ICT for Health
Laboratory # 4
Detect activities

Monica Visintin

Politecnico di Torino

ICT4SS
Information and Communications Technologies for Smart Societies

December 4th 2023

Table of Contents

Description of laboratory # 4

View the data

Preliminary analysis

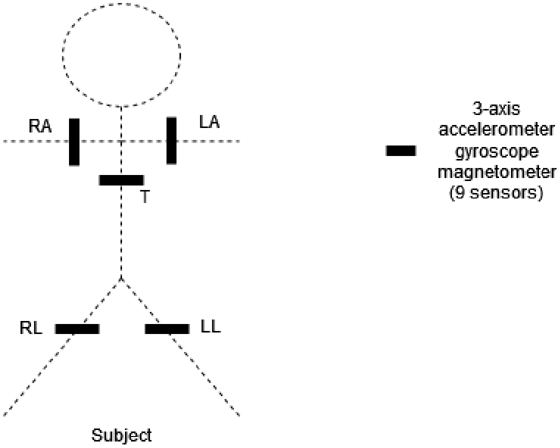
True clustering

What do you have to do?

Description [1]

In fitness, people would like to automatically detect their current activity, in order to have a more precise evaluation of the calories, etc.
Task of this lab is to detect the activity (one among 19) starting from the samples taken at frequency 25 Hz from 3-axes accelerometers, gyroscopes, magnetometers placed on torso, right arm, left arm, right leg, left leg: 5 positions, 3 sensors per position, each measuring 3 values on the x-y-z coordinates, total of 45 values for each sample (`https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities`).
Data were taken from 8 subjects; each subject practised the activity for 5 minutes and slices/files of 5 seconds were created (each slice/file contains 125 rows and 45 columns). We will work on subject number 5.

Description [2]



Description [3]

We want a simple algorithm to detect the activity (not neural networks, taught in the parallel class), valid for a specific subject. So we do not want an algorithm trained on many people and valid for any person, but an algorithm that works on a given subject. The idea is to use the clustering algorithm **K-Means**:

- Read the first M slices for the 19 activities of the chosen subject.
- Apply K-Means with 19 clusters and find the 19 centroids.
- For each row of the subsequent slices detect the activity as the one with centroid at **minimum distance** from the row.

This lab is used just as a **feasibility study**. In the real life, an Android/IOs application should be written, that runs on the subject smartphone and gets the data from sensors and makes decisions. The application should have a first training phase in which the subject stores the data, labels them with the activity and applies K-Means. Note that we use a **clustering algorithm** to solve a **classification problem**.
K-Means is very simple and less energy is necessary to perform the computations, batteries last longer, less required CPU, etc.

Description [4]

In this project we have many degrees of freedom, but we can reduce them by critically thinking of the context in which the system (sensors plus detection algorithm) should be used.

1. **Less than 15 sensors/45 features can be used: the fewer, the better** because the subject does not need to wear all of them, which might be the reason the subject does not want to use the system. It is better to remove all the sensors in one of the 5 positions on the body, than to remove one sensor in one position and another sensor in another position.
2. Three-axis accelerometers are not more expensive than one-axis accelerometers, so the fact that you use the accelerations in one direction only does not help much on the overall system cost. On the contrary, if you can eliminate the entire 3-axis accelerometer in a given position on the body, the overall cost will be reduced.
3. Data can/should be **processed** before application of K-Means (for example, through smoothing, enhancing peaks, reshaping, etc).
4. A decision about the activity can be made at each timestamp (25 decisions per second) or every few timestamps (for example one decision per second is more than enough) for the given application.

Description [5]

5. In total there are 60 slices (5 seconds each) for each user. The shorter the training phase, the better. In the training phase the subject has to tell the system which activity he/she was doing. Only after the ytraining phase can the system provide correct decisions. The training phase cannot last more than 30 slices (i.e. 3 minutes).

Table of Contents

Description of laboratory # 4

View the data

Preliminary analysis

True clustering

What do you have to do?

Read the data [1]

- In DropBox you find the zipped file with the time series. You can download the data from `https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities`.
- Copy the file on your PC, extract the folders and files.
- In DropBox you also find a Python script that allows you to generate a Pandas dataframe that includes the selected slices of the selected activities of the selected subject.

Read the data [2]

```
def generateDF(filedir,colnames,patients,activities,slices):
    # get the data from files for the selected patients
    # and selected activities
    # concatenate all the slices
    # generate a pandas dataframe with an added column: activity
    x=pd.DataFrame()
    for pat in patients:
        for a in activities:
            subdir=’a’+f”{a:02d}”+’/p’+str(pat)+’/’
            for s in slices:
                filename=filedir+subdir+’s’+f”{s:02d}”+’.txt’
                x1=pd.read_csv(filename,names=colnames)
                x1[’activity’]=a*np.ones((x1.shape[0],),dtype=int)
                x=pd.concat([x,x1], axis=0, join=’outer’,
                            ignore_index=True,
                            keys=None, levels=None, names=None,
                            verify_integrity=False,
                            sort=False, copy=True)
    return x
```

Read the data [3]

- Activities The 19 activities are:

```
actNames=[
    'sitting',  # 1
    'standing', # 2
    'lying on back',# 3
    'lying on right side', # 4
    'ascending stairs' , # 5
    'descending stairs', # 6
    'standing in an elevator still', # 7
    'moving around in an elevator', # 8
    'walking in a parking lot', # 9
    'walking on a treadmill with a speed of 4 km/h in flat', # 10
    'walking on a treadmill with a speed of 4 km/h in 15 deg inclined position', # 11
    'running on a treadmill with a speed of 8 km/h', # 12
    'exercising on a stepper', # 13
    'exercising on a cross trainer', # 14
    'cycling on an exercise bike in horizontal positions', # 15
    'cycling on an exercise bike in vertical positions', # 16
    'rowing', # 17
    'jumping', # 18
    'playing basketball' # 19
   ]
```

Read the data [4]

- Short names for activities:

```
actNamesShort=[
        'sitting',   # 1
        'standing', # 2
        'lying.ba', # 3
        'lying.ri', # 4
        'asc.sta'  , # 5
        'desc.sta', # 6
        'stand.elev', # 7
        'mov.elev', # 8
        'walk.park', # 9
        'walk.4.fl', # 10
        'walk.4.15', # 11
        'run.8', # 12
        'exer.step', # 13
        'exer.train', # 14
        'cycl.hor', # 15
        'cycl.ver', # 16
        'rowing', # 17
        'jumping', # 18
        'play.bb' # 19
        ]
```

Read the data [5]

- The 45 sensors

```
sensNames=[
          'T_xacc',  'T_yacc',  'T_zacc',
          'T_xgyro','T_ygyro','T_zgyro',
          'T_xmag',  'T_ymag',  'T_zmag',
          'RA_xacc',  'RA_yacc',  'RA_zacc',
          'RA_xgyro','RA_ygyro','RA_zgyro',
          'RA_xmag',  'RA_ymag',  'RA_zmag',
          'LA_xacc',  'LA_yacc',  'LA_zacc',
          'LA_xgyro','LA_ygyro','LA_zgyro',
          'LA_xmag',  'LA_ymag',  'LA_zmag',
          'RL_xacc',  'RL_yacc',  'RL_zacc',
          'RL_xgyro','RL_ygyro','RL_zgyro',
          'RL_xmag',  'RL_ymag',  'RL_zmag',
          'LL_xacc',  'LL_yacc',  'LL_zacc',
          'LL_xgyro','LL_ygyro','LL_zgyro',
          'LL_xmag',  'LL_ymag',  'LL_zmag']
```

Read the data [6]

- Folder with data:

```
filedir='../data/'
```

Modify `filedir` so that it matches the position you have the data.

Read the data [7]

- Settings:

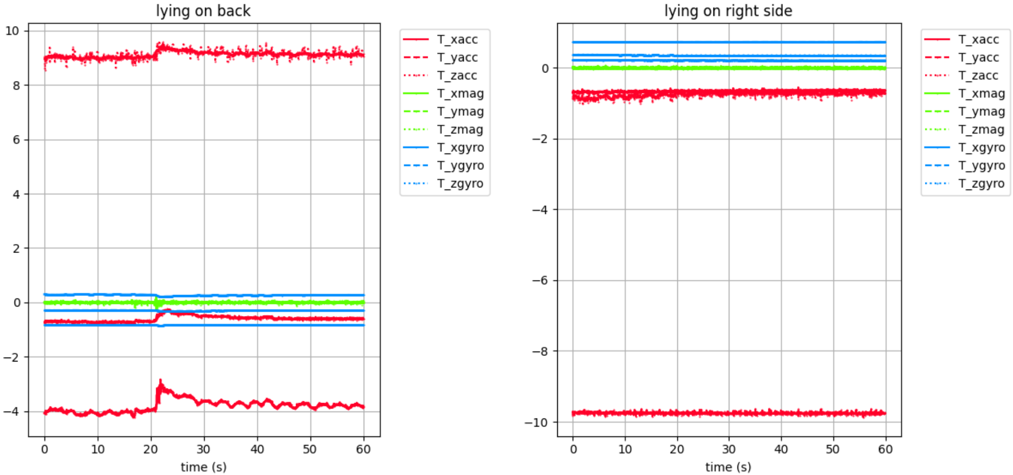
```
patients=[5]  # list of selected patients
NAc=19 # total number of activities
activities=list(range(1,20)) #list of indexes of activities to plot
Num_activities=len(activities)
actNamesSub=[actNamesShort[i-1] for i in activities] # short names of the selected activities
ind_acc = [i for i in range(45) if 'acc' in sensNames[i]]# accelerometers
ind_mag = [i for i in range(45) if 'mag' in sensNames[i]]# magnetometers
ind_gyr = [i for i in range(45) if 'gyr' in sensNames[i]]# gyroscopes
sensors1=ind_acc+ind_mag+ind_gyr
sensorsT = [i for i in sensors1 if 'T' in sensNames[i]] # sensors on torso
sensorsLL = [i for i in sensors1 if 'LL' in sensNames[i]]# sensors on left leg
sensorsLA = [i for i in sensors1 if 'LA' in sensNames[i]]# sensors on left arm
sensorsRL = [i for i in sensors1 if 'RL' in sensNames[i]]# sensors on right leg
sensorsRA = [i for i in sensors1 if 'RA' in sensNames[i]]# sensors on right arm
sensors=sensorsT#+sensorsRA+sensorsLL # selected sensors
sensNamesSub=[sensNames[i] for i in sensors] # names of selected sensors
Nslices=12 # number of slices to plot and to train the system
slices=list(range(1,Nslices+1))# first Nslices to plot
NtotSlices=60 #total number of slices
fs=25 # Hz, sampling frequency
samplesPerSecond=fs # samples in one second
samplesPerSlice=fs*5 # samples in each slice
```

Read the data [8]

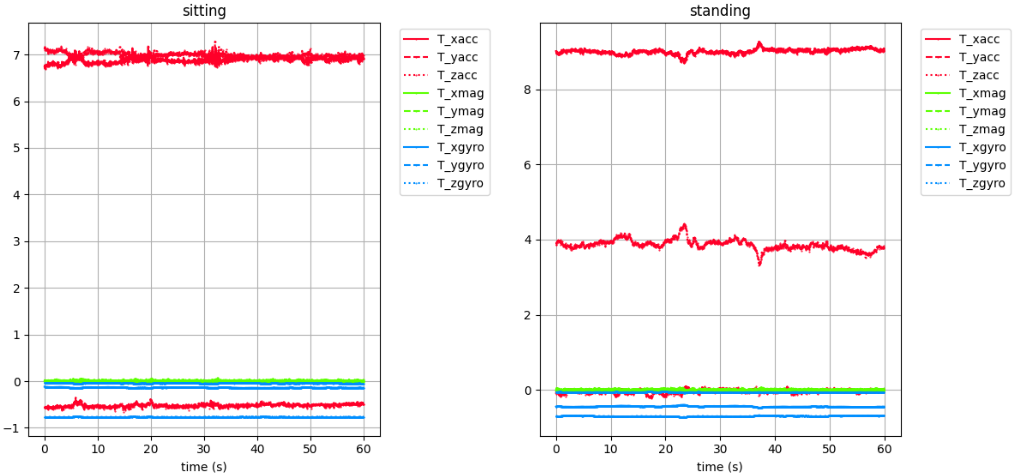
In the example above only the torso sensors and all the activities are considered, only the first 12 slices (60 seconds) are shown.

In the following slides you can see the plots of the original data.

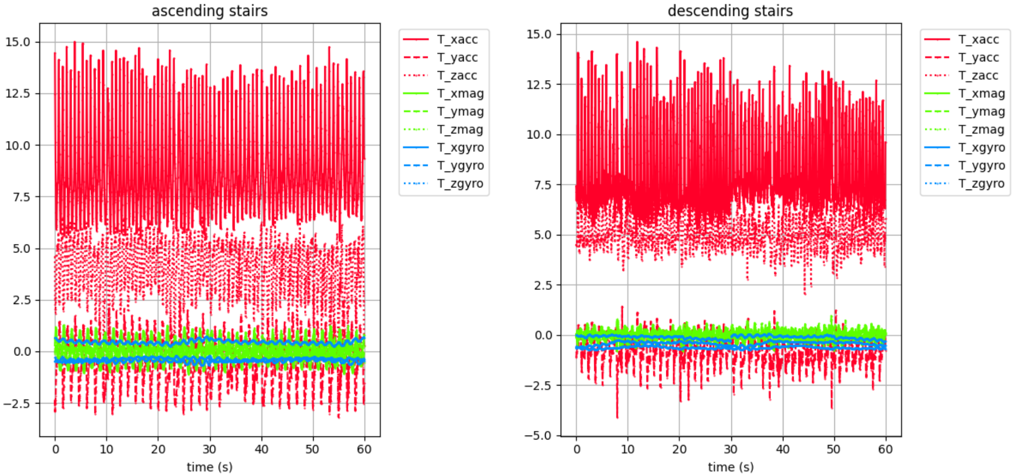
Read the data [9]



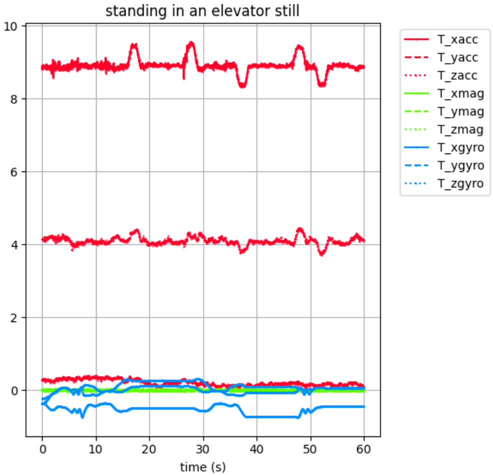
Read the data [10]

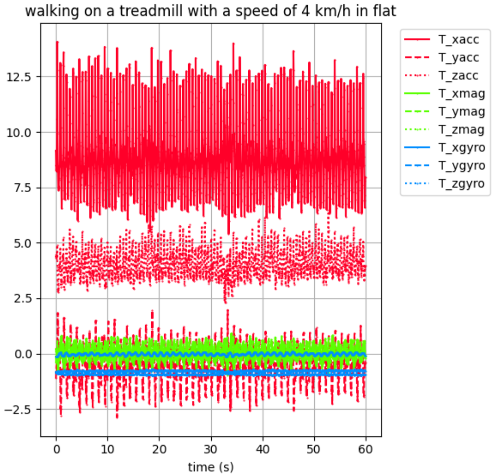


Read the data [11]

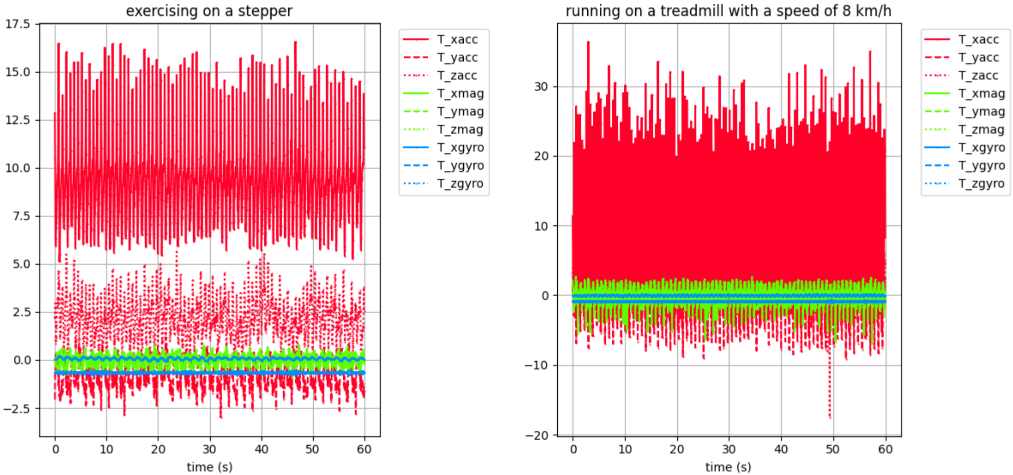


Read the data [12]

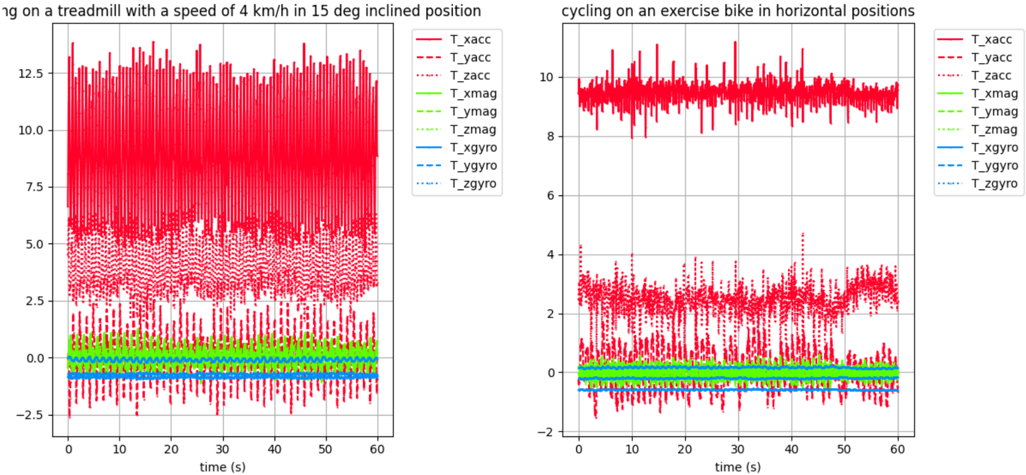




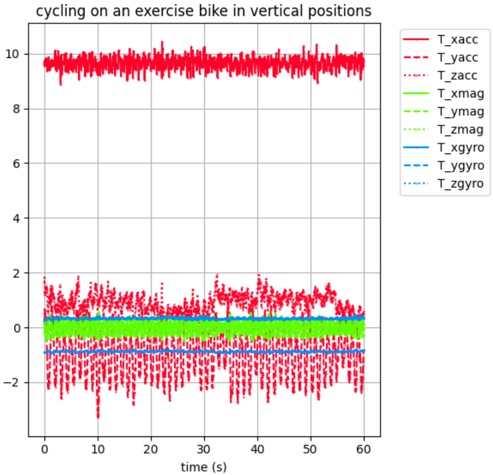
Read the data [13]

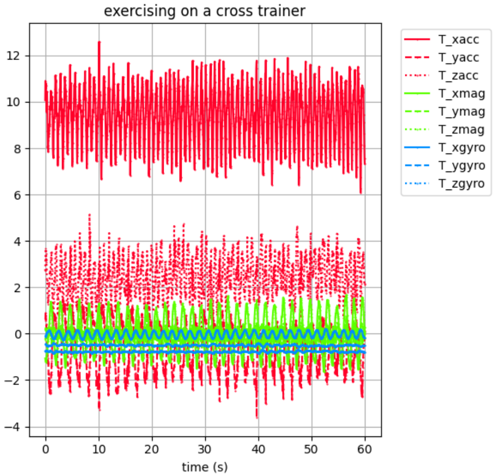


Read the data [14]

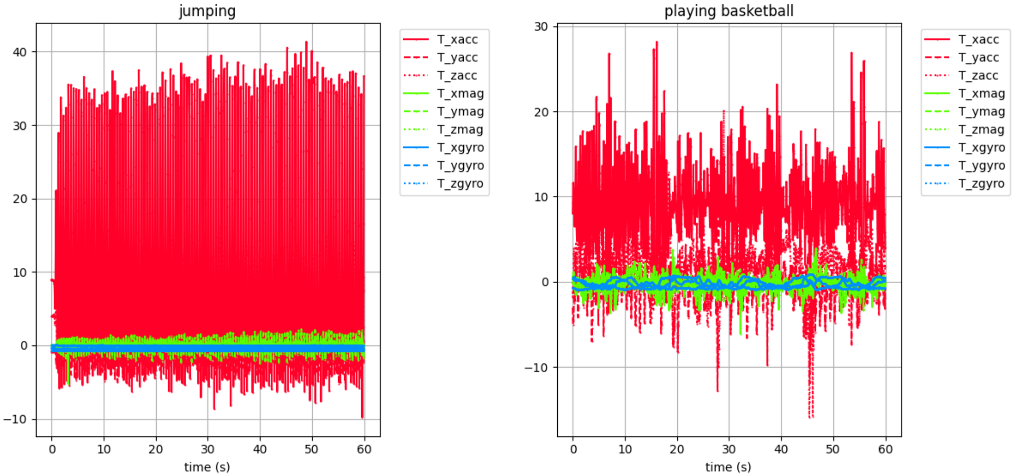


Read the data [15]

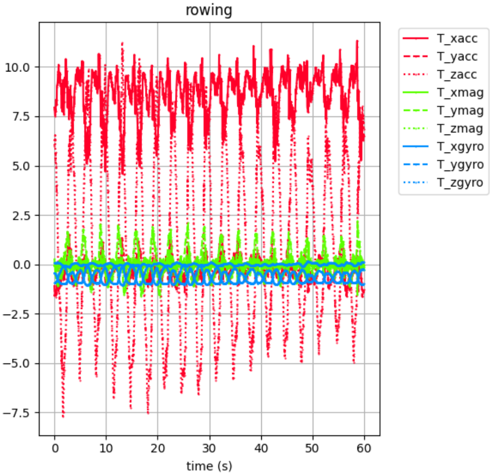

cycling on an exercise bike in vertical positions


exercising on a cross trainer

Read the data [16]



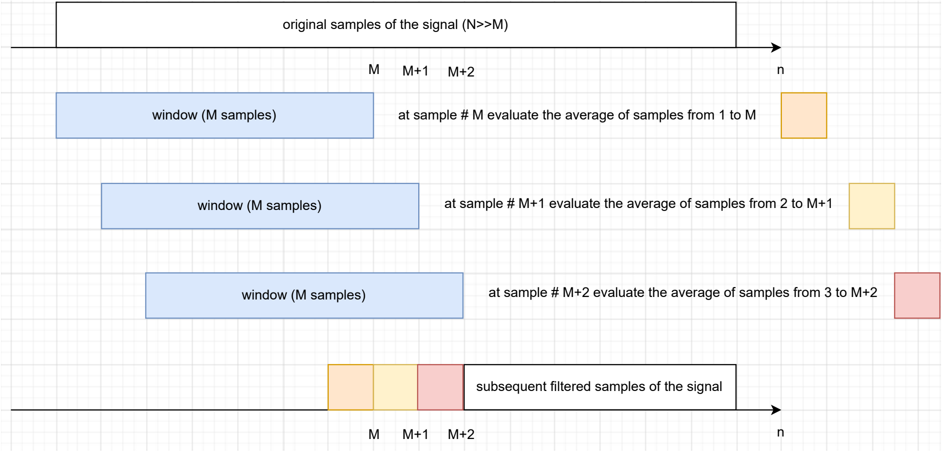
Read the data [17]



A first observation [1]

Some activities clearly generate large variations in the signals of the accelerometer. If we sample the signals as they are, it is highly probable that we cannot distinguish one activity from another because locally they might have the same values of accelerations, even if the signal is “visually” different.
If signals are **filtered**, the variations are removed, but then we miss the amplitude of the variations.
The **idea** is then the following: instead of working with the original data, let us work on the **average signal** (over a specified time interval) and on its **standard deviation** (over the same interval).

A first observation [2]


original samples of the signal (N>>M)
M
M+1
M+2
n
window (M samples)
at sample # M evaluate the average of samples from 1 to M
window (M samples)
at sample # M+1 evaluate the average of samples from 2 to M+1
window (M samples)
at sample # M+2 evaluate the average of samples from 3 to M+2
subsequent filtered samples of the signal
M
M+1
M+2
n

Note that the first $M-1$ samples are missing from the filtered signal (you need at least M samples to generate the first filtered sample).

A first observation [3]

For $k = 0, 1, 2, \ldots$

$$x_m(M+k-1) = \frac{1}{M} \sum_{n=k}^{k+M-1} x(n)$$

$$x_{std}(M+k-1) = \sqrt{\frac{1}{M} \sum_{n=k}^{k+M-1} [x(n) - x_m(k)]^2}$$

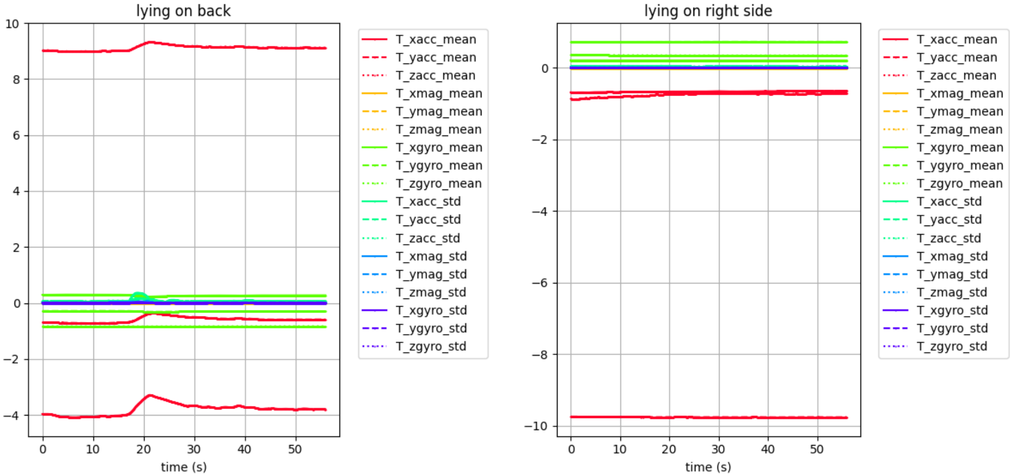
A first observation [4]

```
def genFeatures(x,N):
    # the input is the output of generateDF (including the activity name)
    act=x.activity[0]
    y=x.drop(columns=['activity'])
    cols0=y.columns
    cols1=cols0+'_mean'
    cols2=cols0+'_std'
    y=y.values # get the values of dataFrame x
    Nin=y.shape[0]# number of points in the input dataframe
    Nout=Nin-N # number of valid points in the output dataframe
    yav_m=np.zeros((Nin,y.shape[1]))
    yav_std=np.zeros((Nin,y.shape[1]))
    for k in range(Nout):
        temp=y[k:k+N,:]# get the k-th window with N samples
        yav_m[k+N,:]=np.mean(temp,0)
        yav_std[k+N,:]=np.std(temp,0)
    yav_m=yav_m[N: ,:]
    yav_std=yav_std[N: ,:]
    outm_df=pd.DataFrame(yav_m,columns=cols1)
    outs_df=pd.DataFrame(yav_std,columns=cols2)
    out_a=pd.DataFrame(act*np.ones((Nout,1)),columns=['activity'])
    out_df=pd.concat([outm_df,outs_df,out_a],axis=1)
    return out_df
```

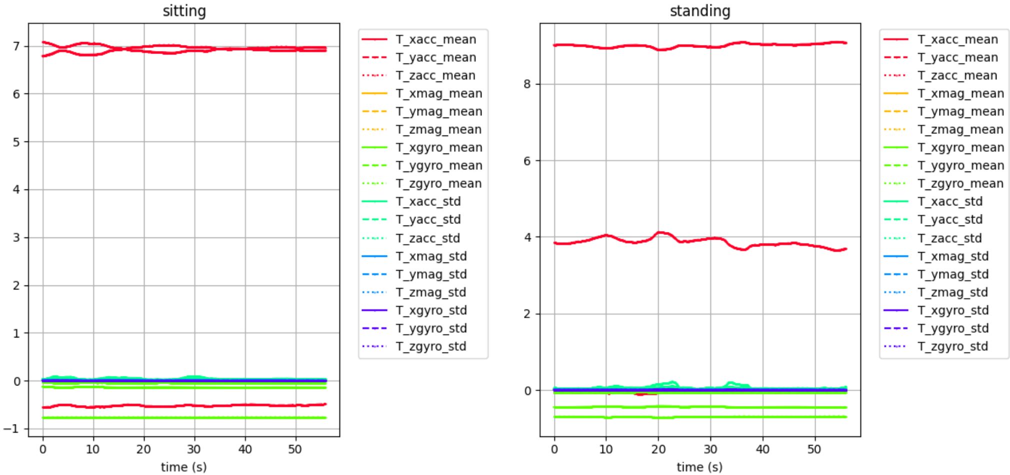
A first observation [5]

In the following slides, you can see the plots when the time interval for the averaging operation M is equal to 4 seconds.

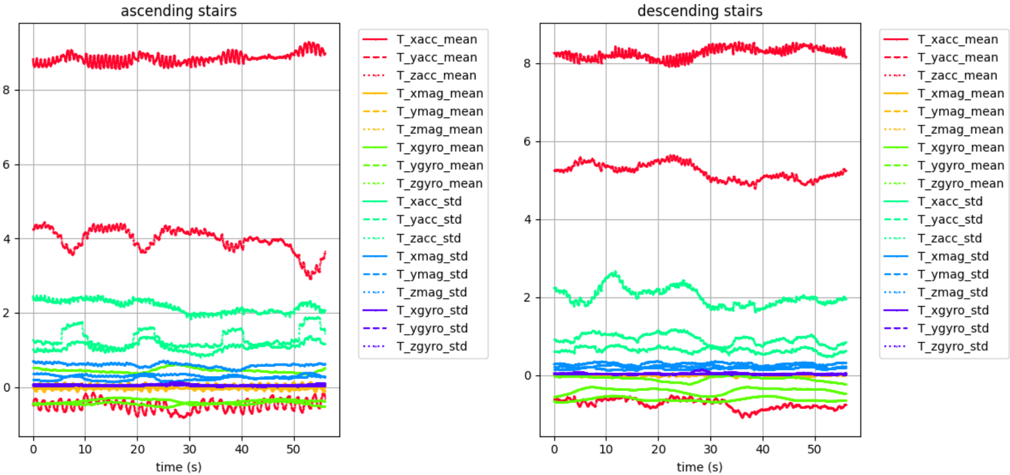
A first observation [6]



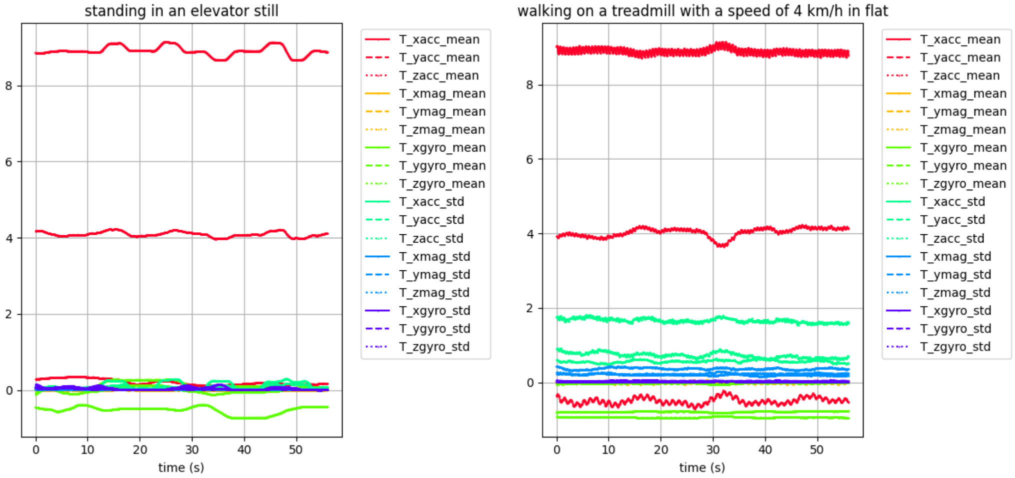
A first observation [7]



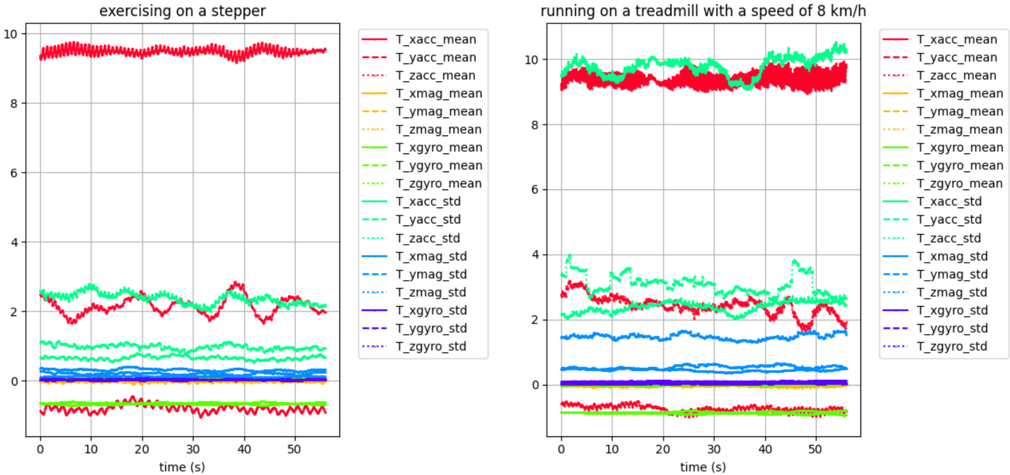
A first observation [8]



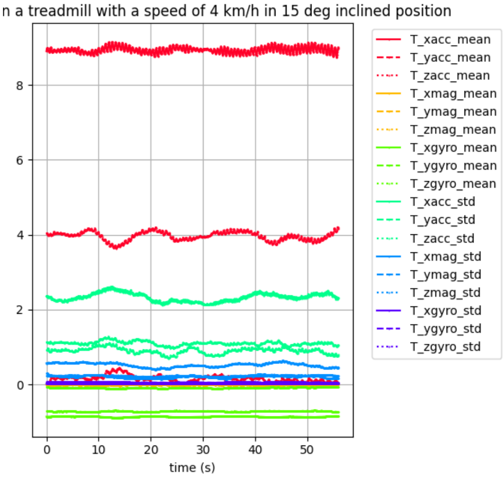
A first observation [9]

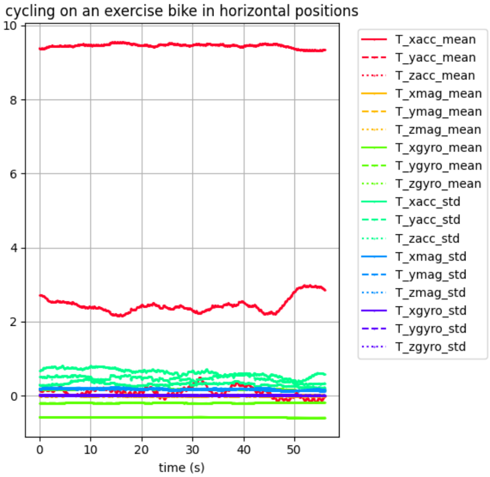


A first observation [10]

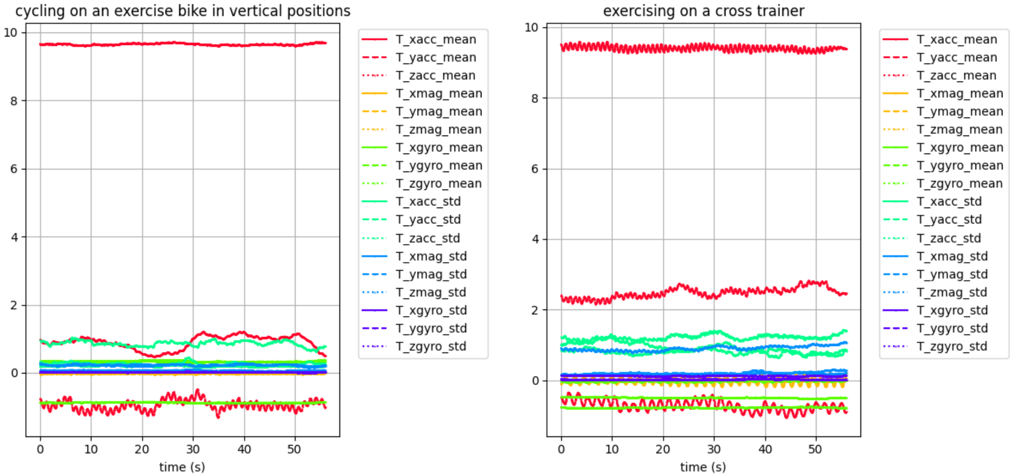


A first observation [11]

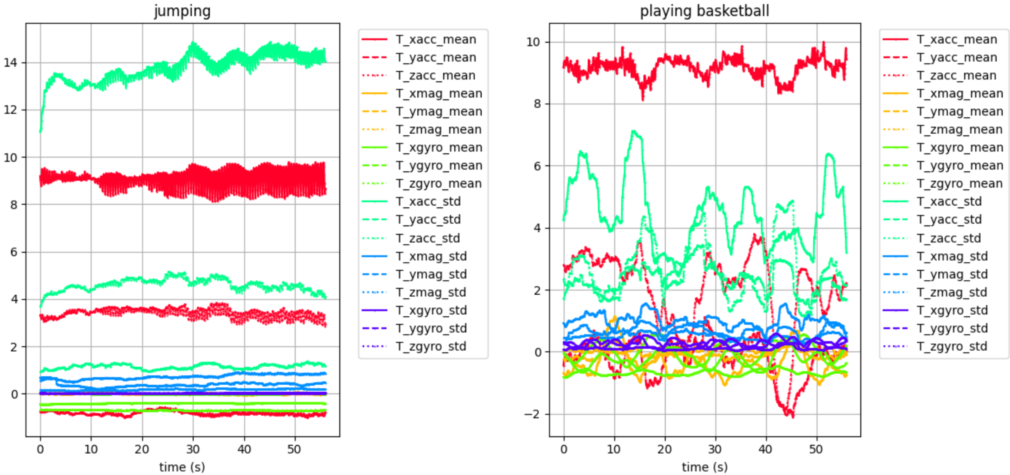




A first observation [12]



A first observation [13]



A first observation [14]

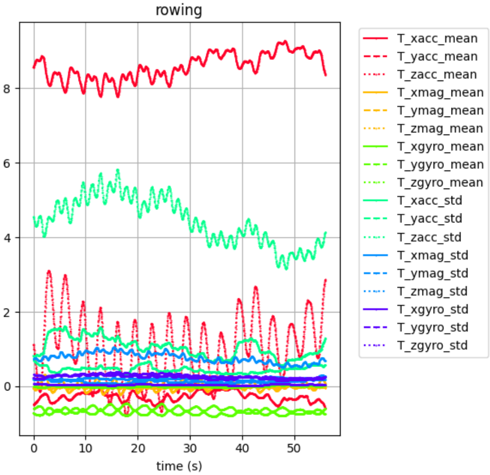


Table of Contents

Description of laboratory # 4

View the data

Preliminary analysis

True clustering

What do you have to do?

“Cluster” analysis [1]

- In this lab we know what we would like K-Means to give us: each cluster should include exactly all the rows/samples of one and only one activity.
- It is therefore useful to start **analyzing the data as they are**, assuming that K-Means will then be able to find these correct/ideal clusters. If everything is OK, K-Means will automatically find the same results.
- For each activity (which we know exactly at this stage), we evaluate the centroid/mean of the data for each sensor. We also measure the standard deviations for each sensor and for each activity, because this tells us how much the points are distant from the mean (for each sensor/feature).

“Cluster” analysis [2]

All the activities are here considered, but only 9 features are used:

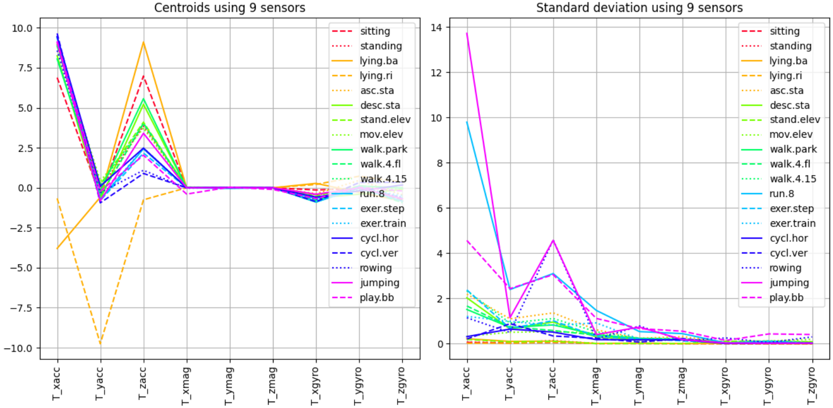
```python
#%% plot centroids and stand. dev. of sensor values
print('Number of used sensors: ',len(sensors))
centroids=np.zeros((NAc,len(sensors)))# centroids for all the activities
stdpoints=np.zeros((NAc,len(sensors)))# variance in cluster for each sensor
plt.figure(figsize=(12,6))
for i in range(1,NAc+1):
    activities=[i]
    x=generateDF(filedir,sensNamesSub,patients,activities,slices)
    x=x.drop(columns=['activity'])
    centroids[i-1,:]=x.mean().values
    plt.subplot(1,2,1)
    lines = plt.plot(centroids[i-1,:],label=actNamesShort[i-1])
    lines[0].set_color(cm(i//3*3/NAc))
    lines[0].set_linestyle(line_styles[i%3])
    stdpoints[i-1]=np.sqrt(x.var().values)
    plt.subplot(1,2,2)
    lines = plt.plot(stdpoints[i-1,:],label=actNamesShort[i-1])
    lines[0].set_color(cm(i//3*3/NAc))
    lines[0].set_linestyle(line_styles[i%3])
plt.subplot(1,2,1)
plt.legend(loc='upper right')
plt.grid()
plt.title('Centroids using '+str(len(sensors))+' sensors')
```

“Cluster” analysis [3]

```
plt.xticks(np.arange(x.shape[1]),list(x.columns),rotation=90)
plt.subplot(1,2,2)
plt.legend(loc='upper right')
plt.grid()
plt.title('Standard deviation using '+str(len(sensors))+' sensors')
plt.xticks(np.arange(x.shape[1]),list(x.columns),rotation=90)
plt.tight_layout()
```

“Cluster” analysis [4]

Case of no filtering

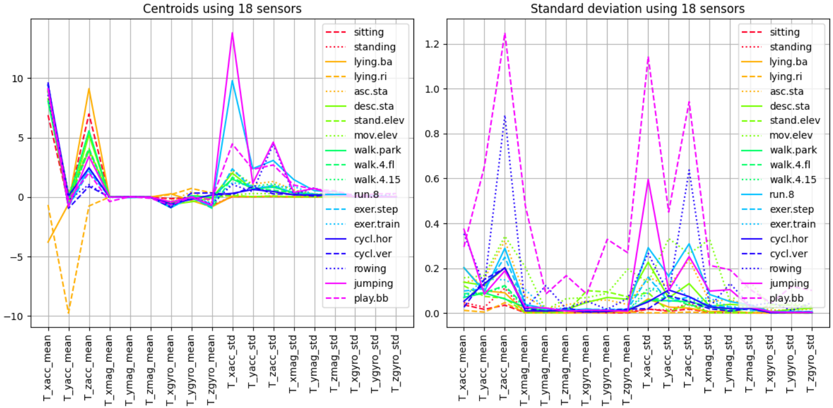


Centroids using 9 sensors

Standard deviation using 9 sensors

“Cluster” analysis [5]

Case of filtering (4 seconds). Note that the centroids are more separated.



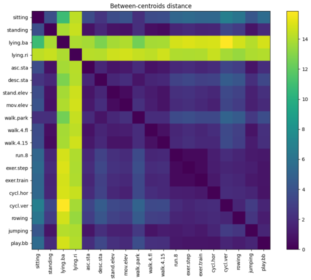
“Cluster” analysis [6]

- Let us now find the distance among the centroids. The centroids should be well separated, so that the subsequent classification can be more easily and correctly accomplished.

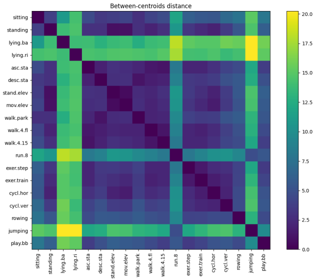
```
#%% between centroids distance
d=np.zeros((NAc,NAc))
for i in range(NAc):
    for j in range(NAc):
        d[i,j]=np.linalg.norm(centroids[i]-centroids[j])
if iplot:
    plt.figure(figsize=(12,10))
    plt.matshow(d,fignum=0)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar()
    plt.xticks(np.arange(NAc),actNamesShort,rotation=90)
    plt.yticks(np.arange(NAc),actNamesShort)
    plt.subplots_adjust(bottom=0.2)
    plt.title('Between-centroids distance')
```

“Cluster” analysis [7]

without filter



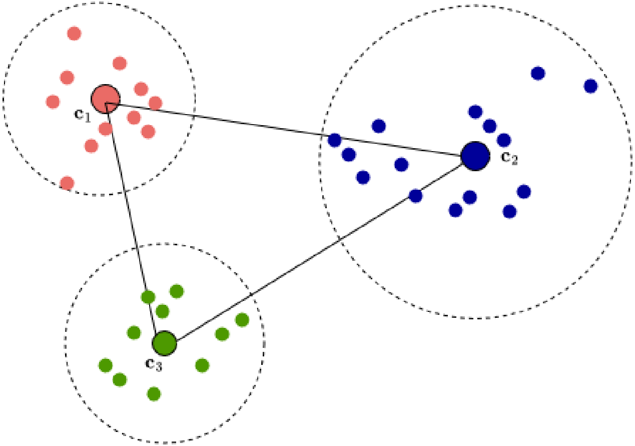
with filter



Without filtering, it would be very difficult to distinguish the activities from ascending the stairs to exercising on a cross trainer. With filtering, distances between couples of centroids are larger and more evenly distributed: running, jumping, lying should be more easily distinguished.

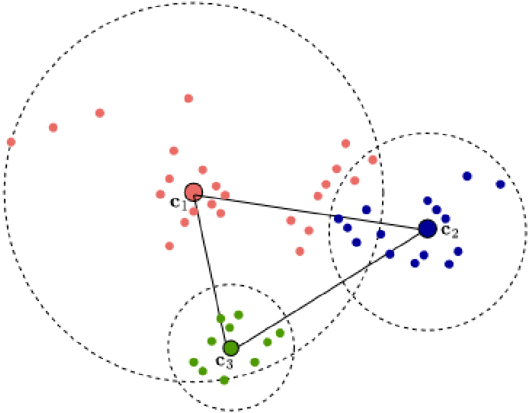
“Cluster” analysis [8]

Desired case: clusters are far apart and points are close to their centroids. K-Means can work correctly.



“Cluster” analysis [9]

Undesired case: points are very spread and distant from centroids. K-Means cannot work correctly (it finds other centroids and clusters).



“Cluster” analysis [10]

- For each activity find the minimum distance between the corresponding centroid and the other centroids:

```
dd=d+np.eye(NAc)*1e6# remove zeros on the diagonal (distance of centroid from itself)
dmin=dd.min(axis=0)# find the minimum distance for each centroid
```

- Find the average distance of the points of a cluster from their centroid. The average square distance is

$$d^2 = \frac{1}{N}\sum_{n=1}^{N}\sum_{f=1}^{F}(\mathbf{x}_{nf} - \mathbf{c}_f)^2$$

  where N is the number of points in the cluster, F is the number of features/columns, $\mathbf{c}$ is the centroid, $\mathbf{x}_n$ is the n-th point in the cluster, $\mathbf{x}_{nf}$ is the value of the f-th feature of the n-th point. Note that

$$d^2 = \sum_{f=1}^{F}\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_{nf} - \mathbf{c}_f)^2 = \sum_{f=1}^{F}\mathrm{var}_f$$

“Cluster” analysis [11]

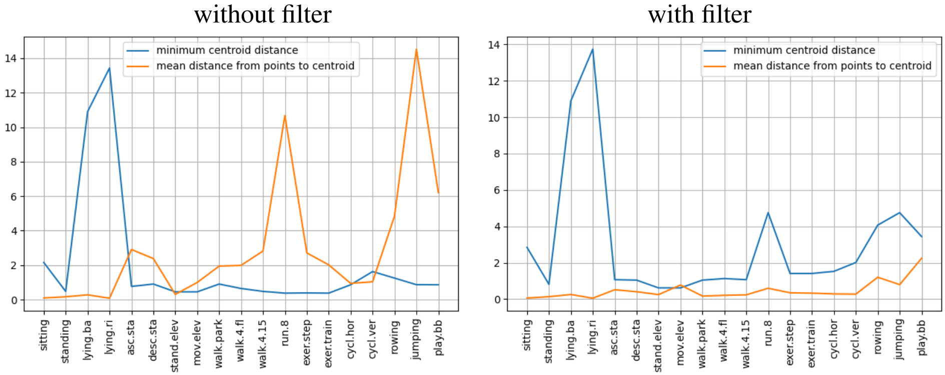
where var_f is the variance of feature f in the cluster. Therefore, the average distance can be evaluated as

$$d = \sqrt{\sum_{f=1}^{F} \text{var}_f}$$

```
dpoints=np.sqrt(np.sum(stdpoints**2,axis=1))
```

If the mean distance of points from the centroid is higher than the minimum distance between the centroid and the other centroids, then we are in the undesired case and the corresponding activity cannot be detected.

“Cluster” analysis [12]



without filter

with filter

Without filter, only the first 4 activities can be easily detected. With the filter, the situation is much better, even if errors can still be made (we are simply considering the standard deviation not the maximum distance between a point in a cluster and its centroid). Note that only torso sensosrs were used to generate these figures.

Table of Contents

Description of laboratory # 4

View the data

Preliminary analysis

True clustering

What do you have to do?

K-Means [1]

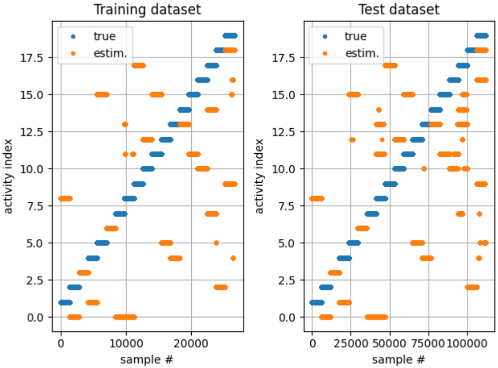
With the preliminary analysis, we saw the importance of filtering and generating new features, based on the standard deviation over time of the original features. Now we can apply K-Means and see if it is able to distinguish all the activities (equivalently, if K-Means is able to find the theoretical centroids that we used in the previous, theoretical, analysis).
We consider a training dataset made of the first 12 slices (60 seconds) and we generate the dataset by appending all the samples of all the activities one after another in a Numpy 2D-array. Then this 2D-Array is given to scikit-learn `KMeans` to get the centroids.

```
#%%generate the training and test datasets by concatenating data for each activity
slicesTrain=list(range(1,Nslices+1))# slices for the training part
slicesTest=list(range(Nslices+1,NtotSlices+1))# first for the test part
Ntrain=Nslices*samplesPerSlice
Ntest=(NtotSlices-Nslices)*samplesPerSlice
dataTrain=np.empty((0,len(sensors)*2), float)
dataTest=np.empty((0,len(sensors)*2), float)
labelTrain=np.empty((0,), float)
labelTest=np.empty((0,), float)for i in range(1,NAc+1):
    activ=[i]
    x=generateDF(filedir,sensNamesSub,sensors,patients,activ,slicesTrain)
    x=genFeatures(x,memory)
    labels=x.activity.values
    x=x.drop(columns=['activity'])
```

K-Means [2]

```
    data=x.values
    dataTrain=np.vstack((dataTrain,data))
    labelTrain=np.hstack((labelTrain,labels))
    x=generateDF(filedir,sensNamesSub,sensors,patients,activ,slicesTest)
    x=genFeatures(x,memory)
    labels=x.activity.values
    x=x.drop(columns=['activity'])
    data=x.values
    dataTest=np.vstack((dataTest,data))
    labelTest=np.hstack((labelTest,labels))
#%% Use K-Means
labsTrue=np.unique(labelTrain)
Nclu=len(labsTrue)
labsTrue=np.unique(labelTrain)
Nclu=len(labsTrue)
clu=KMeans(n_clusters=Nclu,init='k-means++', n_init=10,
           max_iter=300,tol=0.0001, verbose=0,
           random_state=1, copy_x=True)
clu.fit(dataTrain)
labCluTrain=clu.labels_
labsClu=np.unique(labCluTrain)
labCluTest=clu.predict(dataTest)
```

K-Means [3]



Problem: the cluster indexes are randomly generated (from 0 to 18) by K-Means, which means that cluster number 1 not necessarily corresponds to activity number 1. How can we understand if K-Means is working correctly?

Table of Contents

Description of laboratory # 4

View the data

Preliminary analysis

True clustering

What do you have to do?

To do [1]

1. Solve the problem in the previous slide (any solution is accepted).
2. For both training and test datasets, find the confusion matrix that gives

   $$P(i|j) = P(\text{detected activity is } i|\text{the true activity is } j)$$

   in row j column i, for $i = 1, \dots, 19$ and $j = 1, \dots, 19$. Index i corresponds to activity i, index j corresponds to activity j. These values must be visible in a plot (matrix/image).
3. For both training and test datasets, find the accuracy:

   $$\text{Accuracy} = \sum_{i=1}^{19} P(i|i)P(i)$$

   where $P(i)$ is the probability of activity i, $P(i) = 1/19$.
4. Find the accuracy of each activity and generate a plot. Find the two activities that are mostly confused (e.g. sitting and standing are confused).

To do [2]

5. With an appropriate data processing, you should get an accuracy around 0.8 for both training and test data. Change sensors, memory of the filter, number of training slices, add features to improve this result.

You can work in groups, but you might be asked about the implemented solution at the exam, which means that you must understand what the software is doing.