List of Figures

Abbreviations

GSP – graph signal processing

IDS – intrusion detection system

CPS – cyber physical system

IIOT – Industrial Internet Of Things

PCA – Principal Component Analysis

GBF – Graph based Filtering

Notations

D_{pq}	Geometrical distance
σ_{pq}	Variance
D_g^2	Signal value distance
σ	Variance
$\boldsymbol{L}$	Laplacian Matrix
$\boldsymbol{D}$	Diagonal Matrix
$\boldsymbol{W}$	Weight Matrix
λ_i	Eigen Values
$\boldsymbol{u}_i$	Eigen Vectors
$f(\boldsymbol{s}(k))$	Probability Distribution
$\boldsymbol{Q}(k)$	Precision Matrix
C_{ij}	Covariance matrix
$\|\|\cdot\|\|_F$	Frobenious Norm
H_1	Alternative hypothesis
H_0	Null hypothesis
μ_0	Mean
$d_{\mathrm{B}}\left(f_k, f_{k+1}\right)$	Bhattacharyya distance

Abstract

Cyber-Physical Systems (CPS) or ‘smart’ systems are co-engineered interacting networks of physical and computational components. They are a generalization of embedded systems and possess compute, communicate and control capabilities, they interact with the physical world through sensors and actuators. An intrusion detection system (IDS) is a system that monitors the activities of a given CPS and searches for anything suspicious and alerts the user when it comes across such an activity the paper uses sensor measurements as the target graph-signal and utilizing the statistical properties of the graph-signal for intrusion detection. Intrusion detection comes of 2 types one is signature based where a detailed list of possible anomalies is given to the IDS these possible anomalies are called signatures this type of IDS finds difficulties in finding new attacks. The other type is anomaly based where anything that the IDS is not familiar with is considered to be an anomaly, this can be applied with machine learning where the model can be trained to acquire knowledge about previous activities. The paper utilizes graph signal processing (GSP) which is basically signal processing done on graphs, Research in GSP aims to develop tools for processing data defined on irregular graph domains, we need to extend classical signal processing concepts and tools such as Fourier transform, filtering and frequency response to data residing on graphs. All of this is collaboratively done by the GSP of our model.

Table of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title**:** Distributed Graph-based Statistical Approach for Intrusion Detection in Cyber-Physical Systems

Journal Name: IET CYBER-PHYSICAL SYSTEMS: THEORY & APPLICATIONS

Publisher: IEEE

Year: 2018

This paper focuses on a network of sensors and the variety of changes are captured by the sensor nodes. Sensor networks play a vital role in CPSs as it is this framework that manages the infrastructure economically and socially. It should be noted that this network intrusion detection play a very important role of network security and as we keep heading into the future it will only prove to be more importance. The paper implements a blind intrusion detection system using statistical properties of the target graph signals. The network consists of a variety of nodes, The time-series temperature data are generated in the intervals of 1 hour for 30 days. We consider 64 randomly distributed sensors collecting data corresponding to the room temperature at 64 locations for 720-time instants.

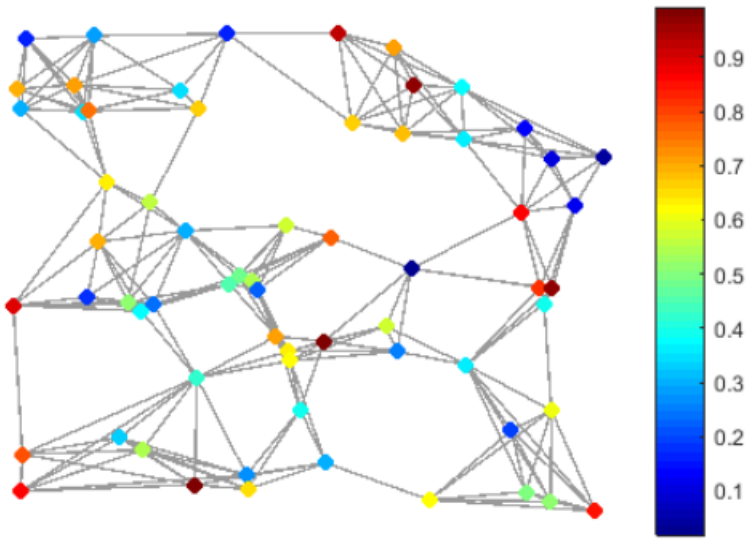


Fig 1

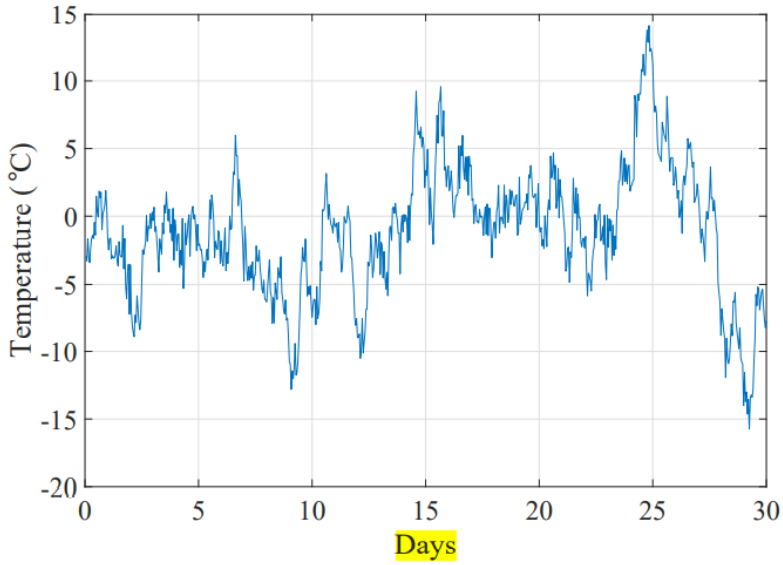


Fig 2

The proposed architecture for the implementation of the paper is as follows:

Initially a graph affinity matrix is constructed based on the surroundings of the sensors an affinity matrix also called as similarity matrix shows the mutual similarities between corresponding nodes in this case sensors for example spatial dependencies.

The magnitude of the calculated date from the sensors are random variables which are graph signals and they are distributed with Gaussian Markov Random Field (GMRF), i.e. The GMRF model is used to detect any abnormal behaviour from the standard supervised model. The GSP of the paper is split into 3 main parts which are i) Graph Construction ii) Graph model iii) Parameter Estimation.

The GSP detects any deviation from the normal behaviour of the network, A Bayesian log-likelihood ratio test is a detection method based on the graph signal statistics, it is a hypothesis-based testing where the hypotheses H1 and H0 represent as to whether the signal has an anomaly or not H1 being there is an anomaly in the graph signal and H0 being there isn't.

A temporal analysis also known as time series analysis which analyses the graph signals with respect to time is done be calculating the Bhattacharyya distance between the probability distributions at consecutive time intervals.

CHAPTER 2

METHODOLOGY

Proposed graph based modeling framework

- Measurement corresponding to sensor *l*

$$S_l(k) = \boldsymbol{h}_l(\boldsymbol{x}(k)) + \nu_l(k),$$

k - time index
$v_l(k)$ - uncertainty in sensor model
$\boldsymbol{h}_l(.)$ - general observational model

- Instantaneous graph signal

$$\boldsymbol{s}(k) = [S_1(k), \ldots, S_N(k)],$$

Graph construction

- Similarity weight defined by standard Gaussian kernel

$$w_{\mathrm{pq}} = \exp\left(-\left(\frac{D_{\mathrm{pq}}^2}{2\sigma_{\mathrm{pq}}^2} + \frac{D_g^2}{2\sigma_g^2}\right)\right),$$

Where D_{pq} is the geometrical distance

$$D_{\mathrm{pq}} = \sqrt{(X_p - X_q)^2 + (Y_p - Y_q)^2},$$

D^2_g is the signal value distance

$$D_g^2 = (S_p - S_q)^2$$

- Laplacian matrix

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W},$$

W - graph similarity matrix

$$\boldsymbol{D} = \mathrm{diag}\left\{\sum_q w_{1q}, \ldots, \sum_q w_{\mathrm{Nq}}\right\}.$$

- Eigen decomposition of Laplacian

$$\boldsymbol{L} = \sum_i \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^T,$$

$\lambda = \{\lambda_i\}_{i=1,\dots,N}$ - set of eigen values

$\mathcal{U} = \{\boldsymbol{u}_i\}_{i=1,\dots,N}$ - set of orthogonal eigen vectors

- Decomposing graph signal ***s***(*k*) into graph Fourier domain component

$$\tilde{\boldsymbol{s}}(k) = \boldsymbol{U}(k)^T \boldsymbol{s}(k),$$

U(*k*) is matrix composed of the eigen vectors of ***L***(*k*)

- Graph smoothness regularizer

$$\boldsymbol{s}^T(k)\boldsymbol{L}(k)\boldsymbol{s}(k) = \sum_{i=1}^{N} \lambda_i \left(\boldsymbol{u}_i(k)^T \boldsymbol{s}(k)\right)^2,$$

Graph model

- GMRF probability density function

$$f(\boldsymbol{s}(k)) = (2\pi)^{\frac{-N}{2}} |\boldsymbol{Q}(k)|^{\frac{1}{2}} \times \exp\left(\frac{-1}{2}(\boldsymbol{s}(k) - \boldsymbol{m}(k))^T \boldsymbol{Q}(k)(\boldsymbol{s}(k) - \boldsymbol{m}(k))\right)$$

m(*k*) - mean vector
Q(*k*) - symmetric precision matrix

Relation between precision matrix and similarity matrix

$$Q_{\acute{p}\acute{q}} = \begin{cases} \sum_{\acute{q}} w_{\acute{p}\acute{q}} & \acute{p} = \acute{q} \\ -w_{\acute{p}\acute{q}} & \acute{p} \neq \acute{q} \end{cases}$$

Graph based intrusion detection

- Binary hypothesis test

$$H_1 \quad : \quad \boldsymbol{y}(k) \quad = \boldsymbol{s}(k) + \xi \boldsymbol{a}(k)$$

$$H_0 \quad : \quad \boldsymbol{y}(k) \quad = \boldsymbol{s}(k)$$

H_1 - alternate hypothesis
H_0 - null hypothesis

H_1 and H_0 represent wether the signal suffers from significant change by the anomaly $\boldsymbol{a}$

- Decision rule defined using likelihood ratio test

$$\Lambda(\boldsymbol{y}(k)) = \frac{Pr(\boldsymbol{y}(k)|H_1)}{Pr(\boldsymbol{y}(k)|H_0)}$$

$$= \frac{Pr(\boldsymbol{y}(k) - \xi \boldsymbol{a}(k); \boldsymbol{Q}(k), \boldsymbol{m}(k))}{Pr(\boldsymbol{y}(k); \boldsymbol{Q}(k), \boldsymbol{m}(k))} \begin{matrix} H_1 \\ > \\ < \\ H_0 \end{matrix} \eta,$$

η - Threshold

- Log-likelihood ratio

$$l(\boldsymbol{y}(k)) = \ln\left(\frac{Pr(\boldsymbol{y}(k)|H_1)}{Pr(\boldsymbol{y}(k)|H_0)}\right)$$

$$= \ln\left(\frac{Pr(\boldsymbol{y}(k) - \xi \boldsymbol{a}(k); \boldsymbol{Q}(k), \boldsymbol{m}(k))}{Pr(\boldsymbol{y}(k); \boldsymbol{Q}(k), \boldsymbol{m}(k))}\right) \begin{matrix} H_1 \\ > \\ < \\ H_0 \end{matrix} \tau,$$

- Decision statistic

$$l(\boldsymbol{y}(k)) = -\frac{1}{2}(\boldsymbol{y}(k) - \xi \boldsymbol{a}(k))^T \boldsymbol{Q}(k)(\boldsymbol{y}(k) - \xi \boldsymbol{a}(k))$$
$$+ \frac{1}{2}\boldsymbol{y}(k)^T \boldsymbol{Q}(k)\boldsymbol{y}(k).$$

- Probabilities of false alarm and detection

$$P_{\text{Fa}} = Q\left(\frac{\tau - \mu_0}{\sigma_0}\right)$$
$$P_{\text{Det}} = Q\left(\frac{\tau - \mu_1}{\sigma_1}\right),$$

Where $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-z^2/2} dz$

- Decision threshold obtained using Neyman-Pearson criterion

$$\tau = \sigma_0 Q^{-1}(P_{\text{Fa}}) + \mu_0.$$

- Expression for analyzing the performance of the proposed detector

$$P_{\text{Det}} = Q\left(\frac{\sigma_0}{\sigma_1} Q^{-1}(P_{\text{Fa}}) - \frac{\mu_1 - \mu_0}{\sigma_1}\right).$$

CHAPTER 3

MERITS AND DEMERITS

In the related works of the base paper there are a variety of signal analysis approaches out of which one of them namely PCA, is commonly used in a lot of IDS and Machine Learning models. It is a dimension reduction method that can be useful in intrusion detection by applying the data which we get from observing the CPS onto the principal axis and splitting the given subspace into normal and abnormal by setting a constant threshold. The abnormal values are then removed giving refined data or more relevant data.

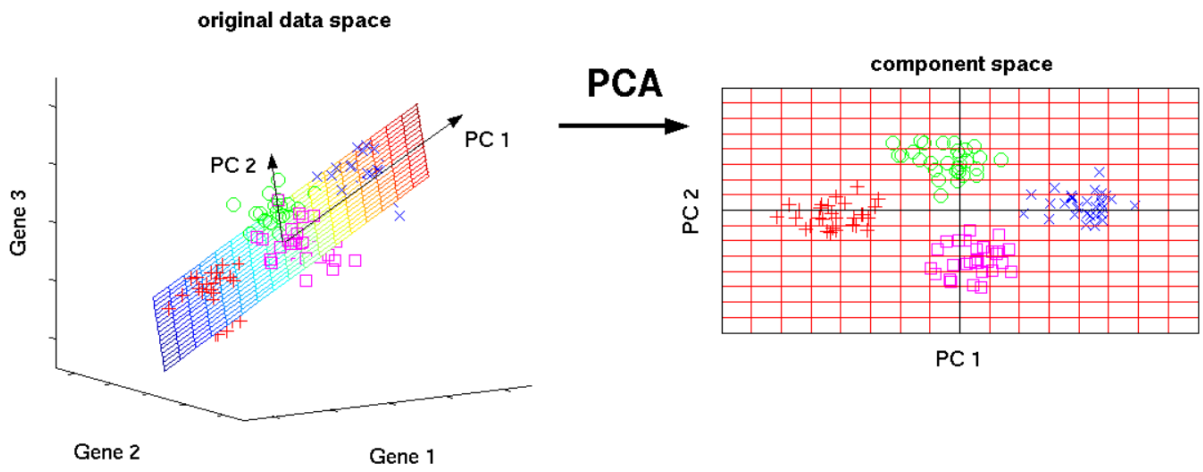


Fig 3

Merits:

- We are using a blind intrusion detection system, meaning that, the attacker has no way of knowing what underlying structure we have in hand to use that to detect him.

- The graph model we proposed is easily trainable for a given Cyber Physical System and it is very simple.

- The computations required by this model is comparatively less and the detection delay is insignificant.

- The proposed model is highly accurate and outperforms the other models in a given monotonous Cyber Physical System.

- This model is much easier to visualise and thus will prove useful while finding the point of intrusion.

- It is highly flexible in terms of creating a specific threshold according to the user needs.

Demerits:

- This proposed system will only work on a single type of data within a Cyber Physical System. For e.g, in this project we use temperature sensor's data.

- It will only be effective as long as the intruder doesn't know about the graph structure, if the intruder already knows the underlying graph structure, he can find ways to avoid detection.

- For an exceedingly complex Cyber Physical System, the proposed model's space and time complexity may make the computations and threshold value to vary.

CHAPTER 4

SOURCE CODE

Generating Dataset:
Temperature Dataset

```
import random
import csv

l = []
l1 = []

##l2 = []
##l3 = []
##x = 0
for a in range(10):
    l = []
    file_name = 'temp_time'+ str(a)+ '.csv'
    for i in range(1):
        l1 = []
        #l1 += ['#'+ str(i)]

        l3 = []
        l3 += [i]

        for j in range(64):
            l1 += [round(random.uniform(-20.0, 15.0), 1)]

    ##    for k in l1[1:]:
    ##        x += k
    ##    y = round(x / 64, 1)
    ##
    ##    x = 0
    ##    l1 += ['##' + str(y)]
        l += [l1]

        #l3 += [y]

        #l2 += [l3]

    with open(file_name, 'a', newline='') as file:
```

```
        writer = csv.writer(file)
        writer.writerows(l)

##with open('datatest2.csv', 'a', newline='') as file:
##     writer = csv.writer(file)
##     writer.writerows(l2)
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
--
```

Graph Signal Processing

Graph Construction:
Calculating Weight and Laplacian matrix

```
import csv
import copy
import math
import statistics

l1 = []     #contains position (x,y) of all the 64 sensors -> [[x0, y0], [x1, y1], ... ]

with open('graphdataset.csv') as file:
    reader = csv.reader(file)
    for row in reader:
        l = []      #(xi, yi) 0<= i < 63 -> [xi, yi]
        x = float(row[0])
        y = float(row[1])
        l = [x,y]
        l1 += [l]
#print(l1)
l2 = []     #distance of si from sj 0<= i, j < 63 -> [[d0, d1, d2, ... , d63], [
for i in l1:
    d1 = []
```

```
    for j in l1:
        d = round(((i[0] - j[0]) ** 2 + (i[1] - j[1]) ** 2) ** 0.5, 2)
        d1 += [d]
    l2 += [d1]
#print(l2)
l3 = copy.deepcopy(l2)

#============================================================
min2 = []
min3 = []          #6 nearest sensor of each sensor -> [[dist1, sensor number1], [dist2, sensor number2], ... [dist6, sensor number6],

for i in l3:
    min2 = []
    for j in range(7):
        min1 = min(i)
        k = i.index(min1)
        min2 += [min1, k]
        i[k] = 9999
    min2.remove(0.0)
    min2.pop(0)
    min3 += [min2]
#print(min3)
#============================================================
ll = []
ll1 = []
t = []
ll2 = []
ll3 = [] #weight matrix
for a in range(10):
    ll = []
    ll1 = []
```

```
    t = []
    ll2 = []
    ll3 = [] #weight matrix
    t = []
    file_name = 'temp_time'+ str(a)+ '.csv'
    file_name1 ='laplacianMat_time'+ str(a)+ '.csv'
    #file_name2 = 'weightMat_time'+ str(a)+ '.csv'

    with open(file_name, newline='') as file:
        reader = csv.reader(file)
        for row in reader:
            for j in row:
                t += [float(j)]
    print(t)
    #print(t)
    sums = 0 #####
    for i in range(64):
        ll = []
        ll2 = []
        k = 0
            s = [min3[i][1], min3[i][3], min3[i][5], min3[i][7], min3[i][9], min3[i][11]] #nearest
sensor number
        #print(s)
        for j in range(64):
            if j in s:
                jj = j
                ll += [min3[i][k]]
                #print(ll)
                if t[i] != t[jj] : #####
                    sums = ((t[i] - t[jj]) ** 2) / ( 2 * statistics.variance(t))
                    #print(t[i], t[j], sums)
                if i != j:
```

```
                ll2 += [round(math.exp(-    (( (min3[i][k] ** 2) / (2 * statistics.variance([i, jj])) ) + sums)), 2) ] ######
                #print(round(math.exp(-    (( (min3[i][k] ** 2) / (2 * statistics.variance([i, jj])) ) + sums)), 2) )
                k += 2
                #print(k)
            else:
                ll += [0]
                ll2 += [0]
        ll1 += [ll]
        ll3 += [ll2]
##    with open(file_name2, 'a', newline='') as file:
##        writer = csv.writer(file)
##        writer.writerows(ll3)

    #print(ll1)
    #print(ll3)
    ll4 = []
    ll5 = [] #diagonal matrix

    for i in range(64):
        weight = 0
        ll4 = []
        for j in range(64):
            weight += ll3[i][j]

        for k in range(64):
            if i == k:
                ll4 += [round(weight, 2)]
            else:
                ll4 += [0]
        ll5 += [ll4]
```

```
    #print(ll5)
    ll6 = [] #laplacian matrix
    for i in range(64):
        ll7 = []
        for j in range(64):
            ll7 += [ll5[i][j] - ll3[i][j]]

        ll6 += [ll7]

    #print(ll6)
    with open(file_name1, 'a', newline='') as file:
        writer = csv.writer(file)
        writer.writerows(ll6)
```

Eigen decomposition

```
import numpy as np
import csv
from numpy.linalg import eig

arr = []
arr1 = []
with open("laplacianMat_time0.csv") as lm:
        reader = csv.reader(lm)
        for row in reader:
                arr = []
                for i in row:
                        arr += [float(i)]
                arr1 += [arr]

#print(arr1)
```

```
arr1 = np.array(arr1)
arr1t = arr1.transpose()
Laplace = []
Laplace1 = []
for i in range(64):
        Laplace1 = []
        for j in range(64):
                if((arr1[i][j]==0 and arr1t[i][j]!=0) or (arr1[i][j]!=0 and arr1t[i][j]==0) ):
                        Laplace1 += [arr1[i][j] + arr1t[i][j]]
                else:
                        Laplace1 += [round((arr1[i][j] + arr1t[i][j])/2, 2)]


        Laplace += [Laplace1]
#for i in Laplace:
#       print(i)
Laplace = np.array(Laplace)


w,v=eig(Laplace) #w = eigen values ; v = eigen vectors
vt = v.transpose() #Transpose of eigen vectors
L = []
for i in range(64):
        temp= w[i]*v[i]*vt[i]
        L.append(temp)
print(L)
state = []
with open("temp_dataset.csv",'r') as tds :
        reader = csv.reader(tds)
        for i in reader:
                for j in i:
                        state += [float(j)]
                #print(i)
```

```
        #print(state)
        state = np.array(state)
        state = state.transpose()
#print(state)
statedash = [] # Fourier Decomposition
statedash = np.dot(vt,state)
#print(statedash.shape)
graph_smoothness_regularizer = [] # optimizing graph smoothness
for i in range(64):
   lol = w[i] * (vt[i]*state[i])**2
   graph_smoothness_regularizer.append(lol)
graph_smoothness_regularizer = np.dot(np.dot(np.dot(vt,state),np.dot(vt,state)),w)
graph_smoothness_regularizer = np.array(graph_smoothness_regularizer)
#print(graph_smoothness_regularizer.shape)

with open("Symmetric_laplacian_matrix.csv",'a',newline="") as file:
        writer = csv.writer(file)
        writer.writerows(Laplace)

with open("Eigen_decomposition1.csv",'a',newline="") as file:
        writer = csv.writer(file)
        writer.writerows(L)


with open("graph_smoothness_regularizer.csv",'a',newline="") as file:
        writer = csv.writer(file)
        writer.writerows(graph_smoothness_regularizer)
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
```

Intrusion Detection :

```
%% Consider eps = 0.5

% y=zeros(1,1000) ;
% Pfaa = zeros(1,1001) ;
% var0 = 8 ;
% mean0 = 5 ;
% Pfaa=(1e-5:1e-5:1e-2) ;
% for i = 1 : 1000
%     Pfa = Pfaa(1,i);
%     T = npwgnthresh(Pfa,1,'real')
%     variance = 1;
%     T = sqrt(variance * db2pow(T));
%     Pfa = i ;
%     Pd = (T - var0) / mean0 ;
%     fun = @(Pd) ( exp(-Pd.^2/2) ) ;
%     Pdet = (1/sqrt(2*pi)) * integral(fun,Pd,Inf) ;
%     y(1,i) = Pdet;
% end
% y1 = zeros(1,1000);
% y1(1,2:1000) = y(1,1:999);
% plot(Pfaa,y1,'color','red')
% Area_of_ROC = trapz(Pfaa,y1) * 1e2
% xlabel('Probability of False Alarm PFa [0 - 10^-2]')
% ylabel('Pobability of Detection PDet [0 - 1]')

rng default
Pfa = 1e-3;
T = npwgnthresh(Pfa,1,'real');
variance = 1;
threshold = sqrt(variance * db2pow(T));
rng default
N = 100;
x = sqrt(variance) * randn(N,1);
x1 = x(1:100);Pfa = 1e-3;
T = npwgnthresh(Pfa,1,'real');
variance = 1;
threshold = sqrt(variance * db2pow(T));
rng default
```

```
N = 1e6;
x = sqrt(variance) * randn(N,1);
x1 = x(1:100);
plot(x1)
line([1 length(x1)],[threshold threshold],'Color','red')
xlabel('Sample')
ylabel('Value')
plot(x1)
line([1 length(x1)],[threshold threshold],'Color','red')
xlabel('Sample')
ylabel('Value')
```

```
# gatherdata.m

s = zeros(1,64,10)
for i = 1 : 10
T = readtable(strcat('C:\Users\Shiv\Documents\MATLAB\Datasets\weightMat_time',int2str(i-1),'.csv'))
W(:,:,i) = T{:,:};
T = readtable(strcat('C:\Users\Shiv\Documents\MATLAB\Datasets\laplacianMat_time',int2str(i-1),'.csv'))
lset(:,:,i) = T{:,:};
T = csvread(strcat('C:\Users\Shiv\Documents\MATLAB\Datasets\temp_time',int2str(i-1),'.csv'))
s(1:1,1:64,i) = T(1,:);
end
save('mymat.mat','s','W','lset')
```

```
clear all
close all
clc

%%%%%%%%%%

load('mymat.mat')

Array=csvread('graphdataset2.csv');
col1 = Array(:, 1);
col2 = Array(:, 2);
```

```
scatter(col1,col2)


% %
% S=csvread("datatest1.csv") ;
% s=S(1:5,1:64) ;
%
T=readtable('graphdataset2.csv');
p=T{:,1:2};

%p = vertex coordinates
%

% W = csvread('weight_matrix.csv') ;
w = zeros(64,64,10)
for t = 1:5
for i = 1 : 64
   for j = 1 : 64
      if W(i,j,t) > 0
         w(i,j,t)=1+W(i,j,t);
      end
   end
end
end

% %w - Weight matrix
%
% ed = readtable('Eigen_decomposition1.csv') ;
% ed = ed(1:64,1:64) ;
%
% L = readtable('laplacian_matrix.csv')  ;
% l = L{1:64,1:64} ;
%
%
% save('mymat.mat','p','w','s','l','ed')
%
% s = Array(:,1);

%%%%%%%%%%
```

```
%%%%%%%%%%

y = s;
N = 64 ;
% No. of sensors

k = 10 ;
% Time instances

% Mean vector

% for i = 1 : 5
%     m(i) = mean(s(i)) ;
% end
sum = 0
x1 = -0.59
x2 = 1
for i = 1:10
[u v]=eig(lset(i));
sum= sum +  u *  transpose(v * s(i) )
ed(i) = sum
sum = 0
   lset(i) = norm(s(i)-mean(s(i)) ,"fro")^2  + x1*trace(ed(i)) + x2*(norm(lset(i),"fro"))^2
end

% Precision Matrix

%METHOD 0
% Q = zeros(64,64,10)
% for t = 1 : 10
% for i = 1 : 64
%     for j = 1 : 64
%         if w(i,j,t) > 0
%             Q(i,j,t) = -lset(i,j,t);
%         end
%     end
% end
```

```
% end


%METHOD 1
Q = zeros(64,64,10) ;
sum = 0;
for t = 1 : 10
   for i = 1 : 64
      for j = 1 : 64
            if  i == j
            for q = 1 : 64
            sum = sum + w(i,q,t);
            end
            Q(i,j,t) = sum;
            sum = 0;
            else
            Q(i,j,t) = - 1 * w(i,j,t);
            end
      end
   end
end

% METHOD 2
% C = zeros(64,64,10);
% sum = 0
% for i = 1 : 64
%    for j = 1 : 64
%       for z = 1 : 10
%          sum = sum + transpose( s(1,i,t) - mean(s(1,i,t) ) ) * ( s(1,j,t) - mean(s(1,:,t) ) );
%       end
%       C(i,j,z) = (1/64-1) * sum
%       sum = 0
%    end
% end
% for i = 1 : 10
% Q(:,:,i) = inv(C(:,:,i))
% end


for i = 1 : 10
```

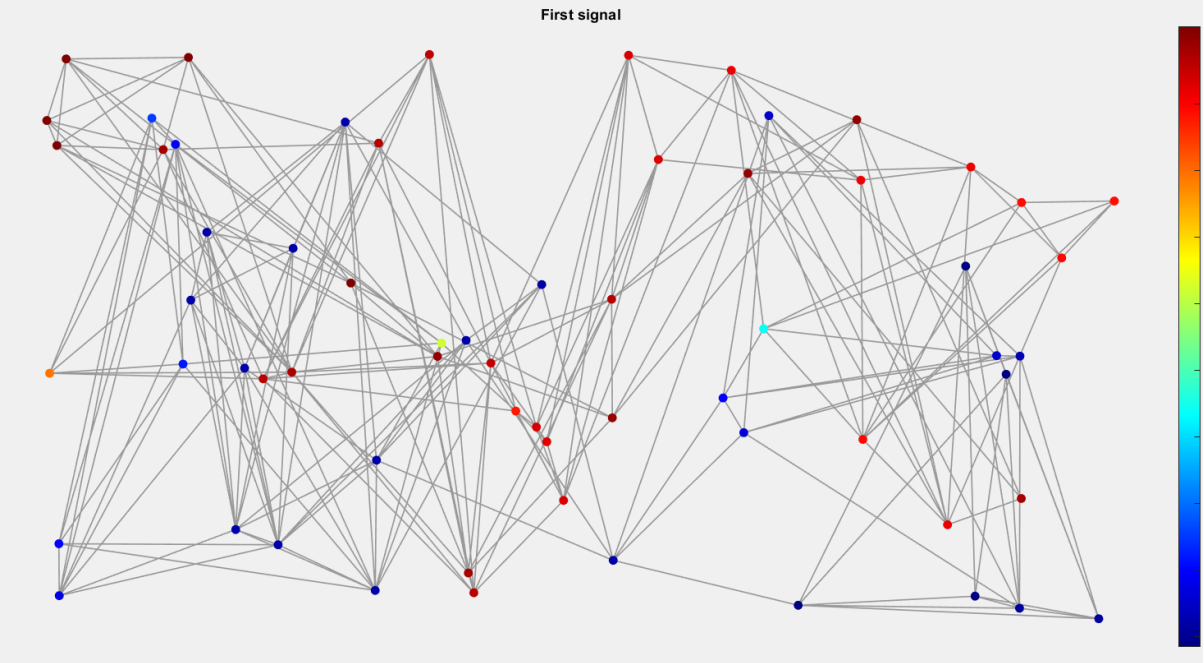
```
C(:,:,i) = cov(y(:,:,i))
Q(:,:,i) = inv(C(:,:,i))
end
means= [1:10];
vars = [1:10];
eps=0.5
for k = 1:10
means(k) = - (1/4) * (y(:,:,k) - eps) * Q(:,:,k) * ( transpose(y(:,:,k)) - eps ) - (1/4) *  (y(:,:,k) +eps) * Q(:,:,k) * (transpose(y(:,:,k) + eps )) + (1/2) * ( ( y(:,:,k) ) * Q(:,:,k) * transpose(y(:,:,k)) ) ;
vars(k) =  (1/16) * ( ( (y(:,:,k) - eps) * Q(:,:,k) * ( transpose(y(:,:,k)) - eps ) ) +  ( y(:,:,k) +eps ) * Q(:,:,k) * ( transpose(y(:,:,k) + eps ) ) ) .^ 2 ;
end
mean0 = min(means)
var0 = max(vars)
threshold = 7.335
for t = 1 : 10
for i = 1 : 64
if ( means(k) > threshold )
   means(k) = -1 * means(-k) ;
end
end
end


%mean1 = -1* mean0 ;
%var1 = var0 ;

%var = var0/var1 ;
% var1 = 1
% mean1 = 3
% mean0 = 0.1
% var0 = 2

x1 = -0.05
x2 = 0.02
for i = 1:10
[u v]=eig(lset(i));
sum= sum +  u *  transpose(v * s(i ) )
ed(i) = sum
```

```
sum = 0
   lset(i) = norm(s(i)-mean(s(i)) ,"fro")^2  + x1*trace(ed(i)) + x2*(norm(lset(i),"fro"))^2
end

for i = 1 : 9
dislset(i) = 10 + abs(norm(lset(:,:,i) - lset(:,:,i+1), "fro")) ;
end

%m(:) = mean(s(:))

i=0;
BD = zeros(1,9);
for i = 1 : 9
m(i) = mean(s(1,:,i));
m(i+1) = mean(s(1,:,i+1));
mt = m(i) - m(i+1) ;
Qt(:,:) = ( Q(:,:,i) + Q(:,:,i+1) ) ./ 2 ;
denm(:,:) = Q(:,:,i) * Q(:,:,i+1) ;
%dellll(:,:) = log( Qt / sqrt(denm) );
BDd(:,:,i) = (1/8) * mt^2 * Qt(:,:) + log(0.5 * Qt(:,:)./denm(:,:))
BD(i) = norm(BDd(:,:,i))
% BD(1,i) = bhattacharyya(X1,X2);
end
figure
hold on
plot(BD,'r')
plot(dislset,'g')

legend('Bhattacharyya','LaplacianDist')
%syms x
% f19(x) = exp((-x^2)/2) ;
% syms Qf(x)
% fun = exp(-x^2);
% Qf =  (1/sqrt(2*pi)) * int(fun,x,x,Inf) ;
% iQf = finverse(Qf) ;
% Pfa = 0.5 ;
% Pdet = Qf(var * iQf(Pfa) - (mean1-mean0)/var1) ;

% Pfa = 0.01;
% T = npwgnthresh(Pfa,var0,'coherent');
```

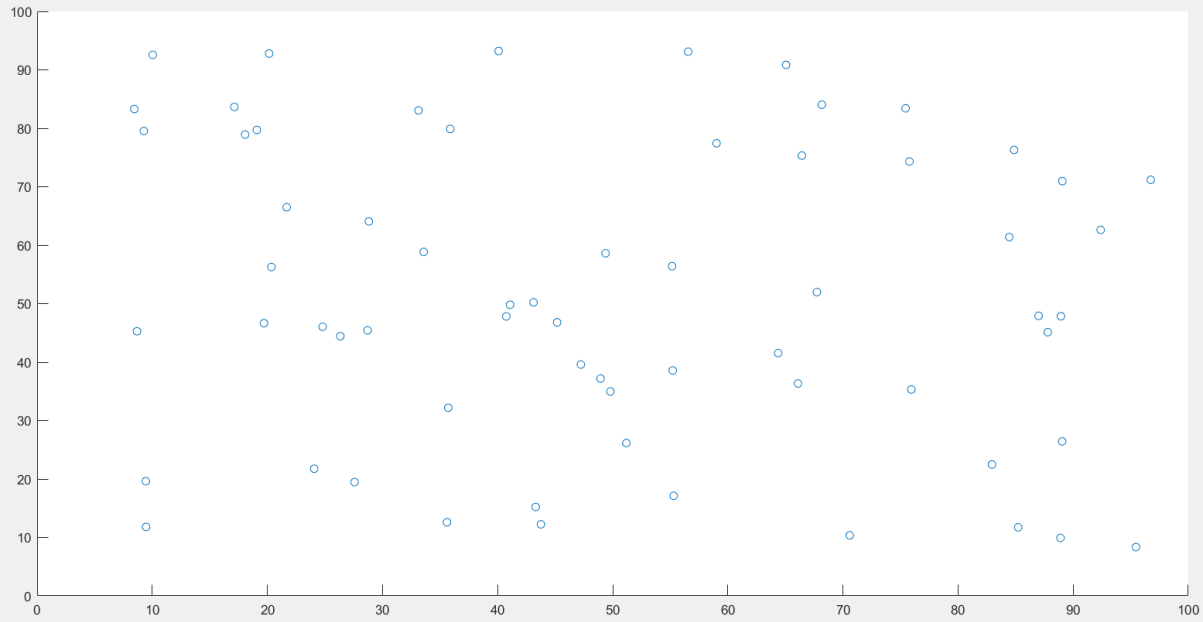
```
% variance = 1;
% T = sqrt(var0 * db2pow(T)) ;
%
% rng default
% N = 720;
% x = sqrt(variance) * randn(N,0.0001);
% falsealarmrate = sum(x > T)/N
% %Pdet =  [0:0.01:1];
% %plot(Pdet,Pfa)
% x = (T - mean0) / var1 ;
% x
% fun = @(x) ( exp(-x.^2/2) )
% Pdet = (1/sqrt(2*pi)) * integral(fun,x,Inf)
% fplot(Pdet)
% %plot(Pdet,Pfa)
% %line([0 Pfa],[T,T],'Color','red')
% xlabel('Pfa')
% ylabel('Pdet')

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------
```

CHAPTER 5

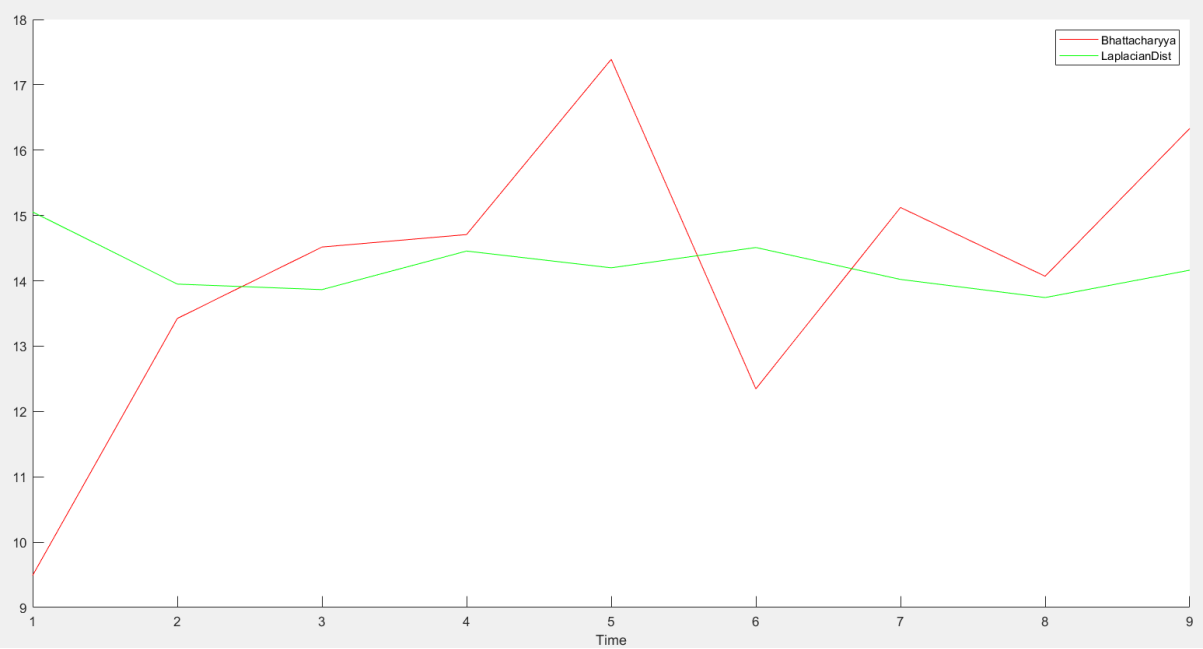
SNAPSHOTS



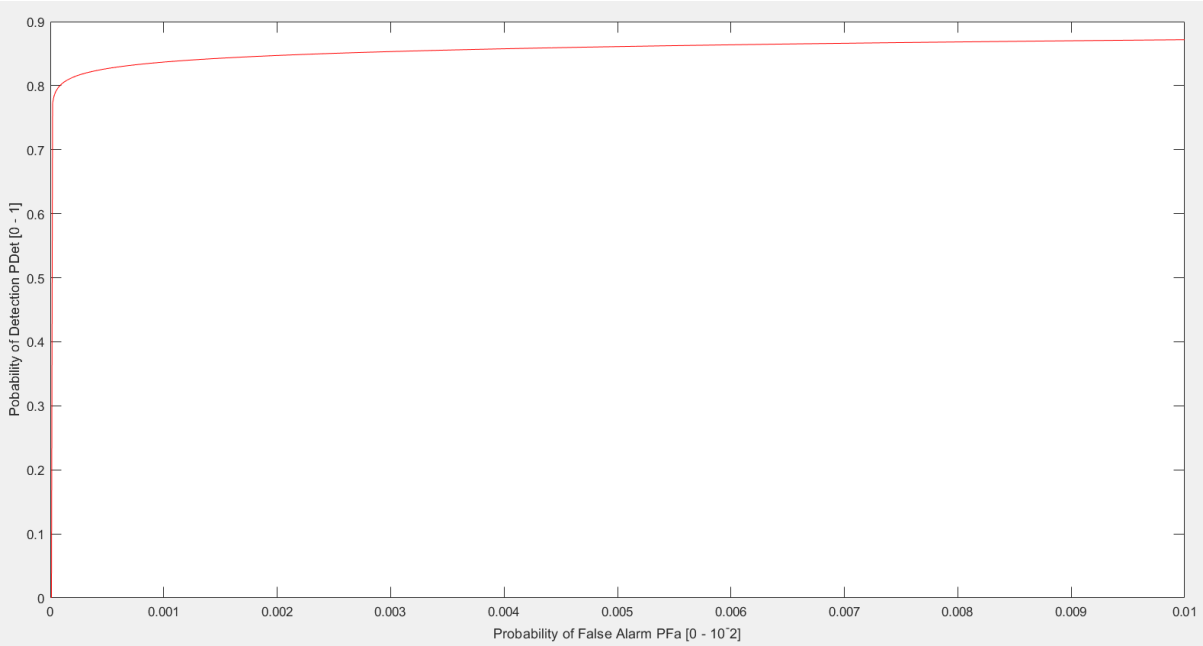
Visualization of first time instance signals



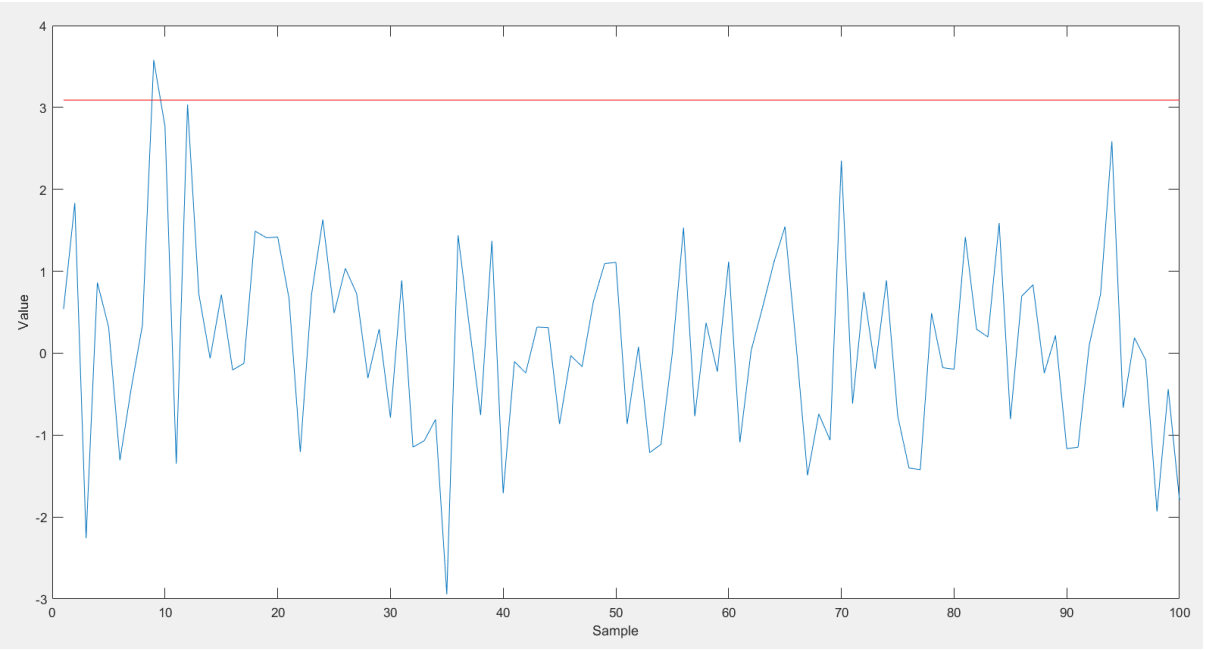
Sensor node coordinates



Bhattacharyya vs Laplacian distance



ROC curve of proposed system



Neyman-Pearson criteria check

CHAPTER 6

CONCLUSION AND FUTURE PLANS

The experimental ROC curves are determined by applying Monte Carlo simulations which are mathematical algorithms to generate random sample data based on some known distribution which in this case are the Laplacian matrices formed from GSP for numerical experiments. The paper ran 1000 pseudo random sequences but due to the limitations of our device we couldn't run that many at a time. The ROC curve denoted the intrusion detection method over the probability of a false alarm (x -axis) and probability of detection of an anomaly (y-axis)

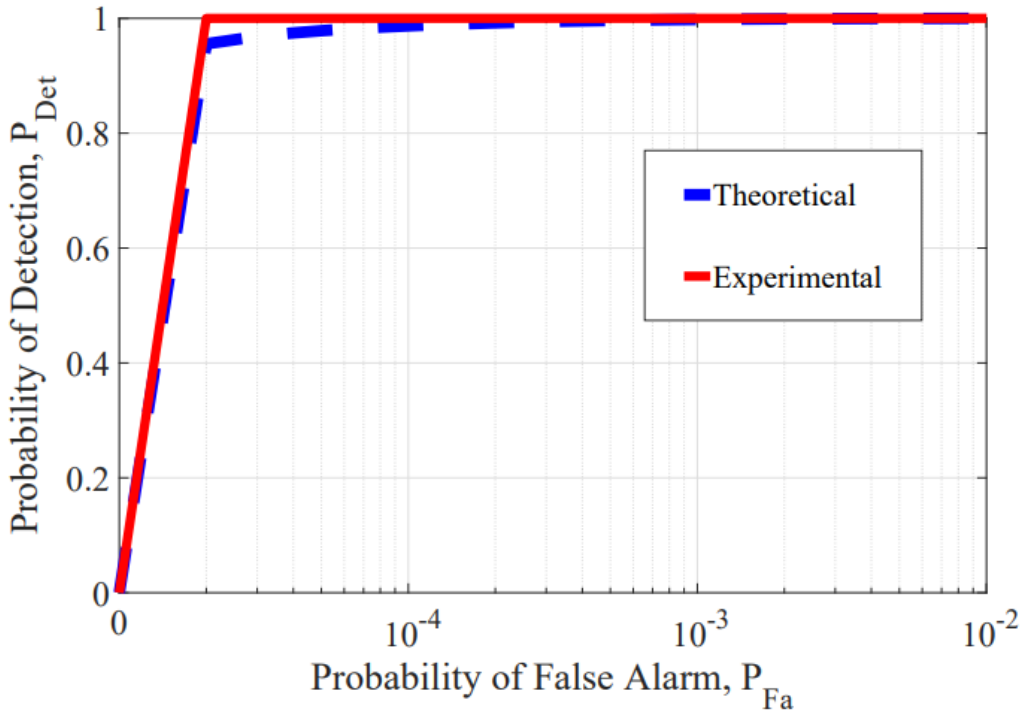


Fig 4

The implementation of this paper shows an intrusion detection system with respect to the authors own GMRF model, what we would like to do is compare this model with respect to other famous models such as Principal Component Analysis (PCA) and Graph Based Filtering (GBF). We compare these based on the respective ROC, what we would like to infer from this comparison is that the proposed model is the most ideal model for a detection of an anomaly.

CHAPTER 7

REFERENCES

http://markummitchell.github.io/engauge-digitizer/

https://online.stat.psu.edu/stat415/lesson/26/26.1

https://www.mathworks.com/matlabcentral/fileexchange/70567-graph_signal_processing

https://mathworld.wolfram.com/FrobeniusNorm.html

CHAPTER 8

APPENDIX

Title: Distributed Graph-based Statistical Approach for Intrusion Detection in Cyber-Physical Systems

Journal Name: IET CYBER-PHYSICAL SYSTEMS: THEORY & APPLICATIONS

Publisher: IEEE

Year: 2018

Link: https://ieeexplore.ieee.org/document/8027126