sensordata.txt";
    String baseDir = Environment.getExternalStorageDirectory().getAbsolutePath();
    String pathDir = baseDir + "/Android/data/com.mypackage.app_accelerometer/";
    File file;

    File gpxfile;
    FileWriter writer;
    private OutputStreamWriter outputWriter;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
//        initializeViews();
        SM=(SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mySensor=SM.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        SM.registerListener( this,mySensor,SensorManager.SENSOR_DELAY_NORMAL);
        xText=(TextView) findViewById(R.id.xText);
        yText=(TextView) findViewById(R.id.yText);
        zText=(TextView) findViewById(R.id.zText);
        text=(TextView) findViewById(R.id.text);
        file=new File(MainActivity.this.getFilesDir(),"Kalman");
        if (!file.exists()) {
            file.mkdir();
        }
    }
```



```
//    @Override
    public void onStartClick(View view) {
            flag=true;

        try {
            text.setText("starting writing data");
            gpxfile=new File(file,"sensordata.txt");
            writer=new FileWriter(gpxfile);
//            FileOutputStream fileout=openFileOutput(,MODE_PRIVATE);
//            outputWriter=new OutputStreamWriter(fileout);
            KF.init();

        } catch(Exception e){

            text.setText(e.getMessage());
            e.printStackTrace();
        }
    }
//    @Override
    public void onStopClick(View view) {
        flag=false;

        try {

//            outputWriter.close();
            writer.close();
            text.setText("Stopped writting data!");
            Toast.makeText(MainActivity.this,"File saved successfully!",
            Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            text.setText(e.getMessage());
            e.printStackTrace();
        }
    }
//    protected void onResume() {
//        super.onResume();
//
//    }
    @Override
    public void onSensorChanged(SensorEvent event) {



        double[][] Z=new double[3][1];
        for (int i=0; i<3; i++) {
            Z[i][0] = event.values[i];
        }
        double[][] zcap = KF.track(Z, System.currentTimeMillis()/1e3);

        xText.setText("X: " + Z[0][0]+"->X_new: "+zcap[0][0]);
        yText.setText("Y: " + Z[1][0]+"->Y_new: "+zcap[1][0]);
        zText.setText("Z: " + Z[2][0]+"->Z_new: "+zcap[2][0]);
        if (flag) {
            String str = (Z[0][0] + "," + Z[1][0] + "," + Z[2][0] + "\t\t\t" + zcap[0][0]+','+zcap[1]
[0]+','+zcap[2][0]+'\n');

        try {
//            outputWriter.write(str);
            writer.append(str);
            writer.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    }

    @Override
    public void onAccuracyChanged(Sensor sensor,int accuracy){

    }

}
```

Matlab Code to generate plots

```
% Matlab Code to generate plots:

load testmat_pratyush;

% Initial Guess
% state

x=randn(9,1);

% Covariance

P=eye(9);



% Process Noise covariance Q

Q=0.0007*eye(9);



% Measurement Noise covariance R

R=0.07*eye(3);



X=[]; Z=[]; Gain=[]; Err=[];

N=length(t);

% Construct H matrix

        H=[zeros(3,6) eye(3)];



for n=1:N-1

        h=t(n+1)-t(n);

        h2=h^2/2;

        % Construct Phi matrix
        phi=[eye(3)      h*eye(3) h2*eye(3)
        zeros(3)         eye(3)  h*eye(3)
        zeros(3)         zeros(3)    eye(3)];

        %  Compute the Kalman Gain K
        K=P*H'*inv(H*P*H'+R);

        % Update the states
        z=a(n,:)';
        err=(z-H*x);
        x=x+K*err;

        % Update the P
        P=(eye(9)-K*H)*P;

        %Save the values
        X=[X;x(:)'];
        Z=[Z;z(:)'];
        Gain=[Gain;K(:)'];
        Err=[Err;err(:)'];

        % Project Ahead
        x=phi*x;
        P=phi*P*phi'+Q;
end
```

```
ae=[X(:,7:9);a(end,:)];


subplot(311)

plot([a(:,1) ae(:,1)])

grid on


subplot(312)

plot([a(:,2) ae(:,2)])

grid on



subplot(313)

plot([a(:,3) ae(:,3)])

grid on


shg
```

Matlab Code for training and plotting from generated data

```
clc;
clear all;

axyzr = readmatrix('Filtered_Data_run.txt');
axyzr2 = readmatrix('Filtered_Data_run2.txt');

axyzw = readmatrix('Filtered_Data_walk.txt');
axyzw2 = readmatrix('Filtered_Data_walk2.txt');
```

```
axyzs = readmatrix('Filtered_Data_still.txt');

axyzd = readmatrix('Filtered_Data_down1.txt');
axyzd2 = readmatrix('Filtered_Data_down2.txt');
axyzd3 = readmatrix('Filtered_Data_down3.txt');

axyzu = readmatrix('Filtered_Data_up1.txt');
axyzu2 = readmatrix('Filtered_Data_up2.txt');
axyzu3 = readmatrix('Filtered_Data_up3.txt');

ar = [];
aw = [];
as = [];
ad = [];
au = [];

for i=1:size(axyzr, 1)
    ar = [ar; norm(axyzr(i,:), 2)];
    aw = [aw; norm(axyzw(i,:), 2)];
    as = [as; norm(axyzs(i,:), 2)];
end
for i=1:size(axyzr2, 1)
    ar = [ar; norm(axyzr2(i,:), 2)];
    aw = [aw; norm(axyzw2(i,:), 2)];
end
for i=1:size(axyzd, 1)
    ad = [ad; norm(axyzd(i,:), 2)];
    au = [au; norm(axyzu(i,:), 2)];
end
for i=1:size(axyzd2, 1)
    ad = [ad; norm(axyzd2(i,:), 2)];
    au = [au; norm(axyzu2(i,:), 2)];
end
for i=1:size(axyzd3, 1)
    ad = [ad; norm(axyzd3(i,:), 2)];
    au = [au; norm(axyzu3(i,:), 2)];
end

% ar = ar - mean(ar);
% aw = aw - mean(aw);
% as = as - mean(as);
% ad = ad - mean(ad);
% au = au - mean(au);
```

```
XIr = 0:40/100:40;
size(XIr)
[Fr, XIr] = ksdensity(ar, XIr);
[Fw, XIw] = ksdensity(aw, XIr);
[Fs, XIs] = ksdensity(as, XIr);
[Fd, XId] = ksdensity(ad, XIr);
[Fu, XIu] = ksdensity(au, XIr);
```

```
Fr = Fr.*(1/sum(Fr));
Fw = Fw.*(1/sum(Fw));
Fs = Fs.*(1/sum(Fs));
Fd = Fd.*(1/sum(Fd));
Fu = Fu.*(1/sum(Fu));
```

```
for i=1:size(XIr, 2);
   Fr(i) = Fr(i)+eps;
   Fw(i) = Fw(i)+eps;
   Fs(i) = Fs(i)+eps;
   Fd(i) = Fd(i)+eps;
   Fu(i) = Fu(i)+eps;
   if Fr(i) == 0 | Fw(i) == 0 | Fs(i) == 0 | Fd(i) == 0 | Fu(i) == 0
       disp("Warning");
   end
end
```

```
markersize = 10;
linewidth = 1;
fontsize = 18;
```

```
figure(1)
hold on
plot(XIr, Fr, 'LineWidth', linewidth ,'MarkerSize',markersize)
plot(XIw, Fw, 'LineWidth', linewidth ,'MarkerSize',markersize)
plot(XIs, Fs, 'LineWidth', linewidth ,'MarkerSize',markersize)
plot(XId, Fd, 'LineWidth', linewidth ,'MarkerSize',markersize)
plot(XIu, Fu, 'LineWidth', linewidth ,'MarkerSize',markersize)
legend('Running', 'Walking', 'Still', 'Going down', 'Going up',
'FontSize', fontsize);
xlabel('Acceleration m/s^2')
ylabel('Probability')
title('Standard Probability Distributions')
hold off
```

```
fileID = fopen('ExtractedPDF\Run.txt', 'w');
```

```
 for i=1:length(Fr)
     fp = Fr(i);
     fprintf(fileID, "%e, ", fp);
 end
fclose(fileID);
```

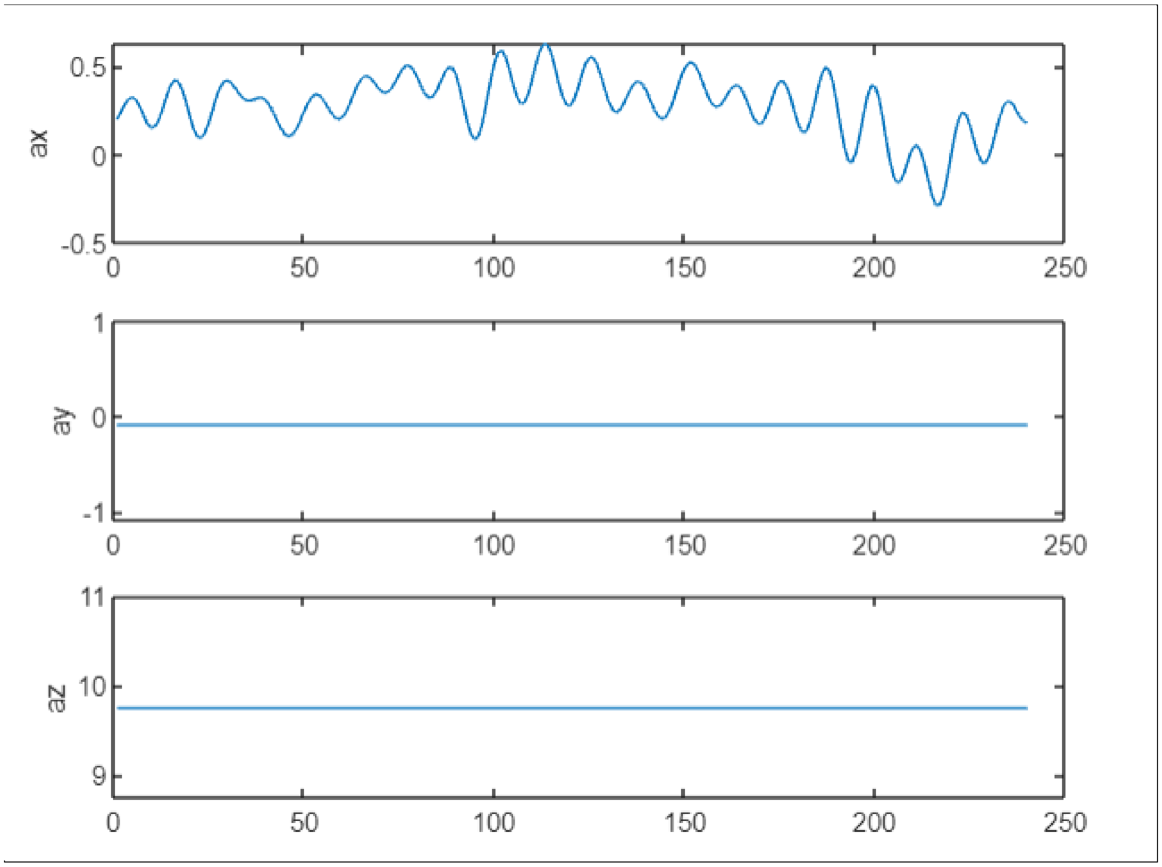
```
fileID = fopen('ExtractedPDF\Walk.txt', 'w');
 for i=1:length(Fw)
     fp = Fw(i);
     fprintf(fileID, "%e, ", fp);
 end
fclose(fileID);
```

```
fileID = fopen('ExtractedPDF\Still.txt', 'w');
 for i=1:length(Fs)
     fp = Fs(i);
     fprintf(fileID, "%e, ", fp);
 end
fclose(fileID);
```

```
fileID = fopen('ExtractedPDF\Up.txt', 'w');
 for i=1:length(Fu)
     fp = Fu(i);
     fprintf(fileID, "%e, ", fp);
 end
fclose(fileID);
```

```
fileID = fopen('ExtractedPDF\Down.txt', 'w');
 for i=1:length(Fd)
     fp = Fd(i);
     fprintf(fileID, "%e, ", fp);
 end
fclose(fileID);
```

Taking fourier transform of accelerometer data along each axis
Derive the magnitudes of fourier coefficients using log operation
Threshold the Fourier Space data using magnitude as criteria
Taking the Inverse fourier transform
For x-axis motion :

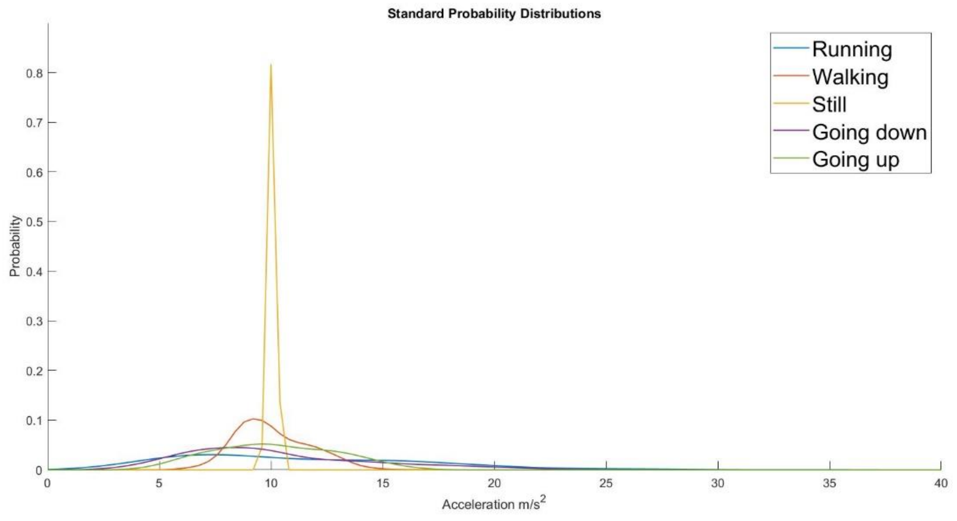


Next the SVM is designed and solved using quadprog

```
%% Design SVM
H = zeros(n,n);
for i=1:n
   for j=i:n
       H(i,j) = y(i)*y(j)*x(:,i)'*x(:,j);
       H(j,i) = H(i,j);
   end
end
f = -ones(n,1);
Aeq=y;
beq=0;
lb=zeros(n,1);
ub=C*ones(n,1);
Alg{1}='trust-region-reflective';
Alg{2}='interior-point-convex';
options=optimset('Algorithm',Alg{2},...
   'Display','off',...
   'MaxIter',20);
alpha=quadprog(H,f,[],[],Aeq,beq,lb,ub,[],options)';
AlmostZero=(abs(alpha)<max(abs(alpha))/1e5);
alpha(AlmostZero)=0;
S=find(alpha>0 & alpha<C);
w=0;
```

```
for i=S
   w=w+alpha(i)*y(i)*x(:,i);
end
b=mean(y(S)-w'*x(:,S));
```

Results


Standard Probability Distributions
Running
Walking
Still
Going down
Going up
Probability
Acceleration m/s2