AIR QUALITY AND ACTIVITY DETECTOR

Juan Compadre Ochando

May 2016

Final year project thesis submitted in support of the degree of

Bachelor of Engineering in Electrical Engineering

Department of Electrical & Electronic Engineering

University of Bristol

DECLARATION AND DISCLAIMER

Unless otherwise acknowledged, the content of this thesis is the original work of the author. None of the work in this thesis has been submitted by the author in support of an application for another degree or qualification at this or any other university or institute of learning.

The views in this document are those of the author and do not in any way represent those of the University.

The author confirms that the printed copy and electronic version of this thesis are identical.

Signed: JUAN COMPADRE OCHANDO

Dated: 12/05/2016

ABSTRACT

The aim of this project is to use a CO_2 sensor to monitor the CO_2 concentration in different environments, especially indoors, and extract information related to the air quality and the behaviour of the people inside the monitored environment. This project is composed of two main parts very related.

The first part consists on analysing the air quality performing two experiments. The first experiment is done outdoors and it will analyse the air quality in some streets in the city of Bristol using the CO_2 concentration measured. The second experiment consists on monitoring the CO_2 level inside the Sphere Office, studying the air quality indoors and detecting the daily routine of the people inside this office.

The second part changes the monitoring environment to a room and studies the CO_2 data to detect the activity that the person inside is doing. The amount of CO_2 exhaled by a person depends on the type of physical activity that he is doing. Depending on how the CO_2 concentration changes it is possible to detect different activities, recognizing some characteristic patterns.

Given a set of data obtained with the sensor, a program written in Matlab is implemented in order to detect the activities of the person automatically. This program does not require human interaction, except for the provision of the CO_2 concentration data. The precision obtained with this program varies between the different activities that can be detected. Evaluating the performance for all the activities together, the prediction is correct from 70 % to 90 % of the cases.

CONTENTS

1. INTRODUCTION

During Industrial Revolution, humans started to release large quantities of contaminant gases to the atmosphere. By that time, the adverse effects that this pollution cause on human's health was unknown. In the present, there is an increase awareness about the problems that a large exposure to contaminant gasses can produce in health. The term of air quality is extended as a consequence of the increasing problems caused by this pollution.

Air pollution represents a significant risk factor for respiratory infections, heart disease, lung cancer… Some of the effects caused by air pollution are: difficulty in breathing, coughing and asthma. The main part of the effects are related with the respiratory system and the cardiovascular system. The most common sources of air pollution are:

- **Carbon Monoxide (CO):** Comes from fuel combustion especially in vehicles and engines. Reduces the amount of oxygen reaching the body's organs.
- **Sulphur Dioxide (SO_2):** Also comes from fuel combustion although volcanoes are natural contaminants of this gas. It is known for aggravating asthma and complicating breathing.
- **Nitrogen Dioxide (NO_2)**: It is one of the main causes of the acid rain and contributes to smog formation [1].
- **Methane:** generated mostly in waste depositions, methane is a highly flammable gas that displaces oxygen, leading to asphyxia when its concentration is high enough to reduce oxygen level significantly.

The main part of the processes that generate these contaminants are industries where burning processes are required, vehicles that uses combustion engines, controlled burns on agriculture and waste depositions. All these processes also generate Carbon Dioxide (CO_2), which is a gas naturally present in the atmosphere and vital to life on Earth. The concentration of this gas in the atmosphere is around 400 ppm [2]. However, as breathing produces CO_2, it can be found in a higher concentration inside buildings. CO_2 is not consider toxic although a very high concentration can cause health problems from headache to asphyxia [3].

As CO_2 is generated along with the other contaminant gases, monitoring the levels of CO_2 in areas with industrial activity or heavy traffic (the main part of the cities) will give an estimation of the air quality. It can be used to identify areas specially polluted and to create alerts for very contaminated environments.

In recent years there has been an increasing concern over the effects of indoor air quality on people's health. To improve energy efficiency some changes have been produced in building design that have led to new structures more airtight than older structures. As a result, they provide indoor environments where some contaminants are found in much higher concentrations than outdoors.

However, in buildings situated close to industrial zones or streets with a lot of traffic, the main contributors to some indoor contaminants can be outdoor sources.

The most important indoor contaminants are [4]:

- **Asbestos**: generic term that applies to a group of impure hydrated silicate minerals that were used in some building materials and some types of paint. Can cause skin irritation and are related to the lung cancer.
- **Carbon monoxide (CO):** caused by incomplete combustion in heaters or gas stoves.
- **Formaldehyde**: is a colourless gas with a pungent odour. The main sources are materials such as plywood, resins and adhesives. The rate of emission of formaldehyde varies according to the conditions of temperature and humidity.
- **Nitrogen dioxide (NO_2):** as the CO, the most common sources are gas appliances, kerosene heaters, and wood stoves, as well as the smoke of cigarettes.
- **Volatile organic compounds:** Any chemical compound that contains at least one carbon and one hydrogen atom in its molecular structure is referred to as an organic compound. One of the most common types are called aliphatic hydrocarbons and can be emitted by adhesives, furnishings and clothing, building materials or combustion appliances.
- **Carbon dioxide (CO_2):** advanced before, high concentrations of CO_2 around the order of 30,000 ppm can affect humans causing breathing and pulse rate increase. Above 50,000 ppm effects like headaches and sight impairment appears and at a level of 100,000 ppm a person can become unconscious.

As a result of human breathing, carbon dioxide is usually the contaminant with a higher concentration in an indoor environment. Elevated indoor CO_2 concentration reflects inadequate air ventilation, which allows other air pollutants to increment and create health, productivity and comfort problems. However many contaminants are independent of human presence (such as the ones due to emissions of the building materials or paint) and are present even when the CO2 level is low. Again, monitoring indoor CO_2 concentration can give partial information about the air quality.

In addition, monitoring the CO_2 level in an indoor environment gives lots of information about the human activity. The amount of exhaled CO_2 by a person depends, among other factors, on how demanding is the physical activity in that moment [6]. This quantity is higher as the physical activity is more demanding. Therefore, the presence of a person in an indoor environment increases the CO_2 concentration in a different rate depending on the activity doing.

Contrary, when this environment is empty, the CO_2 concentration decreases approaching to the outdoor level, supposing that the environment is not completely airtight. This different variation rates can be used to determine the activity of people in an indoor environment.

Monitoring the activities of a person has numerous applications. Following the objectives of the Sphere Project [6], monitoring the activities of a person gives information about his habits and lifestyle. It can be detected when the person leaves the house, when his physical activity is high or low or even when the user is sleeping. With this information, important changes in the routine of a person can be detected and this can help to an early detection of numerous diseases. For example if a user is detected to remain in his bedroom for more time than usual it can be associated to fatigue, discomfort or even depression. If the CO_2 level in the living room increase higher than usual it can be guessed that someone is visiting. Going back to the air quality, the system can detect when the CO_2 concentration is over the recommended limit of 1000ppm and the ventilation can be increased.

2. AIR QUALITY

This first part of the project focus on the analysis of the CO_2 concentration to study the air quality in both indoors and outdoors environments. This analysis utilizes the relation between CO_2 and another contaminants.

2.1. Air quality outdoors

This part of the project consists on measuring the air quality in the street. In particular, the objective is to measure the CO_2 concentration in the streets to check where the areas with a better and worse air quality are situated. The presence of high levels of CO_2 usually means a presence of another contaminants especially in a transited street, where the main amount of CO_2 is produced by the combustion engine of the transports, the heating systems of the residences or the waste gas of the industries [7].

This experiment consists on measure the CO_2 concentration throughout some transited streets of Bristol. For this purpose, the CO_2 sensor (view appendix 3) will be carried during a walk taking measures of the CO_2 in the street. At the same time, a smartphone executing the free application GPSLogger, will provide the exact position coordinates every second. With this data it is possible to create a map where the CO_2 concentration levels are represented.

The information that the CO_2 sensor provides is the CO_2 concentration on a specific point in time. This measures are taken every 10 seconds. The application GPSLogger provides a CSV file (a type of file that contains data stored in columns) where it relates the position coordinates with the time when the smartphone was in that specific position (apart from other information that will not be used). This information about the coordinates is given every second. Trough the time information it is possible to relate a position with its CO2 concentration, including this information in the CSV file as another column. Finally, using the Google service My Maps, a map representing the CO2 levels in the monitored streets can be created.

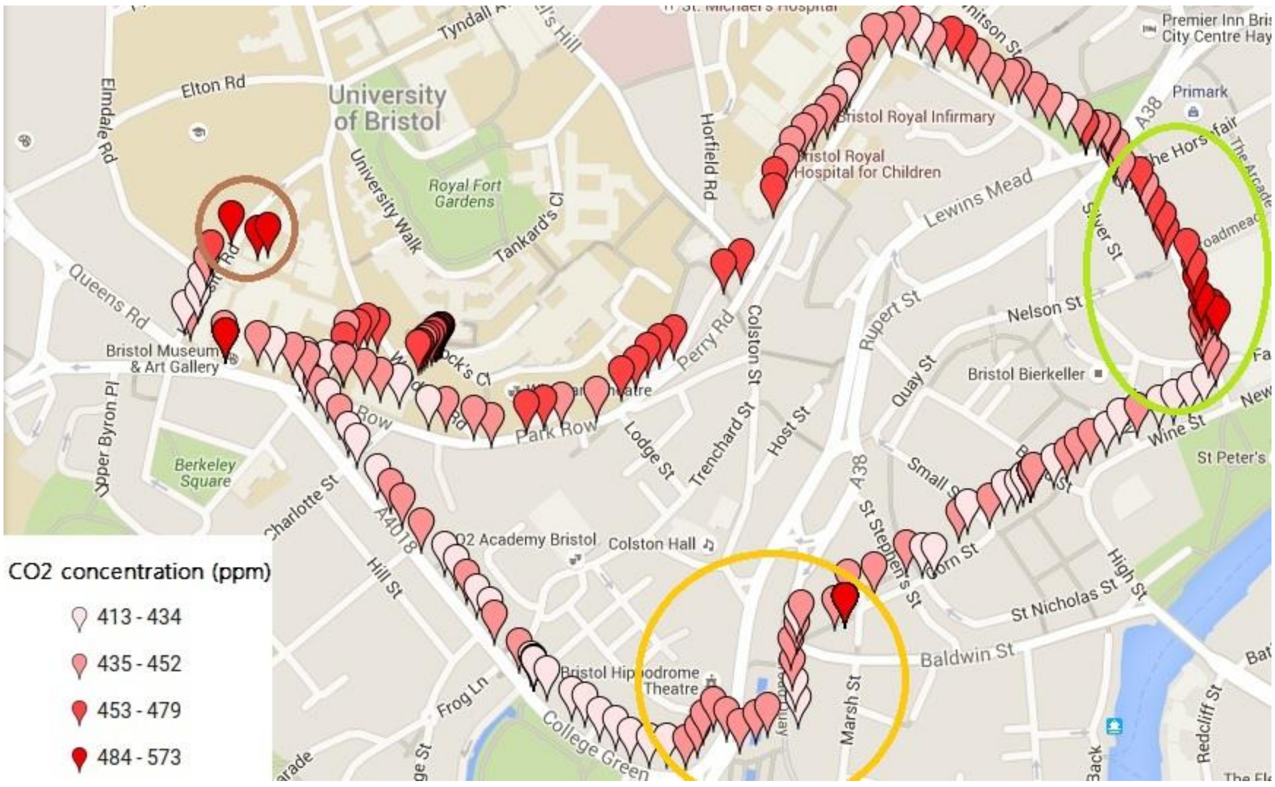

Figure 1.

The colour scale represents various levels on the CO_2 concentration. The orange circle represents a very busy point where there are lots of buses. This corresponds with slightly higher levels of CO_2. The green circle covers a narrow street also very busy where there are lots of stores. The maximum CO_2 levels are found here.

Perry Rd. is a street with a lot of traffic and it is reflected in the CO_2 levels, which are higher than in other less transited streets, as Corn St. The street labelled as A4018 also has a lot of traffic but it is very wide so there is a lot of air flowing and as a result the Co2 concentration is lower. Finally, the brown circle mark the points at the end of the route, which are inside a building, so a higher CO_2 level is expected.

From these results, it can be conclude that some bad air quality points can be created in a city due to the traffic and to the limited airflow in some streets. Performing more similar experiments, these points could be detected with more precision and also the evolution of the air quality during all day can be analysed. With this data, the traffic can be controlled in order to maintain an air pollution below a certain threshold.

Another possible practical application could be a system with some CO_2 sensors spread over the city that sends information about the air quality to the smartphones of the user. As the main part of the smartphones include a GPS system, an alarm could be set to warn the user when they are entering to a bad air quality area.

2.2. Air quality indoors

This part of the project consists on studying the CO_2 levels in closed environments. The main objective is to analyse the air quality along some working days to see if it is suitable for a working environment. Corporations such as ASHRAE (American Society of Heating, Refrigerating, and Air-Conditioning Engineers) and OSHA (Occupational Safety and Health Administration) define the standard of 1000 ppm as the maximum allowed concentration inside a building.

The chosen place for this task is the Sphere Office. In this environment the ventilation is constant during all day. This experiment took place during December so the windows were closed during all day and the air was only renovated by the ventilation system. This situation gives an option to check the air quality in the worst case, supposing that the ventilation system is always working.

In figure 2 is represented the evolution of the CO_2 levels during 10 days, from 30th November to 10th December.

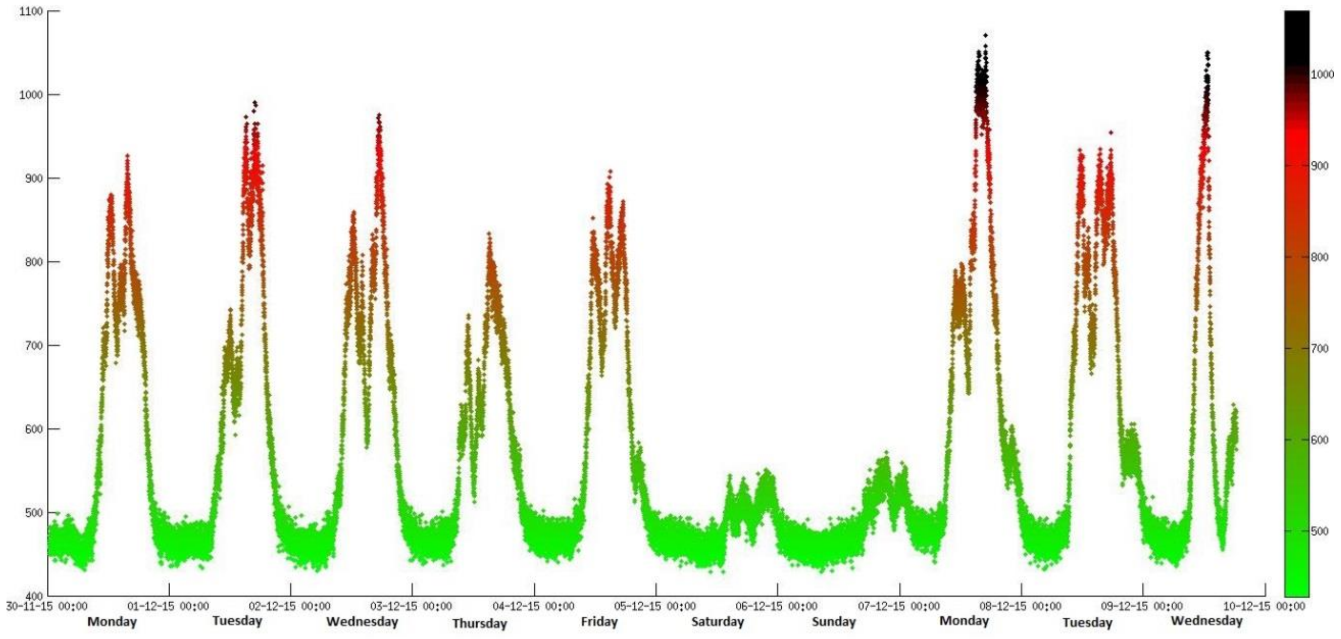

Figure 2

The results from this graph show that a level of CO_2 higher than 1000ppm was only reached in two days (7th and 9th December) for a very short time.

The CO2 maximum CO2 levels reached are high but they are not usually higher than 1000ppm so the ventilation in Sphere Office is enough for the amount of people working there. However if the amount of workers increases it is necessary to increase the performance of the ventilation system if it is not possible to open the window (because of the weather).

Using the information analysed until now, a practical application could be to implement an alarm when the CO_2 level reaches a certain threshold in order to renew the air inside the monitored environment. This can be more complex than a simple alarm. For example, if a CO_2 sensor is implemented in a home automation module, it could send a signal in order to increase the performance or open some windows (depending on the temperature of the environment, the weather…). This could work for rooms independently to be more efficient.

Other use of this information could be a presence detector. In the graph it is easy to identify which days there is no one in the office (Saturday and Sunday) because the CO2 concentration remains very low. In this case it would not be able to detect a person instantly but it could be useful for combining information with other presence sensors (a camera) to obtain more certainty.

Observing the graph in detail, it is easy to identify some patterns that are repeated each day. In figure 3 the last three days are represented.

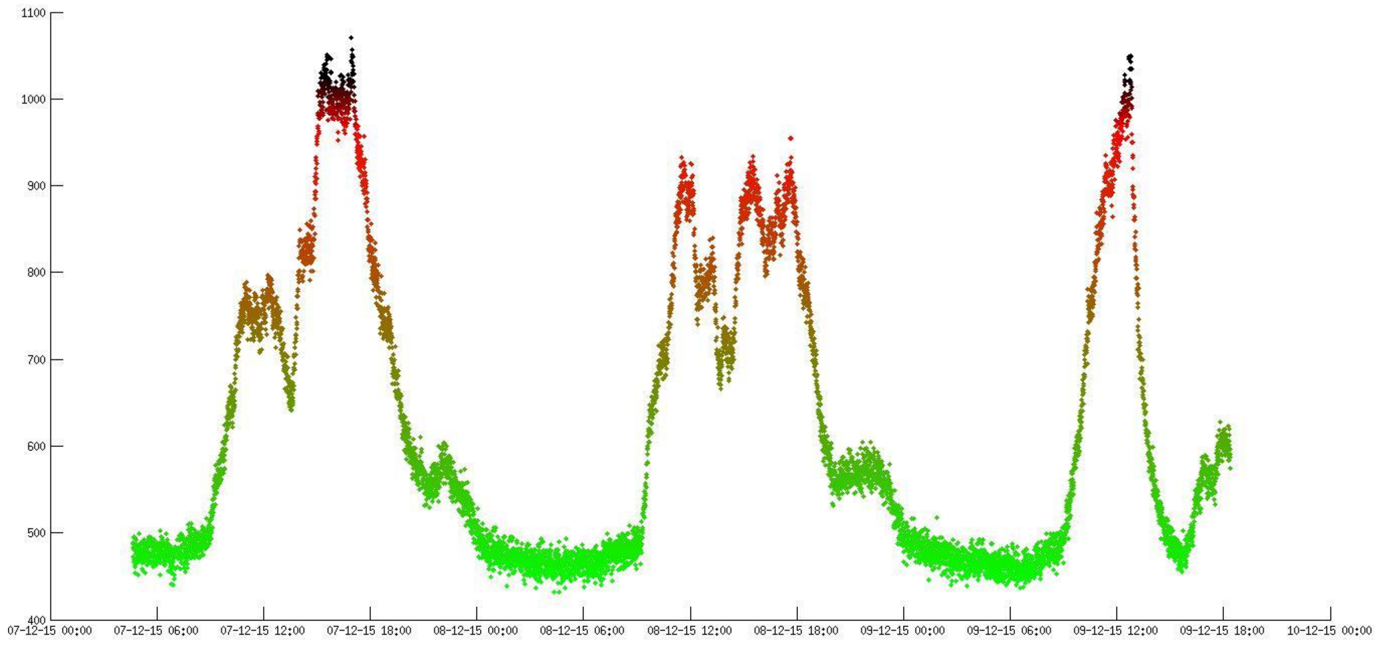

Figure 3

During the night, the CO_2 concentration is below 500ppm. This value is close to the atmospheric CO_2 normal concentration (400ppm) [2] so it is sure that no one is in the office. From 9 am the concentration level starts to increase with a high rate, coinciding with the time when the researchers start arriving to the office. This increasing rate is maintained until 11:30 am approximately, when some people goes out, takes a break, etc. The CO_2 rate around this time is very variable between days, which is logic because there is no rule on when the people have to leave the office.

Another clear behaviour can be observed around 1 pm. Usually, a decreasing rate can be observed some time before. People usually have lunch at this time so they start leaving the office some time before, which produces the CO_2 level to decrease. Effectively, a local minimum can be seen around 1:30 pm, when less people is inside the office (this is a mean, it varies between days). After that, the researchers return to the office and it is possible to observe a CO_2 concentration increment similar to the one in the morning. This increasing rate usually last around 2 or 3 hours.

Finally, after 18:00 a very high decreasing rate is observed every day. By this time there is no one in the office and the CO2 concentration starts decreasing, approaching the outside concentration.

An interesting case occurs the last day. This day there was a lunch for all the Sphere researchers so, as usual, the people started to arrive at 9 am. The graph shows an increasing rate similar to other days. However the people did not start going out or taking a break by 12:00, they waited inside until 13:00 to go all together to have lunch. In the graph this behaviour can be seen very clear. The increasing rate is maintained until 13:00, moment in which it suddenly becomes a high decreasing rate. A consequence of everyone been inside, and therefore contributing on the increasing of the CO_2 level, is that a peak higher than 1000ppm is reached. The high decreasing rate is produced because no one was in the office after the researchers left.

Another detail that can be extracted is that this time the decreasing rate is higher than other days when everyone leaves the office. The reason is that during the working hours the ventilation system is renewing the air but this does not happen after the working hours. As any other day there is some people working at that time, the ventilation system is switched on and the concentration of CO_2 reduces faster than any other day.

To sum up, monitoring the CO_2 levels is useful to get information about air quality but it is possible to obtain much more information related to the activities of the people inside the monitored space. In this case the only information extracted is related to the presence of people inside the room but it is possible to extract information related to what kind of activity is the people doing by analysing the data in a more detailed way, looking at the values of the increasing and decreasing rates and the observable patterns in the different graphs.

3. ACTIVITY DETECTOR

Following the analysis of the previous experiment, the next part of the project will consist on identify different activities that a person can do inside a room. Different activities will produce different increasing or decreasing rates in the CO_2 level. This will be the main principle used to distinguish them.

A room of 14 m^2 will be used for this purpose. Inside this room, the CO_2 concentration has been monitored during various days and the activities and duration have been noted in order to have a ground truth.

This part will be divided in two parts. The first part consists on a previous study aiming to determine which activities can be predicted. The second part explains the implementation of a program written in Matlab which determines these activities automatically.

Respect to the air quality, in all these measures, the CO_2 concentration limit of 1000ppm is overcome most of the time so it is necessary to improve the ventilation in the room.

3.1. Previous study

In the next figures there are some examples of some data recollected where it is possible to distinguish different increasing and decreasing rates. The noted information about activities of the person inside the room is also indicated:

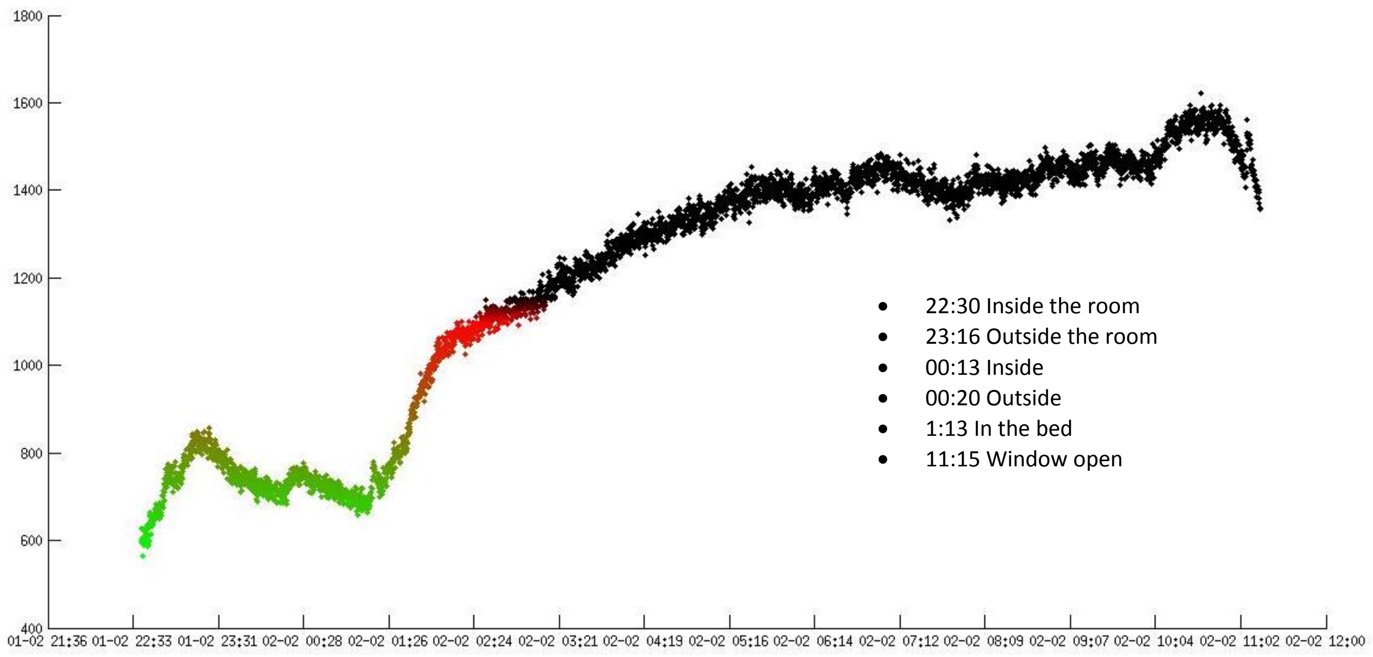

Figure 4

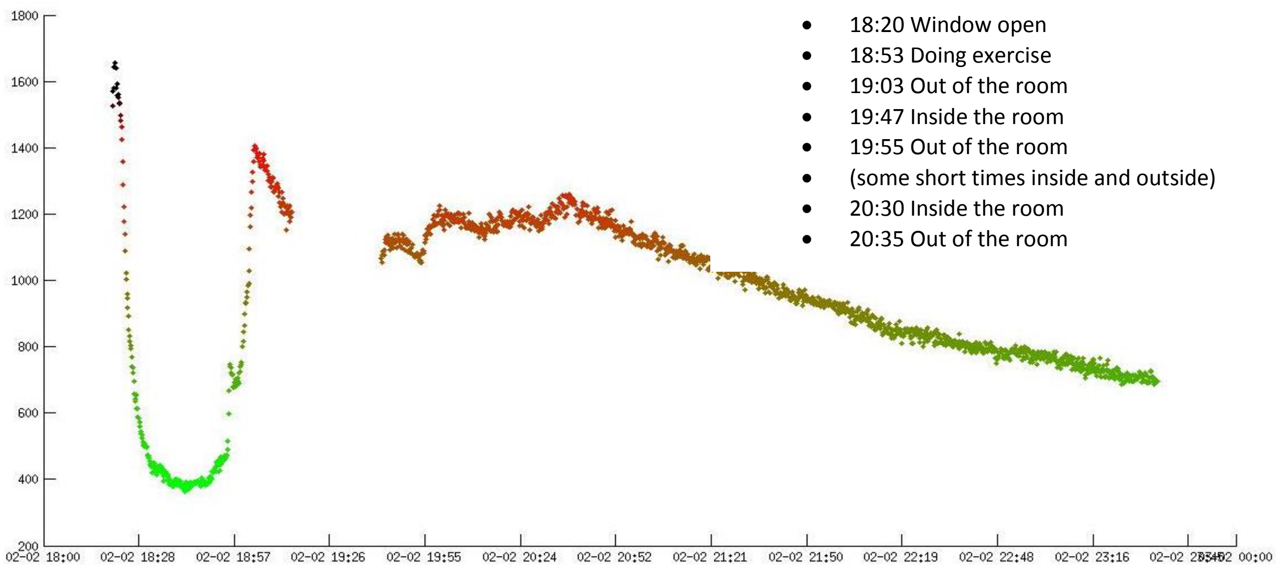

Figure 5

A very good correspondence between the slopes of the graph and the events in the room can be observed. At the time when there is no one inside a relatively low decreasing rate can be observed (this activity will be called “out” from now). This decreasing rate is higher when the window is open (activity “window open”). This is something logic because more fresh air enters to the room and therefore the CO_2 concentrations approaches to the concentration in the open air.

When the user is inside the room (activity “inside”) doing an activity like studying or resting, an increasing rate is observed. This increasing rate is much higher when the user is doing exercise (activity “exercise”). Again, this is something logic because of the amount of CO_2 produced by the body when doing a demanding physical activity [5]. The last activity and the most complicated one that it is possible to detect is when the user is sleeping (activity “sleeping”). This activity usually follows a certain pattern: first, the increasing rate of the CO_2 concentration is similar to the activity “inside”. After some time, the increasing rate of CO_2 concentration becomes lower. This can clearly be seen in figure 4. The reason is that the amount of exhaled CO_2 by a person deep sleeping is lower than when he is awake [8] but it takes some time to reach the deep sleeping phase.

The five activities that it is possible to differentiate have been defined. They will be identified basically looking at the increasing or decreasing rate. In the figure 6 and 7 the graphs seen before have been divided manually in different segments, each one corresponding to a different activity. After this division, the different segments of the graph have been approximated to a linear function, using the method of linear regression (see appendix 1). The slope of this linear function will correspond to the increasing rate of the CO_2 concentration and it will be the key to identify the activity in that period of time.

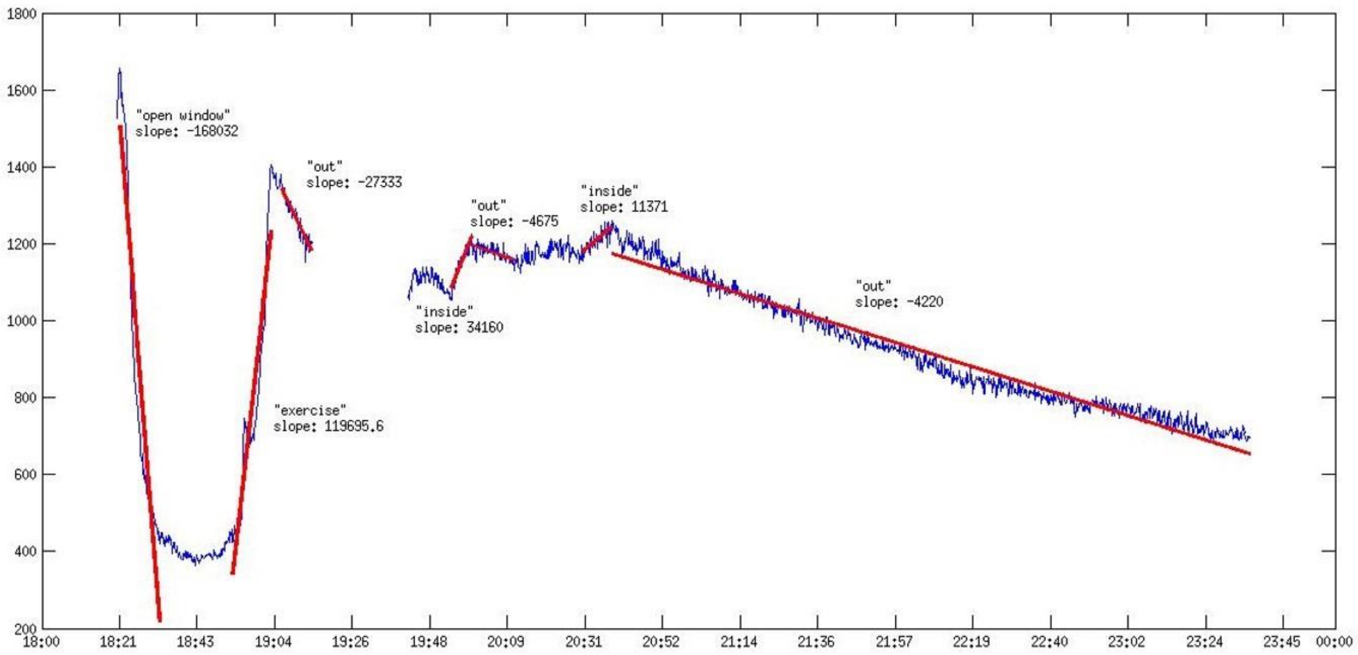

Figure 6

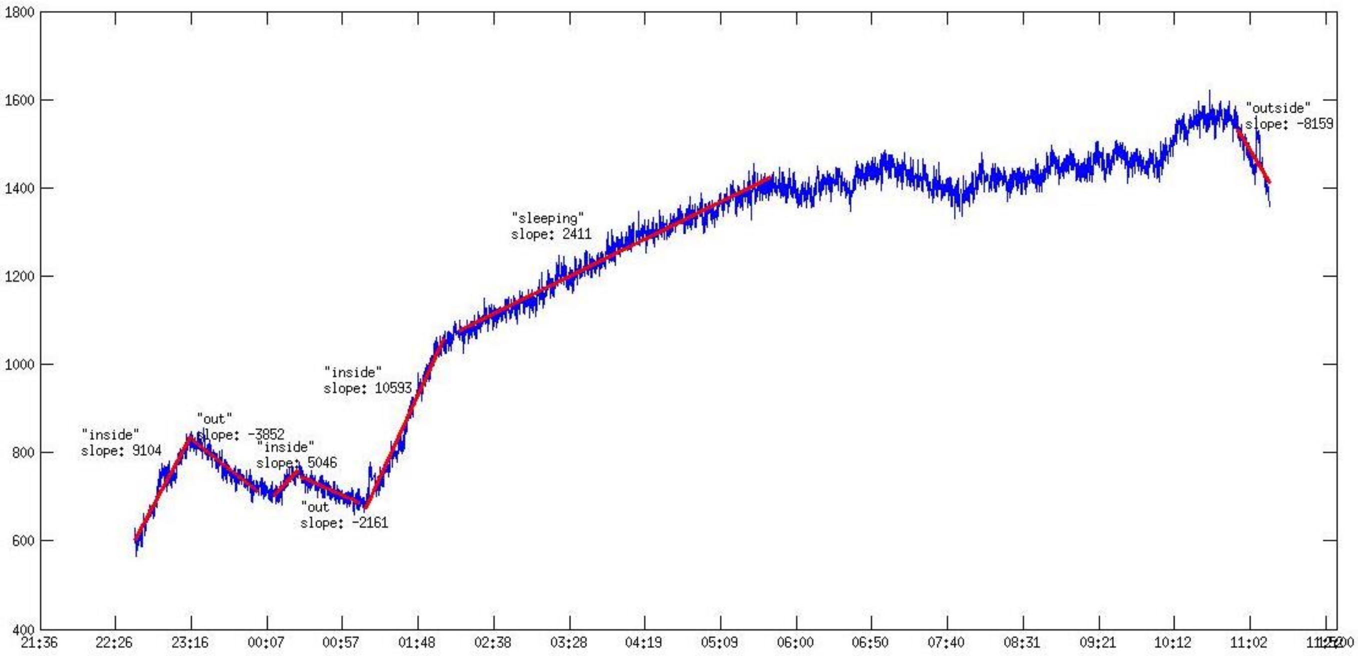

Figure 7

It has been show that it is possible to identify different activities using three steps:

1. Divide the sets of data in segments where the variation rate of Co2 is similar.
2. Approximate the graph to a linear function.
3. Using the slope of this linear function to identify the type of activity.

3.2. Implementation of the activity detector in Matlab

The main objective of this part is to identify different activities and to determine its duration given only a set of data obtained directly from the sensor, as in section 3.1. This has to be done automatically, without any human interaction apart from providing the original data to the algorithm.

The program created for this purpose will basically follow the three steps in section 3.1 but it will add a filtering process before. Furthermore it will combine different iterations of functions that will identify activities, divide the dataset and approximate the graph to linear functions.

All the information needed (linear models, original data, identified activities…) will be stored in variables that will be used and modified in multiple functions. These variables are:

- **date**: Contains the original information about the monitoring time obtained directly from the sensor. It is a vector where every component represents the exact point in the time when a measure was taken (every 10 seconds).

- **co2**: Contains the original information about the CO_2 concentration obtained directly from the sensor. It is a vector where every component represents the value of the CO_2 concentration in every measure. The time when this measure was taken is the value of the same index component in the vector **date**.

- **datem:** It is a cell containing vectors. Each vector corresponds to one segment (that represents an activity). The vectors of **datem** contain the original information of **date** filtered and organized in different segments.

- **co2m**: It is a cell containing vectors. Each vector corresponds to one segment (that represents an activity). The vectors of **co2m** contain the original information of **co2** filtered and organized in different segments.

- **b**: It is a cell that contains the information about the linear model created for every segment. It has the same components as **datem** and **co2m.** In each component is stored the slope of the segment with the same index in **datem** and **co2m**.

- **R**: It is a vector where every component represents the coefficient of determination of the linear model for every segment. Each component corresponds with the same index components in all the other vectors.

- **a**: It is a vector that identifies the activity associated to every segment. Contains numbers that represent these activities. Each component corresponds with the same index components in all the other vectors.

- **g**: It is a vector that contains the activity for every segment when they have been divided in smaller segments of 25 points. See section 4, Ground truth and score.

- **G**: it is a vector whose length is the same than **g** and contains the ground truth values of the activities for every segment of longitude 25 points.

- **I**: it is a vector containing the initial time of every segment of 25 points.

3.2.1. Filtering

The objective of this first process is to smooth the data in order to be analysed easily and to remove the noise in the signal. For this purpose, the high frequency components of the signal need to be removed as they are the cause of the "abrupt" changes between consecutive measures. This can be achieved using a low pass filter.

To implement a low pass filter on Matlab, a low pass digital butterworth filter will be used. Using digital filters the frequency response is normalized, doing $\Omega = \omega T_S$ so the normalized frequency sampling will be $\Omega_S = 2\pi$. The frequency is represented in the x axis up to 2π.

The matlab function used to design this kind of filter is:

```
[b,a] = butter(n,Wn)
```

It returns the transfer function coefficients (a and b) of an n order filter whose normalized cut-off frequency is Wn.

The data obtained from the sensor is stored in two vectors. The vector “date” which stores the different points in time when the measure was taken and the vector “co2” that stores the value of each measure. Figure 8 shows the graph of the dataset that will be used to explain the program from this point.

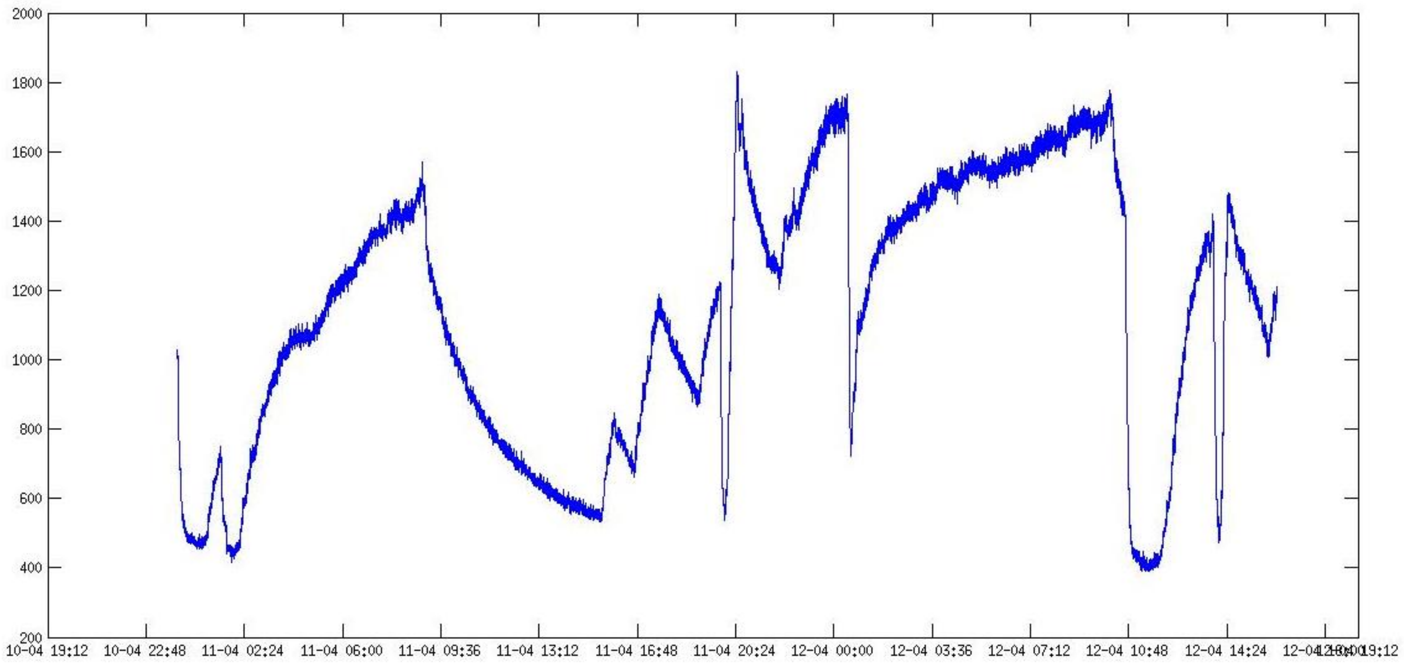

Figure 8

This graph is obtained by:

```
plot(date,co2)
datetick('x','dd-mm HH:MM', 'keepticks')
```

It is very clear that between one measure and the next one there is an abrupt change, which means high frequencies. A low pass filter will help to remove these high components and the result will be a smoother signal.

Trying with a 6-th order filter and a normalized cut-off frequency of 0.01 it is possible to get a result much better than the original graph. This is shown in figure 9.

The coeficients are obtained using:

```
[b,a] = butter(6,0.01);
```

And the filter applied with:

```
co2F = filter(b,a,co2);
```

As before, the plot is obtained with

```
plot(date, co2F)
```

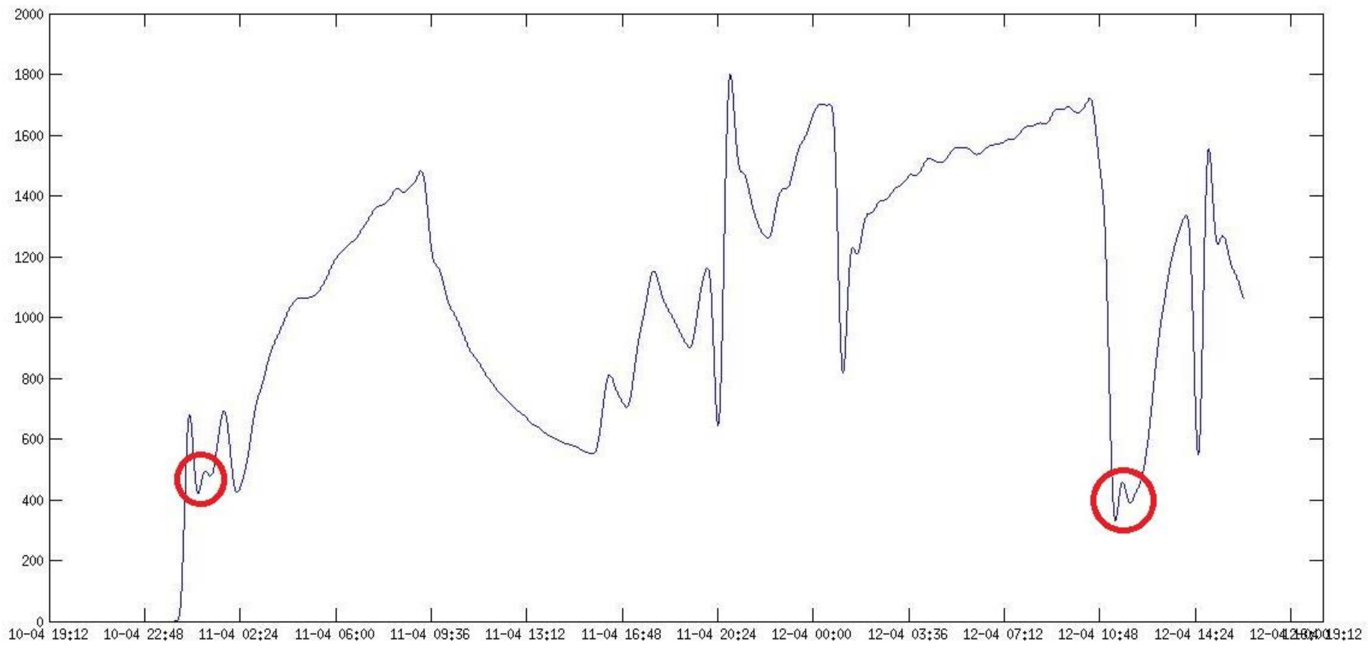

Figure 9

The graph is now very smooth and is much easier to identify minor changes in the variation rate of CO_2. However some new problems have appear. First of all, some unexpected distortion appears in the parts when a big change in the slope is produced. This is indicated with a red circle. The reason is that the filter is removing some high frequencies that have information about the signal. The solution resides on increasing the cut-off frequency of the filter. Using a cut-off frequency of 0.04 and changing the order to 5, the resulting graph have some abrupt changes but it still contains the information that was lost in the previous case.

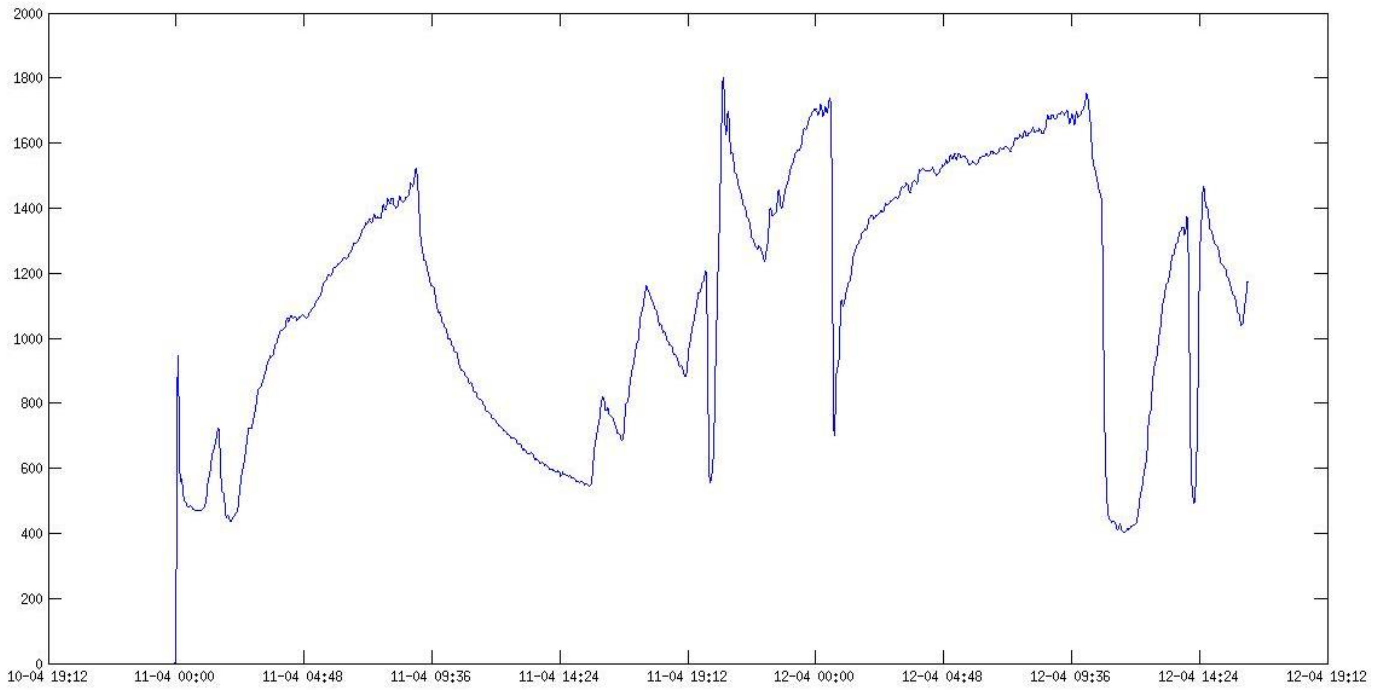

Figure 10

As expected, this problem has disappeared, but there is still another problem. At the beginning of the signal, the filter creates a slope from 0 to the first value of the data that distorts this first part of the data. One practical solution to solve it resides in modifying the original dataset adding some data (2000 new points) at the beginning that “protects” the useful data. Some points with a value equal to the first value of the co2 will be added so that the distortion only affects this “useless” data. Figure 11 shows the filtered result of this data.

This time a function is created to add this new data

```
function [co2, date]=adddata(co2,date)
o = date(1);
s = 1.157412771135569e-04;
nco2=co2(1);

for j=1:2000
    pdate = o - j*s;
    co2 = [nco2; co2];
    date = [pdate; date];
end
end
```

The variable *s* contains the duration of 10 sec in the numeric matlab format. Every iteration of the for loop, a new point in time 10 sec previous to the earliest one is created and a value of co2 equal to the first value of the original dataset is associated.

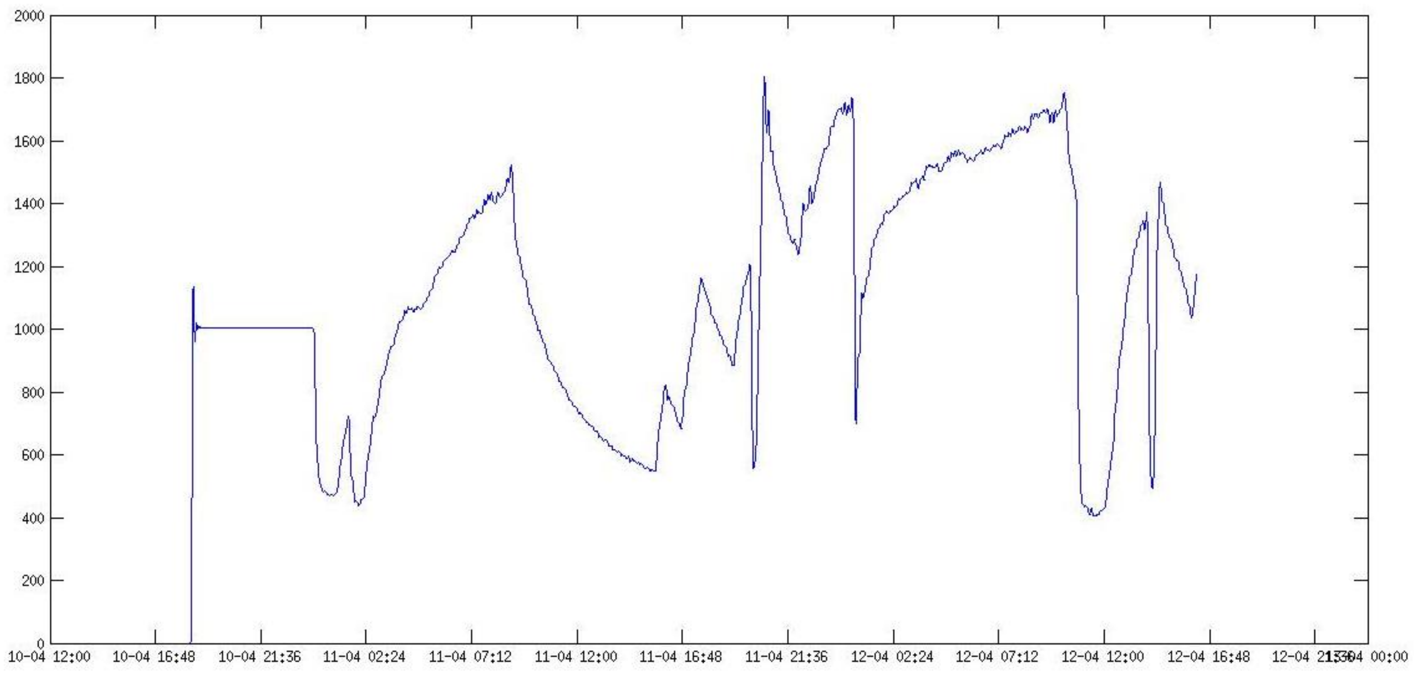

Figure 11

At this point when all the data is correctly filtered, the next step is to remove the added data that is not part of the original dataset. But there is still another minor problem. When the data is filtered, it is slightly delayed in time. Whit the current filter, this delayed time is practically 5 minutes, which means 30 points in the dataset. This problem can be solved at the same time than the “protecting” data is removed. To remove the “protecting” data, the first 2000 points of the co2 and date vectors will be removed. In addition, another 30 points, corresponding to the delay time will be removed, but this time they will be removed only from the co2 vector, so that the introduced delay after applying the filter is compensated. The bad part here is that the final 5 minutes of data will be lost, but this is necessary for identifying the activities at the correct time.

All this processes are put together in function that will be used later:

```
function [dateF, co2F]=butfilter(N,fc,co2,date)

[co2, date]=adddata(co2,date);

[b,a] = butter(N,fc);
co2F = filter(b,a,co2);

dateF=date(2000:(end-30));
co2F=co2F(2030:end);
```

It can be seen how the first 2000 points are removed from both co2 and date vectors but the next 30 points are removed from the beginning of co2 and from the end of date, in order to compensate for the delay introduced in the filtering process.

The data after all these processes is shown in figure 12

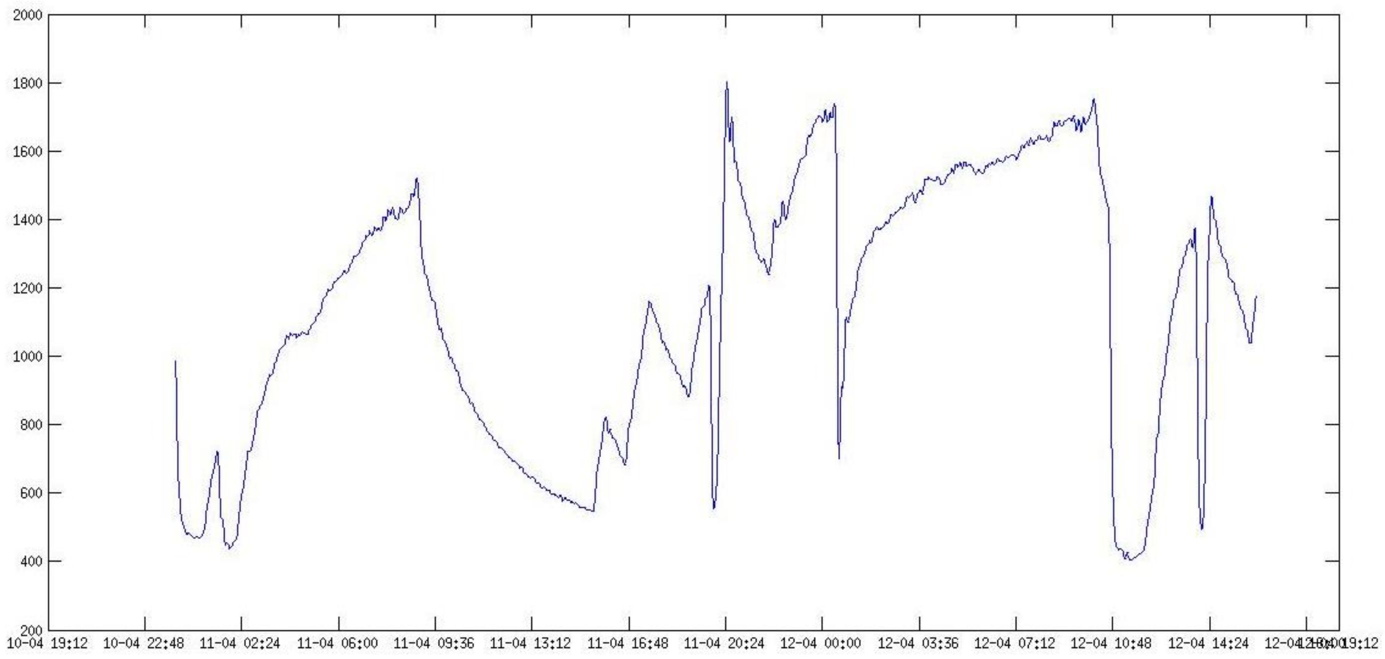

Figure 12

3.2.2. Divide data

After having completed the filtering process, the next objective is to automatically divide the dataset in pieces corresponding to only one activity. For achieving this, all the data will be divided first in pieces of a fixed length. This division process does not use any information about the different possible activities to determine where to start dividing the data, it is only one way to create segments to make the first analysis.

These segments are 100 points long and, after finishing the division process, the slope of all these segments will be calculated to be used by the next function. As mentioned before in section 3.1, the slope of every segment will be calculated using linear regression. Information about linear regression is available in appendix 1.

The division process and the calculation of the slopes of every segment has been implemented in a matlab function that uses the filtered data as the input.

```
function [datem, co2m, b, R] = divide(date,co2)
N=floor(length(date)/100);
for i=1:N
    datem{i} = date(100*(i-1)+1:100*i);
    co2m{i} = co2(100*(i-1)+1:100*i);
    X = [ones(length(datem{i}),1) datem{i}];
    b{i} = X\co2m{i};
    y{i}=X*b{i};
          R(i) = 1 - sum((co2m{i} - y{i}).^2)/sum((co2m{i} -
          mean(co2m{i}))).^2);
end

    N=N+1;
    if(length(date(100*(N-1)+1:end))>25)
        datem{N} = date(100*(N-1)+1:end);
        co2m{N} = co2(100*(N-1)+1:end);
        X = [ones(length(datem{N}),1) datem{N}];
        b{N} = X\co2m{N};
        y{N} = X*b{N};
              R(N) = 1 - sum((co2m{N} - y{N}).^2)/sum((co2m{N} -
              mean(co2m{N}))).^2);
    end
end
```

First of all, the number of segments of longitude 100 is calculated, the function floor rounds towards the lower value so that the last segment, which probably will have less than 100 points, is not assumed to be 100 points and it will not be a problem in the algorithm. Inside the for loop, two cells (datem and co2m) are filled with N vectors of 100 components. Each pair of vectors from datem and co2m will be used to approximate the correspondent 100 points segment by a linear function. Using the operator \ , the approximated linear function is obtained. This result is stored as a two components vector (the second will be the slope of the linear function) inside another cell (b). Using the slope, the values of the dependent variable of the linear approximation are calculated and stored in a vector that will be stored in another cell (y). Finally, the coefficient of determination is calculated and this time it is stored in a vector, as this coefficient is only one scalar value.

After processing the maximum number of segments of 100 points, the data that is left is processed the same way if the number of measures is greater than 25. This threshold is choose because it will be used in the following sections as the minimum length of a segment.

Storing the data using cells allows to have the data well classified so it will be very quick to access a specific segment or value in any other functions. The values of the segments of date, co2, the linear model, the approximation and the coefficient of determination can be accessed using the same index in the different cells (vector for the case of the coefficient of determination).

Figure 13 shows the result of dividing the data in this way and representing the approximation for every segment (in red). The approximations are compared with the original data in order to show the correlation.

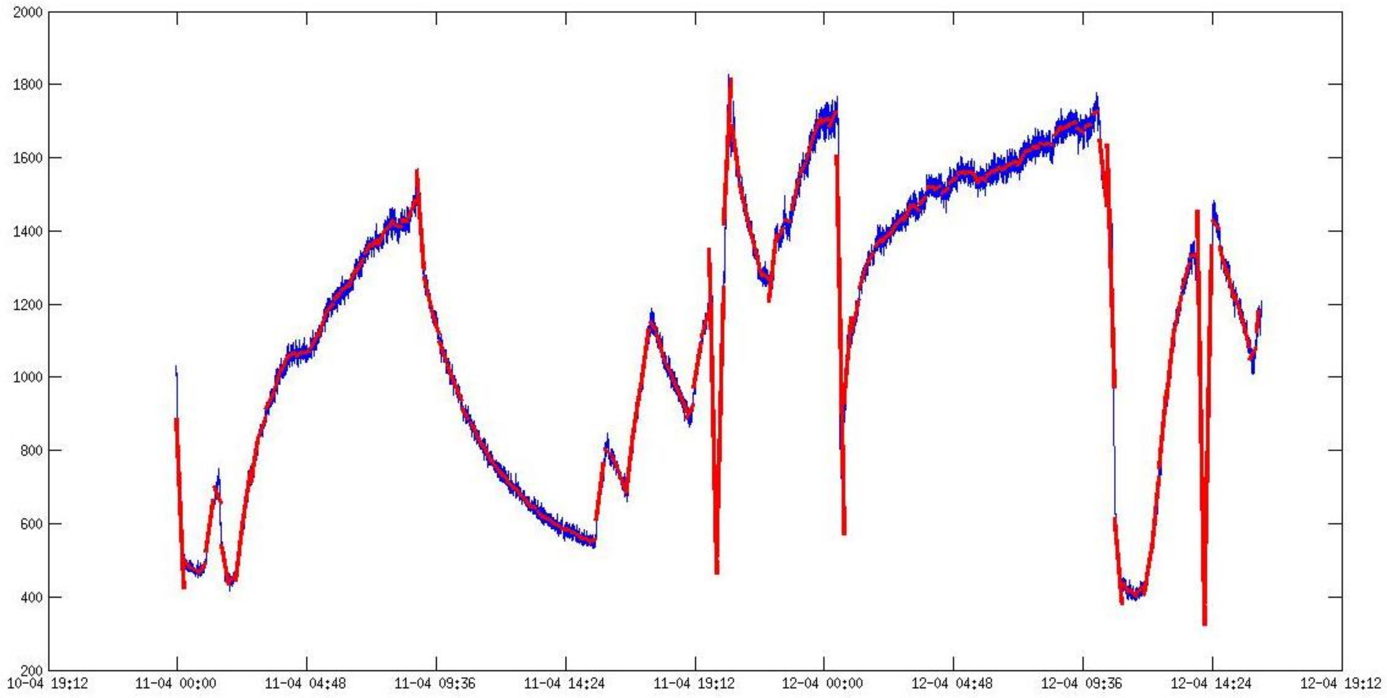

Figure 13

Using segments of 100 points will ensure that the main part of the approximations have a coefficient of determination close to 1.

3.2.3. Unifying slopes

At this point, the original dataset has been divided in numerous segments and each of them have a linear approximation which indicates the approximated slope. With this information it is possible to start classifying the segments depending on the value of the slope. The main objective of this part is to unify each segment with its neighbour segment if they have similar slopes that may correspond with the same activity.

For this purpose, a matlab function has been created whose main operation is to look at the slope of each segment, stored in the cell **b,** and unify the vectors (**date** and **co2**) of this segment inside the cells of **co2n** and **daten**.

```
function [daten, co2n, R, b]=unify(datem,co2m,b,p)
j=1;
daten{j} = datem{1};
co2n{j} = co2m{1};
for i=2:length(datem)
    X = [ones(length(daten{j}),1) daten{j}];
    f = X\co2n{j};
    if abs(f(2)-b{i}(2))<p
        daten{j} = [daten{j}; datem{i}];
        co2n{j} = [co2n{j}; co2m{i}];
    else
      j=j+1;
      daten{j} = datem{i};
      co2n{j} = co2m{i};

    end
end

for i=1:(length(daten))
    X = [ones(length(daten{i}),1) daten{i}];
    b{i} = X\co2n{i};
    y{i}=X*b{i};
    R(i) = 1 - sum((co2n{i} - y{i}).^2)/sum((co2n{i} -
           mean(co2n{i})).^2);
end
```

This function creates two new cells (**daten** and **co2n**) with the unified values of **datem** and **co2m**. In every iteration of the first for loop, it compares the current vectors of the input data (**datem** and **co2m**) with the last stored segment in the output data; **daten** and **co2n** are initialized at the beginning with the first value of **datem** and **co2m**. First of all, it calculates the slope of this last segment using linear regression. If the difference in the slopes of the segments in **datem** and **daten** is lower than a certain threshold (**p**), they will be unified and stored in **daten**. If the difference is bigger, the segment information is copied directly to **daten**.

In order to maintain the correspondence (using the same index) between the vectors of **co2n** and **daten** and their linear model and coefficient of determination, they are calculated again for the cells **co2n** and **date2n** once they have been filled completely.

The threshold has found to be optimal when its value is 3000. There is always a balance between how many segments that represent the same activity are unified and how many segments that represents different activities are incorrectly unified. With a bigger threshold the final number of segments would reduce more but the amount of errors would increase. Contrary, with a lower threshold the number of unified segments would be lower so the ideal objective of unifying all the neighbour segments concerning the same activity would be far from being achieved. The result of this function is shown in figure 14

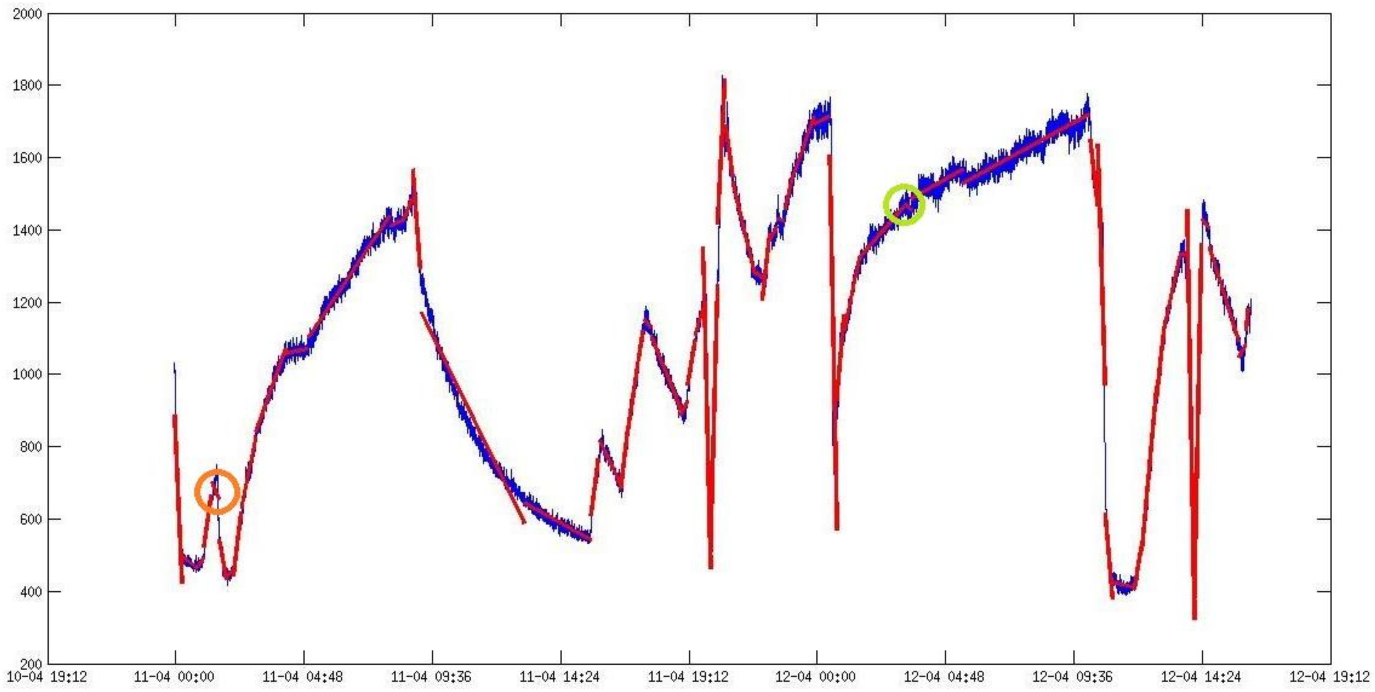

Figure 14

It can be clearly seen how the number of segments have been reduced and that the main part of them are longer now. In this particular case the number of segments has been reduced from 145 to 60, which represents a very good unification. In some cases there is only one segment that identifies the beginning and the end of an activity and that could also identify the activity looking at their slope, but this will be developed later on.

This is the first step to identify automatically when every activity starts and ends. Sometimes it is possible to identify a small segment between two segments with a very similar slope which has a very different slope (circled in green). Looking at the graph it can be deduced that it is an error and that it should be unified with their neighbours. This problem will be solved later.

Another problem that can be observed since the divide function was used is that the segment has been placed between activities with a very different slope, so the resulting slope of the segment does not correspond with the original data. This case is marked in orange in the figure. The next step will be solve this kind of problems.

3.2.4. Dividing segments

As introduced before, one problem that appears when using this algorithm is that the division of the data in different segments does not uses any information regarding to the slope of the segment. This can lead to create a segment that covers a similar part of two different activities. The inflection point between these two activities will be in some point near half of the segment. For this reason when the linear approximation of the segment is calculated the resulting slope is not accurate with the real slope in that part of data. An example of this problem is marked in orange in figure 14.

A good option to solve this problem resides in using the coefficient of determination to check which segments are well adapted to the original data and which of them need to be treated. When a segment is not well adapted it will be split in two segments, which are expected to adapt better to the original data. Performing this operation, the two resulting segments are more likely to be unified with their neighbours in a later execution of the “unify” function.

It is also possible that this problem appears because of the presence of a very short time activity in the original data that cannot be detected alone using 100 points segments. In this case, at least one of the two resulting segments is not expected to be unified in a later execution of “unify”. Although this case is less likely to appear it is possible to find it in some datasets. In any case, no segment whit a longitude lower than 25 points will be used so it is possible to determine the minimum resolution for this program from this number. If every measure is taken every 10 seconds, the resolution will be 250 seconds, i.e. 4 minutes and 10 seconds

The matlab function implemented is shown below:

```
function [datem, co2m, R, b] = coefi(datem,co2m,R,c)
i=1;
j=1;
while j<=length(R)
    if R(j)<c
        datem = {datem{1:i} 0 datem{i+1:end}};
        datem{i+1} = datem{i}(ceil(length(datem{i})/2)+1:end);
        datem{i} = datem{i}(1:ceil(length(datem{i})/2));
        co2m = {co2m{1:i} 0 co2m{i+1:end}};
        co2m{i+1} = co2m{i}(ceil(length(co2m{i})/2)+1:end);
        co2m{i} = co2m{i}(1:ceil(length(co2m{i})/2));
        i=i+1;
    end
    j=j+1;
    i=i+1;
end

for i=1:(length(datem))
    X = [ones(length(datem{i}),1) datem{i}];
    b{i} = X\co2m{i};
    y{i}=X*b{i};
    R(i) = 1 - sum((co2m{i} - y{i}).^2)/sum((co2m{i} -
          mean(co2m{i})).^2);
end
```

The structure concerning cells and vectors used to store the data allows to access the slope of a segment, its coefficient of determination and its original data using the same index.

When the coefficient R^2 is lower than a certain threshold (**c**), a new component in the cells **datem** and **co2m** is created. This new component is filled with half of the data that is being analysed at that moment, creating the second segment of the division. The current vector in **datem** and **co2m** is updated with only the first half of the information so that it will correspond to the first segment of the division. Again, the linear models for this new segments will be calculated. The results of applying this new function can be seen in figure 15.

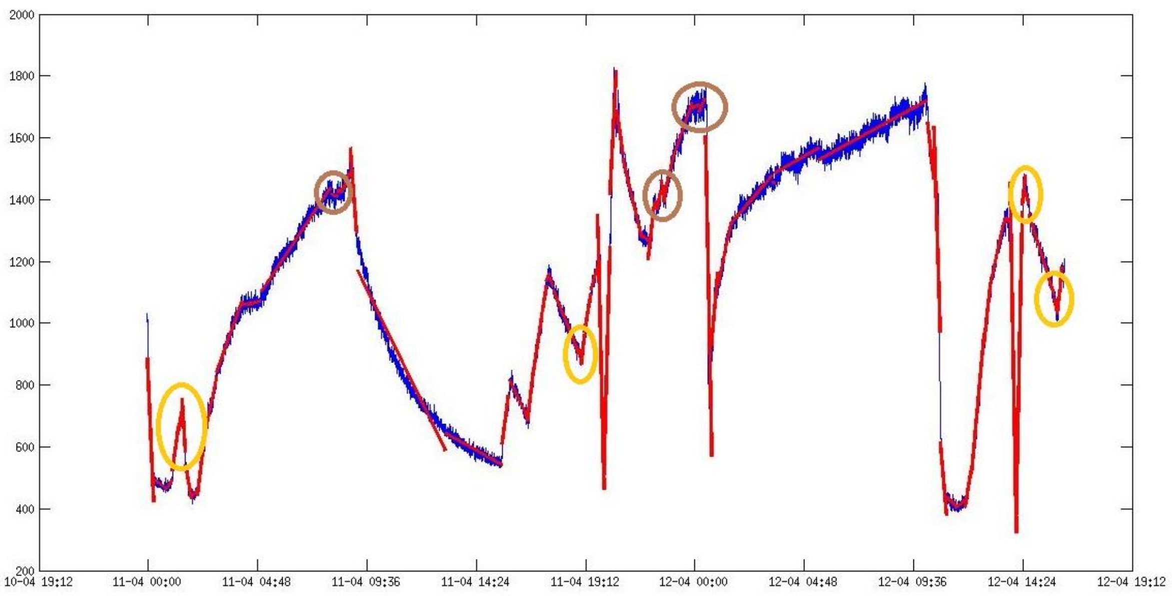

Figure 15

The regions where the correspondence between the linear model and the original data has increased have been marked in orange while the regions where some errors have appear have been marked in brown. This new errors are produced in areas with a very abrupt change in the CO_2 levels that leads to neighbour segments with very different slopes which will not be unified. Although the data has been filtered, this is not enough to avoid these kind of problems. A later function will deal with them.

3.2.5. Second unification of the slopes

The objective of dividing the segments with a low coefficient R^2 is to obtain segments that adapts better to the original data (which has been filtered). Having achieved this objective the next step is trying to unify these new segments with a larger segment representing the same activity. These new segments created in the function “coefi” are very likely to be unified because the main part of them are the result of splitting a segment that was covering the inflection point between two activities.

After unifying these new segments depending on its slope by executing again the function “unify”, the result is a set of segments that approximate the duration of the different activities with a better precision than before. One difference now is that the segments will not be multiples of 100 as some of the segments were 50 points long before being unified to the bigger ones.

Figure 16 shows the performance of using another iteration of the “unify” function with the same threshold as in the first iteration, 3000:

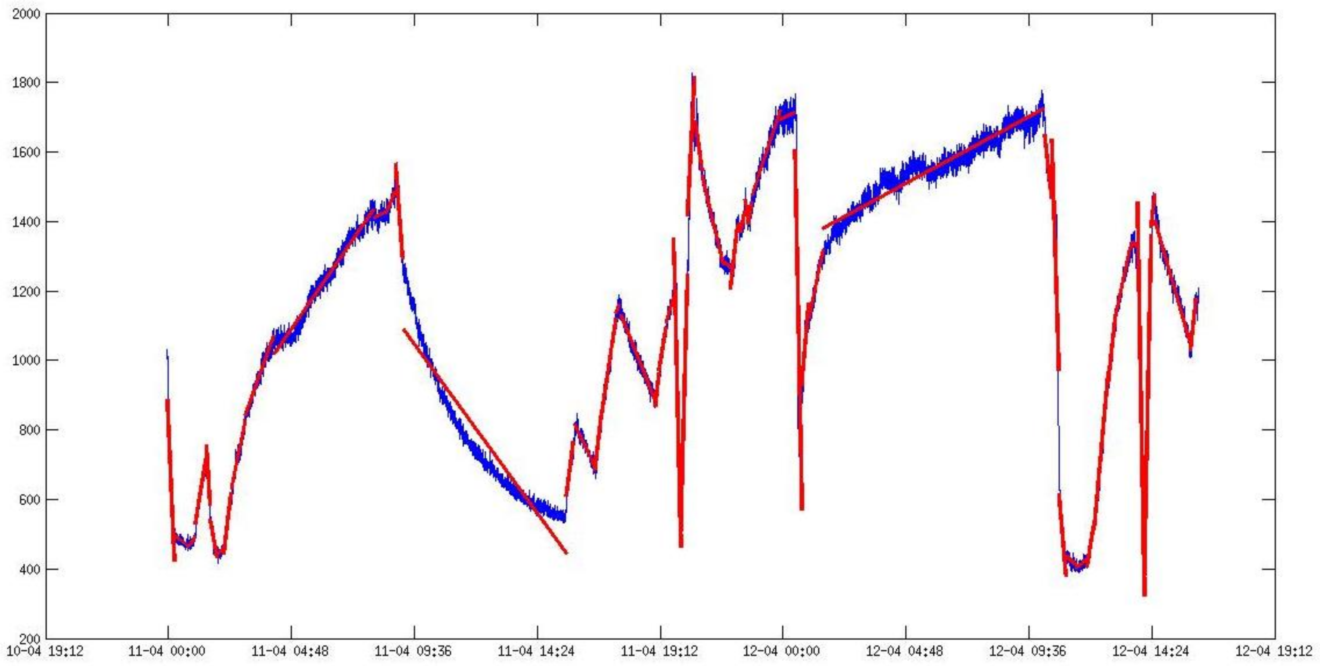

Figure 16

The main part of the new segments created with the function “coefi” have been unified with one of their neighbour segments. Furthermore, this second iteration has allowed a further unification of some long segments with a similar slope. The result is a set of segments that identifies better the beginning and end of the different activities.

Contrary to the longer segments obtained now, there are some little segments corresponding to some misinterpretation of the program and to long transitions between some activities with a very different slope. The next step will be to fix as much as possible these errors without losing the correct information.

3.2.6. Remove Errors

In sections before the problem of having little segments has appeared. Part of this little segments appear between two longer segments with a similar slope value and they have a slope very different. For this reason, they cannot be unified with the long segments and this long segments also cannot be unified because they are not successive. Other case where they appear is in long transition between activities, where the progressive change on the slope can be identified as an activity that has not really occurred. These segments lead to a misinterpretation on the data and they should be removed without losing the correct short segments.

A new function is implemented in order to fix this problem. It creates two new cells (**daten** and **co2n**) containing the updated vectors with the reorganized data that the new segments will use. To do that it will unify the little segments with the most equal neighbour in terms of value of the slope. It will check every term of the input cell datem to identify which segments are shorter than 50 points and therefore represent a problem. These segments will be unified to one of its neighbours only if the absolute value of their slope is lower than 20000. The reason for this threshold is that, as it will be show later, some activities are very short and have a relatively high slope with an absolute value greater than 20000. These kind of segments should not be unified, otherwise it would mean a loss of information.

When a segment is found to be shorter than 50 points, the difference of its slope with the slope of its previous and its posterior neighbour is calculated. Depending on which difference is shorter, the data of this segment will be unified with the data of the segment whose difference is lower and added to the new cell (daten and co2n). If the segment is not going to be unified it will be copied directly to the new cell. Like in the other functions, after having complete the **co2n** and **daten** cells, the new linear models and its coefficient R^2 are calculated using linear regression.

```
function [daten, co2n, R, b] = RemoveErrors(datem,co2m,b)
i=2;
j=1;
daten{1}=datem{1};
co2n{1}=co2m{1};
while i<(length(datem))
    if ((length(datem{i})<51) && (abs(b{i}(2))<20000))
        if((b{i}(2)-b{i+1}(2))>(b{i}(2)-b{i-1}(2)))
            daten{j} = [daten{j}; datem{i}];
            co2n{j} = [co2n{j}; co2m{i}];
        else
            j=j+1;
            daten{j} = [datem{i}; datem{i+1}];
            co2n{j} = [co2m{i}; co2m{i+1}];
            i=i+1;
        end
    else
        j=j+1;
        daten{j}=datem{i};
        co2n{j}=co2m{i};
    end
     i=i+1;
end

daten{j+1}=datem{end};
co2n{j+1}=co2m{end};

clearvars b;

for i=1:(length(daten))
    X = [ones(length(daten{i}),1) daten{i}];
    b{i} = X\co2n{i};
    y{i}=X*b{i};
    R(i) = 1 - sum((co2n{i} - y{i}).^2)/sum((co2n{i} -
    mean(co2n{i})).^2);
end
```

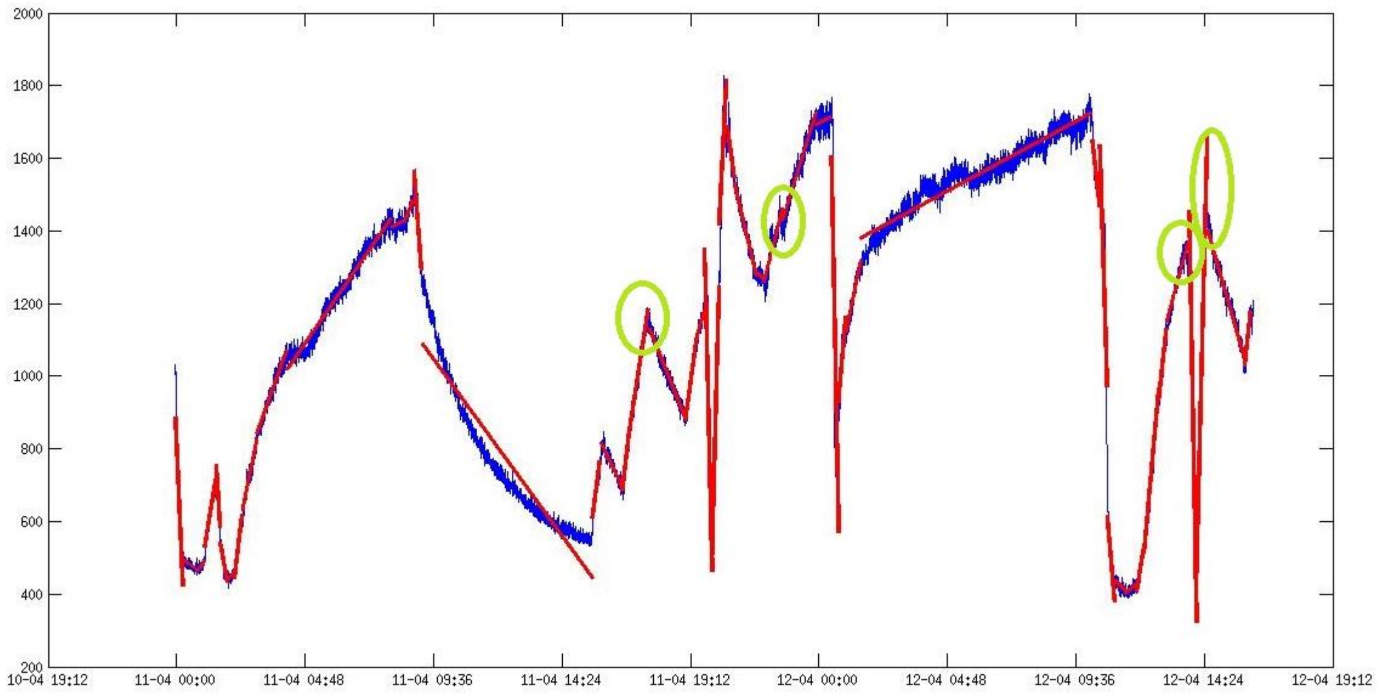

Figure 17

The circles in green indicate areas that had some small segments before and that now have been improved. The number of segments has been reduced from 57 to 48. The limits between different activities are well defined now although the precision can be increased. Furthermore it is possible to detect some segments that probably identify the same activity but that they are not unified.

The next step will be to associate an activity to every segment in order to add more information for processing the data.

3.2.7. Activities

Until this point the only information used to process the data has been the longitude of the segments and the slope of the linear model. No specific activity has been associated to these segments although one of the main objectives has been to define the beginning and end of these activities looking at the points where an abrupt change in the slope is produced.

At this point, the limits are acceptably well defined and using one segment the whole duration of most of the activities is covered. For this reason it is possible to start associating an activity to every segment. The basic idea for this purpose is associate an activity depending on the value of the slope.

This operation leads to the requirement of defining thresholds to distinguish between activities. These thresholds have been obtained experimentally by analysing several datasets. However, in a later section they will be obtained using SVM (Support Vector Machines) learning algorithms.

The matlab function implemented is:

```
function a = activities(datem, b)
for i=1:length(datem)
 if((3500<=b{i}(2)) && (b{i}(2)<40000))
    a(i) = 1;    % 1 for inside
 elseif((-30000<=b{i}(2)) && (b{i}(2)<-100))
     a(i) = 2;    % 2 for out
 elseif(40000<=b{i}(2))
    a(i) = 3;    % 3 for exercise
 elseif(b{i}(2)<-30000)
    a(i) = 4;    % 4 for window open
 elseif((500<b{i}(2)) && (b{i}(2)<3500))
     if length(datem{1,i})<500
         a(i) = 1;    % 1 for inside
     else
         a(i) = 5;    % 5 for sleeping
     end
 else
    a(i) = 6;    % 6 for unknown
 end
end
```

This function creates a new vector (**a**) that stores the associated activity for every segment. This vector continues the correspondence that allows accessing the data, the linear model and the coefficient R^2 using the same index.

Each activity has a numeric value associated, which makes it easier to store this information in a vector. A special case occurs when associating the activity “sleep”. This activity has a slope very similar to the activity “inside” and in some cases segments which corresponds with the activity “inside” have a slope lower than usual so they are identified as “sleep”. However the activity sleep is always longer than the activity “inside” when the value of the slopes is similar. For longer segments concerning the activity “inside” the slope is not so close so this problem does not exist. The threshold defined for this longitude is 500 measures so any segment with a minor length and a slope that can be associated with the activity “sleep” will be marked as “inside”.

When no activity is recognized, a 6 is associated to indicate it. This situation appears when the window is open for a long time. It can be observed that the CO2 levels decrease suddenly for the activity “open window” (it has a big decreasing slope) so when this activity last for too long the outdoor CO_2 level (around 400 ppm) will be reached and the variation rate from this moment will be very low. Therefore, any activity done during this time will not be recognized because the amount of fresh air entering the room is high enough to cover all the other variations on the CO_2 concentration.

The shape of the lineal model for every activity is:

- **Inside**: it has a moderate increasing slope and the longitude is very variable as is the most common activity along with “out”.
- **Out**: it has a moderate decreasing slope and its duration is also very variable as the person can go out of the room for any reason. Although it is approximated to a linear model it can be better adapted to an exponential.
- **Window open:** The longitude is very short and the slope has a very high negative value. When the window is open the air from outside enters the room reducing the Co2 concentration to the outdoor levels (around 400ppm) in a short time.
- **Exercise**: The slope has a very high increasing rate due to the increasing physical activity of the person inside the room. The duration is usually short because the room is not equipped for long exercise sessions.
- **Sleep**: The slope has a low positive value. The production of CO2 while sleeping is minimum in a person [8]. The longitude is usually the longest as no interruption occurs while the person is sleeping. However this is much dependent on the person.

3.2.8. Unifying activities

After associate an activity to every segment it can be observed that some consecutive segments have the same activity associated. This happens when their slopes are not close enough to be unified by the function “unify” presented in section 3.2.3 but they are inside the same thresholds in the function “activities”.

The function “unifyAct” will unify the segments looking only at the activity associated. This way, a stronger but more controlled unification is implemented. This is the code for the function:

```
function [daten, co2n, R, b] = unifyAct(datem, co2m, b, a)

j=1;
daten{j} = datem{1};
co2n{j} = co2m{1};
for i=2:length(datem)
    if (a(i)==a(i-1))
      daten{j} = [daten{j}; datem{i}];
      co2n{j} = [co2n{j}; co2m{i}];
    else
      j=j+1;
      daten{j} = datem{i};
      co2n{j} = co2m{i};
    end
end

clearvars b;

for i=1:(length(daten))
    X = [ones(length(daten{i}),1) daten{i}];
    b{i} = X\co2n{i};
    y{i}=X*b{i};
    R(i) = 1 - sum((co2n{i} - y{i}).^2)/sum((co2n{i} -
          mean(co2n{i})).^2);
end
end
```

The performance of this function is similar to the others. It creates two new cells (**daten** and **co2n**) containing the new unified vectors of the input cells (**datem** and **co2m**). It checks the activity associated to every segment looking at the **a** vector and, if it is the same than the segment before, it unifies them and put them in the new cell (**daten** and **co2n**). If they are different the segment is copied in the new cell. After completing the new cells, the linear models and the coefficient of determination are calculated using linear regression.

Here is the resulting graph of using this new function:

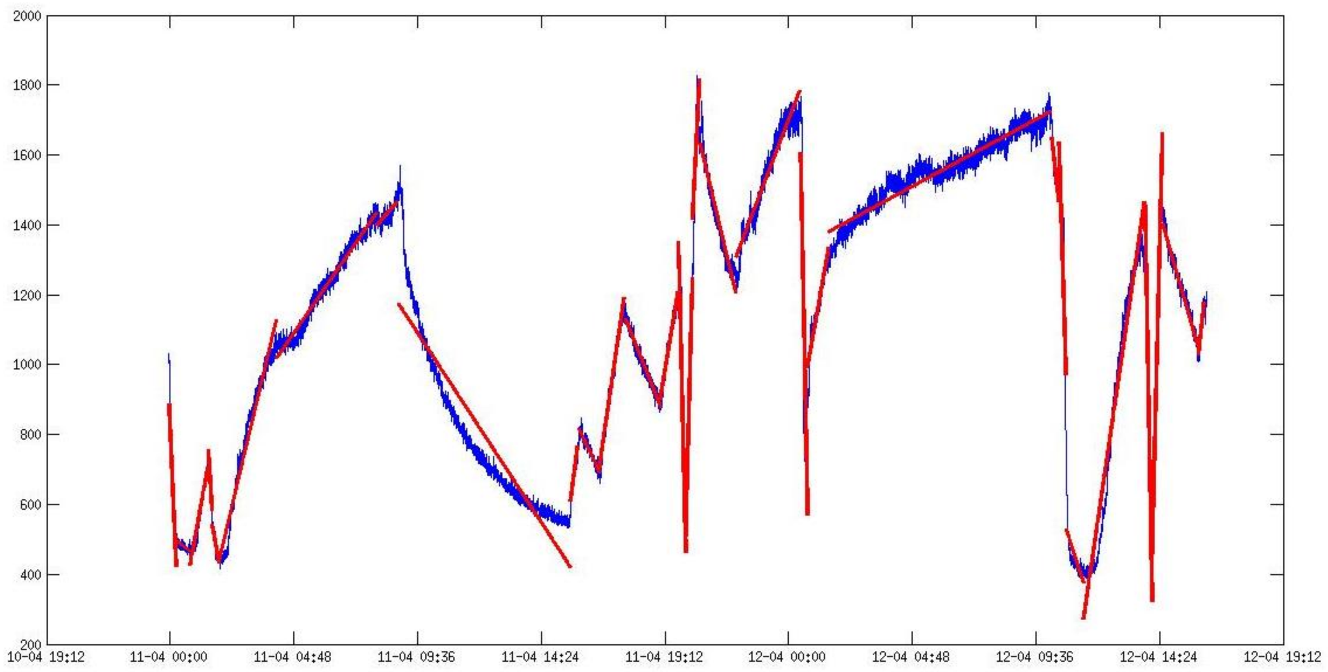

Figure 18

The quantity of unified segments is very big now. From 48 segments, only 30 larger segments are obtained. The linear models define very well the beginning and the end of the activities and also the type of activity. At this point, a good prediction of the activities can be done. The proportion of time when the predicted activity matches the real activity is the 86.1592 %. The method to obtain this score is shown in section 4.2. This prediction can be increased executing again some functions that will remove some errors still present.

3.2.9. Second division of the segments

The main part of the segments have very accurate limits, the linear model for every segment approximates very well the original data. Something that can seem not very accurate is the approximation of the activity sleeping whose shape is more similar to an exponential function rather than to a linear function. Nevertheless, this does not represent a problem because there is no other activity whose approximated linear model produces a slope of this magnitude. The other activity with a descendent slope is “open window” and the absolute value of the slope is much bigger so it will not be misinterpreted.

The objective now is defining even better the limits of the “short time” activities, like “open window” and “exercise”. For this purpose, the function “coefi” will be executed again but using a higher threshold. The function “coefi” checks the coefficient of determination of every segment and if it is lower than the threshold, divides the segment in two parts and construct a linear model for these two new segments that will be adapt better to the original data.

Using a higher threshold will divide some segments that were not divided in the first execution. Experimentally, it has been observed that the value that works better is 0.8. The reason for increasing it is that the main part of the segments are well adapted to the data so its coefficient of determination will be high and therefore they will not be divided. The candidates for been divided are the short segments which are not as precise as the long ones because the amount of data that it is used to create the linear model is smaller.

The result of applying this function again is shown in figure 19 where some corrected errors have been marked in green.

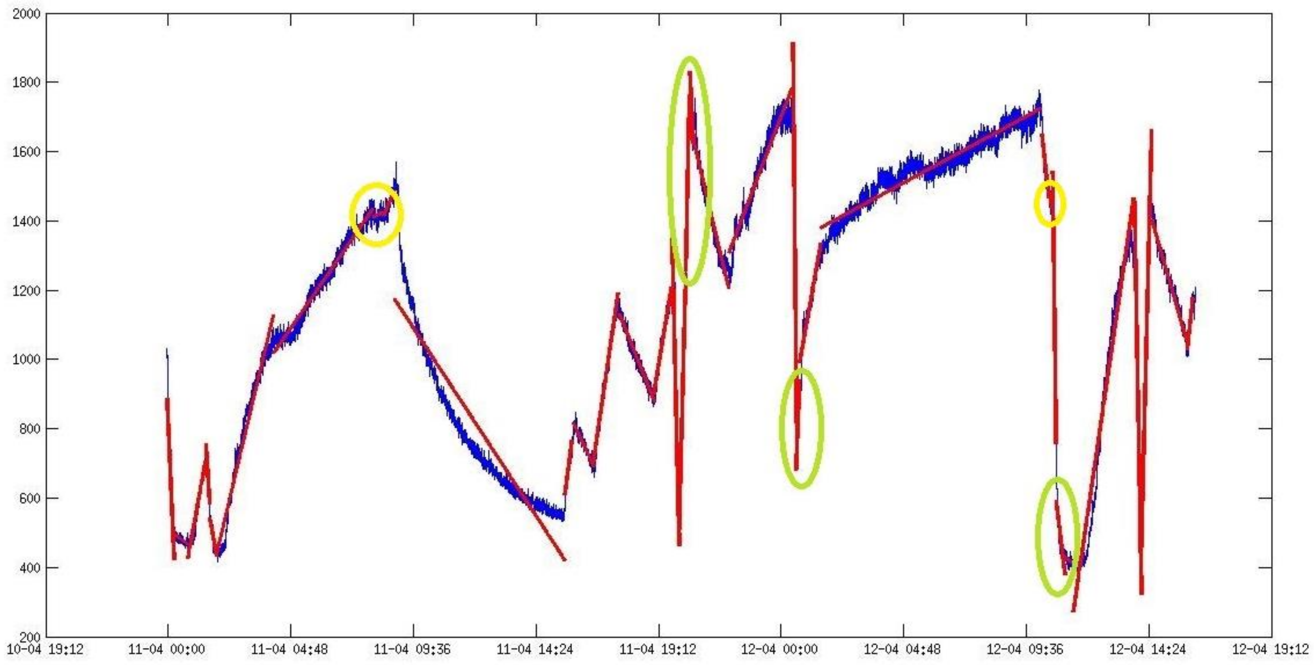

Figure 19

It can be seen that after applying this function the limits of the marked parts are more accurate to the original data than before. The prediction gives a score of 86.8512 % over the real activities. However some errors can appear during this process, like the new little segment marked in yellow. One efficient way to solve that is executing again the function “RemoveErrors”

3.2.10. Removing new errors

This function will deal with the very short error segments that appear after using the function “coefi”. These little segments are usually generated due to a very abrupt data. This function will unify these little segments with the neighbour whose slope is more similar and it will calculate the approximated linear model again.

Using this function again at this point will give the following result, where the error fixed is marked in yellow. It is difficult to appreciate it in the image because of the little longitude of the segment unified.

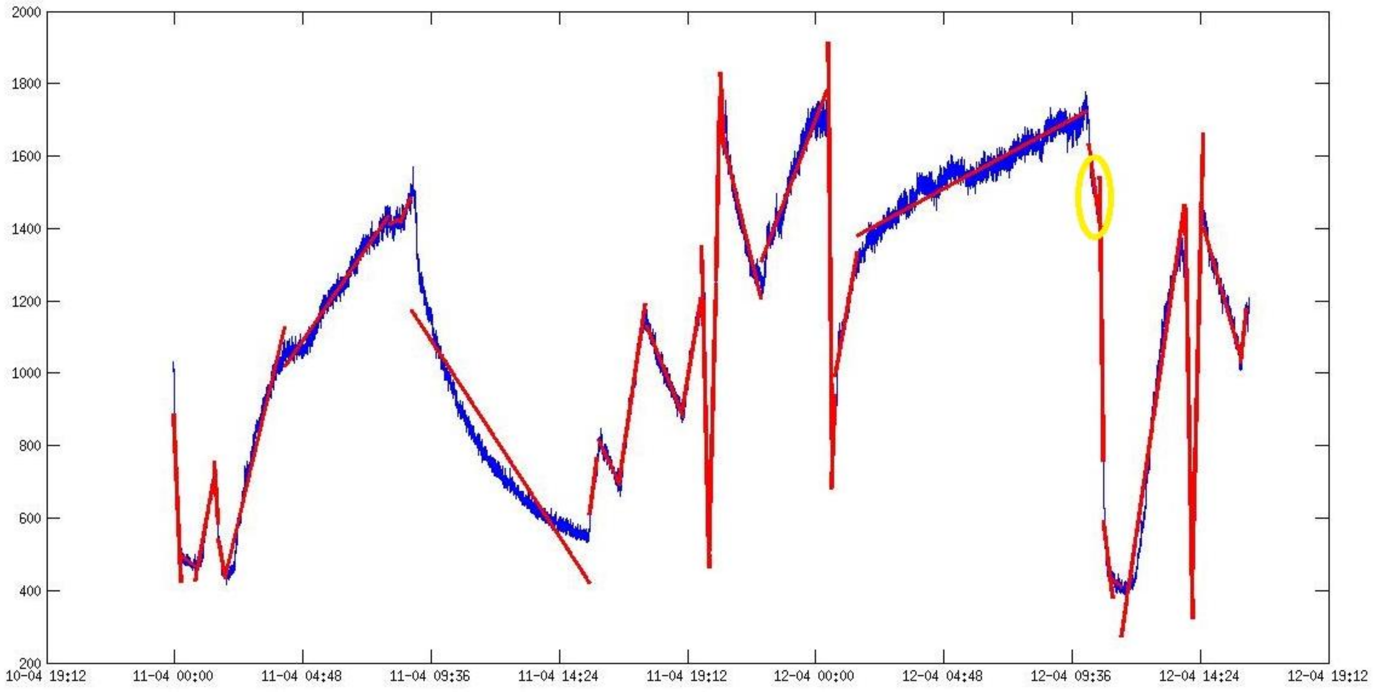

Figure 20

3.2.11. Second unification based on activities

In the steps before some new segments have been obtained. As these segments approximate better the original data, they are likely to be unified with their neighbours. To do it, the function “unifyAct” will be executed again. This function unifies the consecutively segments that have been associated the same activity. But before executing “unifyAct” is necessary to execute “activities” in order to associate a new activity to these new segments.

The obtained graph is shown in figure 21, where the unified segments have been circled in yellow.

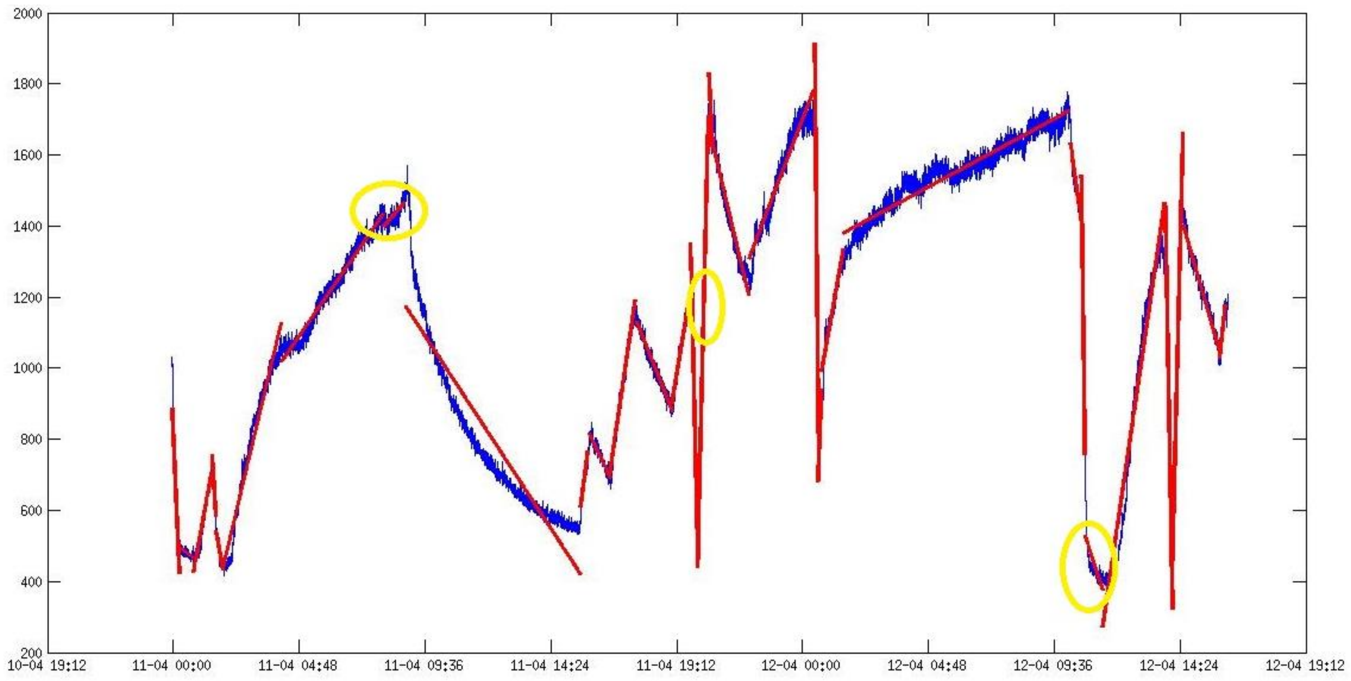

Figure 21.

Due to this process the number of segments is reduced from 34 to 31. After executing “unifyAct”, a new execution of “activities” is necessary in order to update the vector containing the information about activities (**a**).

3.2.12. Improving sleep detection

The last step will focus on improve the detection of the activity “sleep”. Sometimes this activity looks very similar to the activity “inside” in terms of slope, but the longitude is always bigger. Furthermore the shape of the graph while this activity is happening changes in an abrupt way. This can lead to create some segments with a relatively different slope from their neighbours, which will be incorrectly identified as “inside” or “out”, between two segments of the activity “sleep”.

The solution implemented to fix this error consists on checking when a segment whose associated activity is “inside” or “out” with a relatively low absolute value of its slope, occurs just after an activity “sleeping”. In these cases, it is very probable that this segment has been incorrectly identified and that it should represent the activity “sleeping”.

The code for implementing this function is:

```
function [daten, co2n, R, b] = sleep(datem, co2m, b, a)
j=1;
i=1;
while i<length(datem)
    if a(i)==5
        daten{j} = datem{i};
        co2n{j} = co2m{i};
        i=i+1;
        while (((a(i)==1) && (b{i}(2))<7000) || ((a(i)==2) &&
        (abs(b{i}(2))<900)) || (a(i)==6))
            daten{j} = [daten{j}; datem{i}];
            co2n{j} = [co2n{j}; co2m{i}];
            i=i+1;
        end
        j=j+1;
    else
      daten{j} = datem{i};
      co2n{j} = co2m{i};
      j=j+1;
      i=i+1;

    end
end

daten{j} = datem{i};
co2n{j} = co2m{i};

clearvars b;

for i=1:(length(daten))
    X = [ones(length(daten{i}),1) daten{i}];
    b{i} = X\co2n{i};
    y{i}=X*b{i};
    R(i) = 1 - sum((co2n{i} - y{i}).^2)/sum((co2n{i} -
          mean(co2n{i})).^2);
end
end
```

This function creates two new cells (**daten** and **co2n**) which will store the original data (filtered) reorganized in the new segments. It checks, for every segment of the input data (**datem** and **co2m**), if its associated activity is “sleeping” (activity number 5). When it detects this activity, checks if the next segments represent an activity “inside” with an slope lower than 7000 or an activity “out” with an slope bigger than -900. When this happens, the data of these segments is unified with the “sleeping” segment and it is stored in the new cells. It also happens when the segment after the activity “sleeping” has been identified has “unknown”. After this processing, the new linear models for the new segments are calculated using linear regression as in the functions before.

After using this function it is possible to obtain two consequent segments representing the activity “sleeping”. For this reason a new unification using the vector of activities should be implemented. The functions “activities” and “unifyAct” will be executed for this purpose.

After this final unification, the last step before presenting the results is to update the vector of activities (**a**), executing again the function “activities”.

The final graph is shown in figure 22. The segment representing the activity “sleeping” that has been unified is marked in yellow.

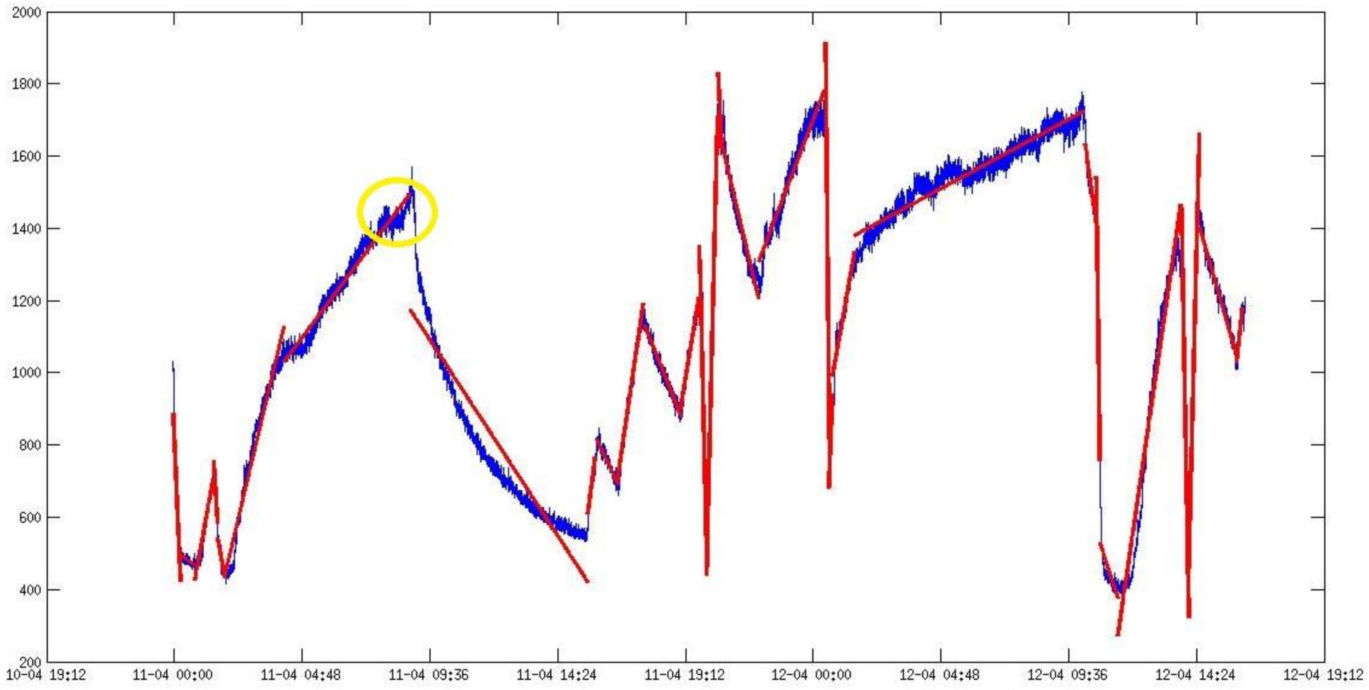

Figure 22

This dataset that is being used for the explanation does not have a lot of problems to detect the activity “sleep” as only one segment has been unified after executing this function. To give an idea of how important is this function and how much it improves the detection of the activity sleep, another dataset where the function sleep has not been executed yet is shown in figure 23.

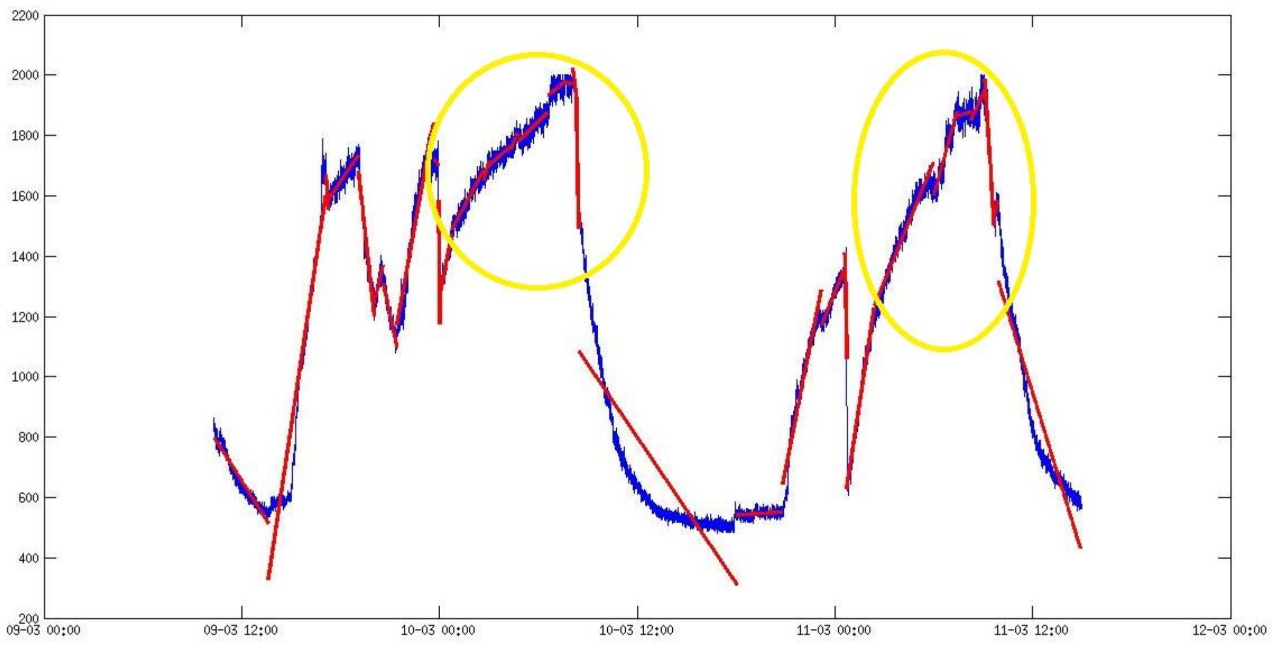

Figure 23

The yellow circles mark the part where the real activity is “sleep”. It can be seen how much this activity is fragmented and how many error segments appear due to the abrupt data. At this moment, the score obtained for the prediction is 69.4444 %. After applying the function “sleep” the result is:

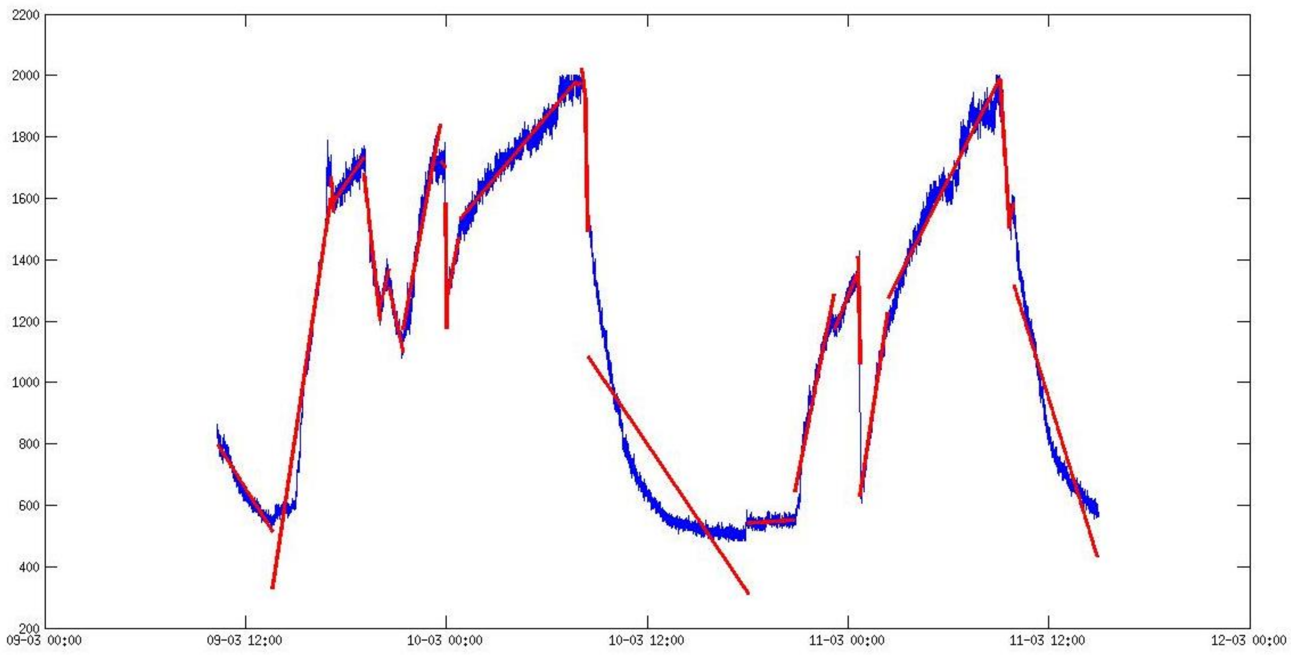

Figure 24.

It can be seen that the segments causing the problems before have been unified and now they have a slope that corresponds with the activity sleeping. After this improvement the score obtained is 78.5714 % which represents a very good improvement in the detection of “sleep”.

3.2.13. Results

The last function will plot the graph and will write on screen the exact times when an activity starts and ends. The code is:

```
function b = results(date,co2,datem,co2m,a)
dcm_obj = datacursormode(figure(figure));
set(dcm_obj,'UpdateFcn',@myupdatefcn)
plot(date,co2)
datetick('x','dd-mm HH:MM', 'keepticks')
hold on

for i=1:length(datem)
    X = [ones(length(datem{i}),1) datem{i}];
    b{i} = X\co2m{i};
    y2{i}=X*b{i};
    plot(datem{i},y2{i},'r','LineWidth',3)

    fprintf(['from ' datestr(datem{i}(1))  ' to '])
    fprintf([datestr(datem{i}(end))  ': '])

    if a(i)==1
        fprintf('inside \r\n')
    elseif a(i)==2
        fprintf('out \r\n')
    elseif a(i)==3
        fprintf('exercise \r\n')
    elseif a(i)==4
        fprintf('window open \r\n')
    elseif a(i)==5
        fprintf('sleeping \r\n')
    else
        fprintf('unknown \r\n')
    end
end
```

This function uses the first value of every vector of **datem** to obtain the time when the associated activity starts. The values on **datem** are stored in a serial numeric format, which represents the whole and fractional number of days from a fixed, pre-set date (January 0, 0000) in the ISO calendar. To convert this number to a string with a format dd/mm/yy HH:MM:SS the function “datestr” is used. At the same, it access the vector **a** and writes the name of the activity corresponding to the number in the current position.

The graph in figure 25 represents the final results obtained. The limits between activities are very well identified and the errors have been corrected as much as possible. It is clear that the linear models approximate very well the original data and that the information about the activities can be guess simply looking at the graph.

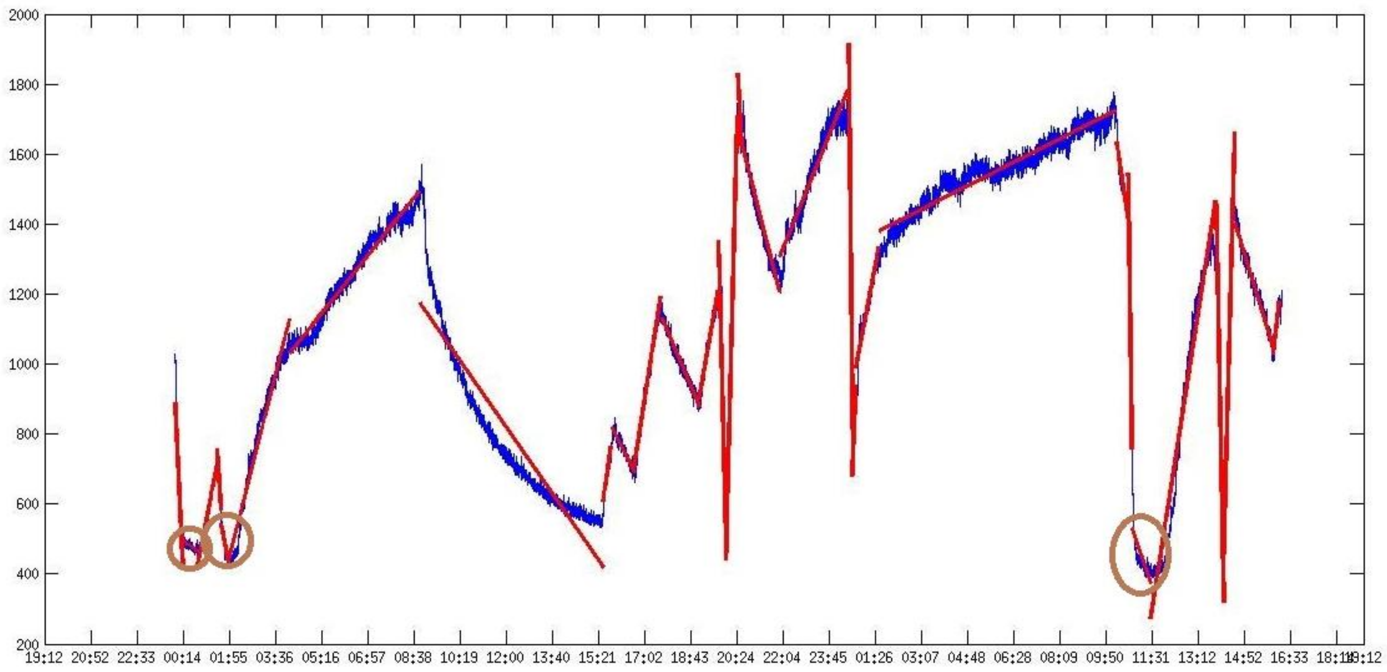

Figure 25

There are some little errors still present. They are marked with a brown circle and are due to the slow transition from the activity “window open” to the activity “inside”. The score obtained now is 88.9273 %, which represents a very good identification of the activities.

From the original data, the points in time where the activity changes have been identified and also the type of activity between these points. All this process does not require human interaction beyond providing the original data. The program performing all these operations is shown below:

```
function [b, datem, co2m, R, a] = ActivityDetector(date,co2)
[dateF, co2F] = butfilter(5,0.04,co2,date);
[datem, co2m, b, R] = divide(dateF,co2F);
[datem, co2m, R, b] = unify(datem, co2m, b,3000);
[datem, co2m, R, b] = coefi(datem,co2m,R,0.4);
[datem, co2m, R, b] = unify(datem, co2m, b,3000);
[datem, co2m, R, b] = RemoveErrors(datem,co2m,b);
a = activities(datem, b);
[datem, co2m, R, b] = unifyAct(datem, co2m, b, a);
[datem, co2m, R, b] = coefi(datem,co2m,R,0.8);
[datem, co2m, R, b] = RemoveErrors(datem,co2m,b);
a = activities(datem, b);
[datem, co2m, R, b] = unifyAct(datem, co2m, b, a);
a = activities(datem, b);
[datem, co2m, R, b] = sleep(datem, co2m, b, a);
a = activities(datem, b);
[datem, co2m, R, b] = unifyAct(datem, co2m, b, a);
a = activities(datem, b);
[b] = results(date,co2,datem,co2m,a);
end
```

The output prediction and the ground truth information are:

```
Predicted Activities
Time: 10-Apr-2016 23:56:36 Activity: window opened
Time: 11-Apr-2016 00:13:18 Activity: out
Time: 11-Apr-2016 00:46:43 Activity: inside
Time: 11-Apr-2016 01:28:29 Activity: window opened
Time: 11-Apr-2016 01:36:51 Activity: out
Time: 11-Apr-2016 01:53:34 Activity: inside
Time: 11-Apr-2016 04:07:16 Activity: sleeping
Time: 11-Apr-2016 08:51:25 Activity: out
Time: 11-Apr-2016 15:32:34 Activity: inside
Time: 11-Apr-2016 15:49:17 Activity: out
Time: 11-Apr-2016 16:39:26 Activity: inside
Time: 11-Apr-2016 17:37:56 Activity: out
Time: 11-Apr-2016 19:01:30 Activity: inside
Time: 11-Apr-2016 19:43:17 Activity: window opened
Time: 11-Apr-2016 20:00:00 Activity: exercise
Time: 11-Apr-2016 20:25:05 Activity: window opened
Time: 11-Apr-2016 20:33:26 Activity: out
Time: 11-Apr-2016 21:57:00 Activity: inside
Time: 12-Apr-2016 00:27:25 Activity: window opened
Time: 12-Apr-2016 00:35:47 Activity: exercise
Time: 12-Apr-2016 00:44:08 Activity: inside
Time: 12-Apr-2016 01:34:16 Activity: sleeping
Time: 12-Apr-2016 10:12:23 Activity: out
Time: 12-Apr-2016 10:37:28 Activity: window opened
Time: 12-Apr-2016 10:45:49 Activity: out
Time: 12-Apr-2016 11:27:36 Activity: inside
Time: 12-Apr-2016 13:49:41 Activity: window opened
Time: 12-Apr-2016 14:06:24 Activity: exercise
Time: 12-Apr-2016 14:31:28 Activity: out
Time: 12-Apr-2016 15:55:02 Activity: inside
```

```
Ground truth
time: beginning              activity: window open
time: '11-Apr-2016 00:55' activity: inside
time: '11-Apr-2016 01:32' activity: window open
time: '11-Apr-2016 02:14' activity: inside
time: '11-Apr-2016 04:07' activity: sleeping
time: '11-Apr-2016 08:57' activity: out
time: '11-Apr-2016 15:30' activity: inside
time: '11-Apr-2016 15:56' activity: out
time: '11-Apr-2016 16:22' activity: inside
time: '11-Apr-2016 17:33' activity: out
time: '11-Apr-2016 19:04' activity: inside
time: '11-Apr-2016 19:49' activity: window open
time: '11-Apr-2016 20:01' activity: exercise
time: '11-Apr-2016 20:36' activity: out
time: '11-Apr-2016 21:46' activity: inside
time: '12-Apr-2016 00:30' activity: window open
time: '12-Apr-2016 00:35' activity: inside
time: '12-Apr-2016 01:34' activity: sleeping
time: '12-Apr-2016 10:11' activity: out
time: '12-Apr-2016 10:40' activity: window open
time: '12-Apr-2016 11:53' activity: inside
time: '12-Apr-2016 13:52' activity: window open
time: '12-Apr-2016 14:07' activity: exercise
time: '12-Apr-2016 14:23' activity: out
time: '12-Apr-2016 15:40' activity: inside
```

Other datasets

More datasets are analysed in order to check the performance of the method developed.

1. Sleep example

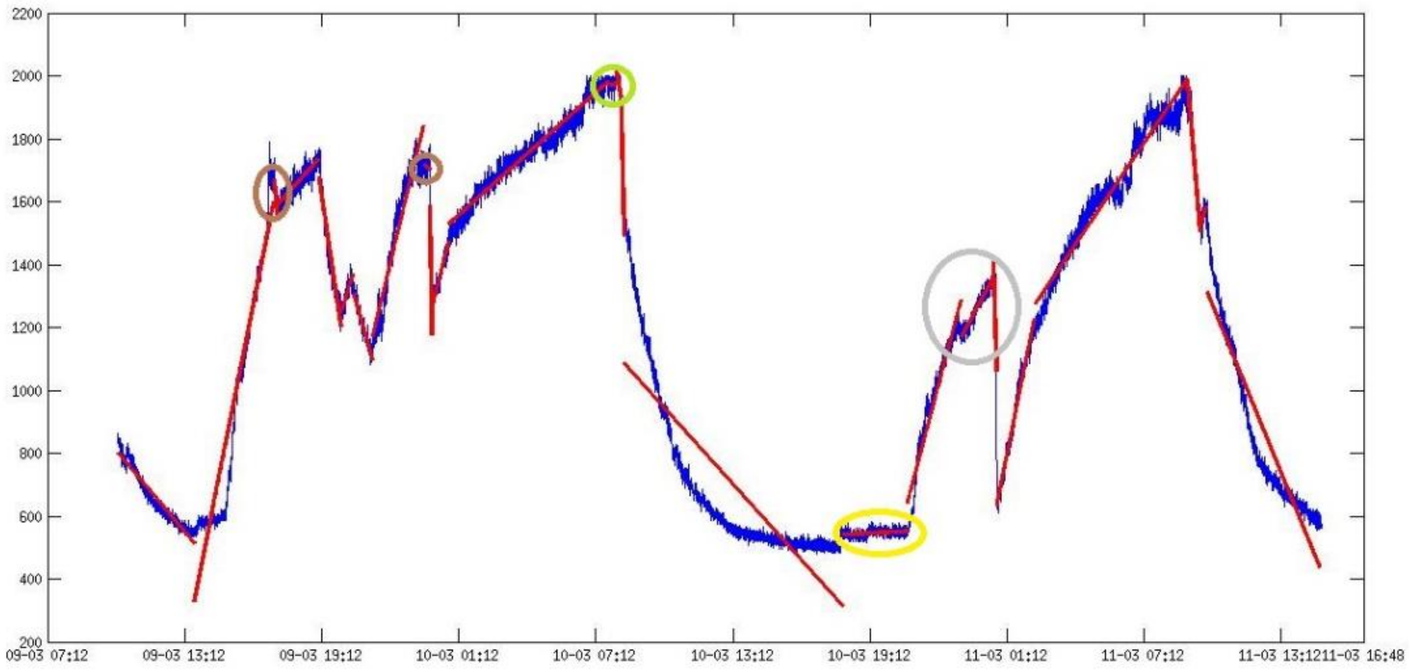

Figure 26

The problems observed here have been marked. In the green circle there is a small segment that should be identified with the activity "sleep" but that has not been unified with the sleep segment due to its slope. In the yellow circle the CO2 concentration has reached the minimum value so although it is clear that the activity in this moment is "out" (because the window has not been opened) it has been identified as "unknown". The grey circle indicates a segment identified as "sleeping" that corresponds to "inside". Other errors due to the precision of the program have been marked in brown. The first of them refers to a segment incorrectly identified as "out" that also induce the next segment to be erroneously identified as "sleep". The score obtained in this case is 78.5714%.

The output prediction and the ground truth information are shown below:

Predicted Activities

```
Time: 09-Mar-2016 10:17:13 Activity: out
Time: 09-Mar-2016 13:37:47 Activity: inside
Time: 09-Mar-2016 17:06:43 Activity: out
Time: 09-Mar-2016 17:15:05 Activity: sleeping
Time: 09-Mar-2016 19:03:43 Activity: out
Time: 09-Mar-2016 20:02:12 Activity: inside
Time: 09-Mar-2016 20:35:38 Activity: out
Time: 09-Mar-2016 21:25:46 Activity: inside
Time: 09-Mar-2016 23:39:27 Activity: out
Time: 09-Mar-2016 23:56:09 Activity: window open
Time: 10-Mar-2016 00:00:20 Activity: inside
Time: 10-Mar-2016 00:46:17 Activity: sleeping
Time: 10-Mar-2016 07:39:54 Activity: out
Time: 10-Mar-2016 07:52:26 Activity: inside
Time: 10-Mar-2016 08:04:58 Activity: out
Time: 10-Mar-2016 08:17:30 Activity: window open
Time: 10-Mar-2016 08:25:52 Activity: out
Time: 10-Mar-2016 18:02:30 Activity: unknown
Time: 10-Mar-2016 20:49:38 Activity: inside
Time: 10-Mar-2016 23:11:41 Activity: sleeping
Time: 11-Mar-2016 00:35:16 Activity: window open
Time: 11-Mar-2016 00:43:37 Activity: inside
Time: 11-Mar-2016 02:23:54 Activity: sleeping
Time: 11-Mar-2016 09:04:58 Activity: out
Time: 11-Mar-2016 09:38:24 Activity: inside
Time: 11-Mar-2016 09:55:07 Activity: out
```

Ground truth

```
time: beginning                activity: out
time: '09-Mar-2016 15:00' activity: inside
time: '09-Mar-2016 16:54' activity: out
time: '09-Mar-2016 17:14' activity: inside
time: '09-Mar-2016 19:01' activity: out
time: '09-Mar-2016 19:57' activity: inside
time: '09-Mar-2016 20:20' activity: out
time: '09-Mar-2016 21:15' activity: inside
time: '09-Mar-2016 23:54' activity: window open
time: '09-Mar-2016 23:57' activity: inside
time: '10-Mar-2016 00:46' activity: sleeping
time: '10-Mar-2016 08:15' activity: out
time: '10-Mar-2016 20:42' activity: inside
time: '11-Mar-2016 00:40' activity: window open
time: '11-Mar-2016 02:24' activity: sleeping
time: '11-Mar-2016 09:17' activity: out
time: '11-Mar-2016 09:41' activity: inside
time: '11-Mar-2016 10:00' activity: out
```

2. Third example

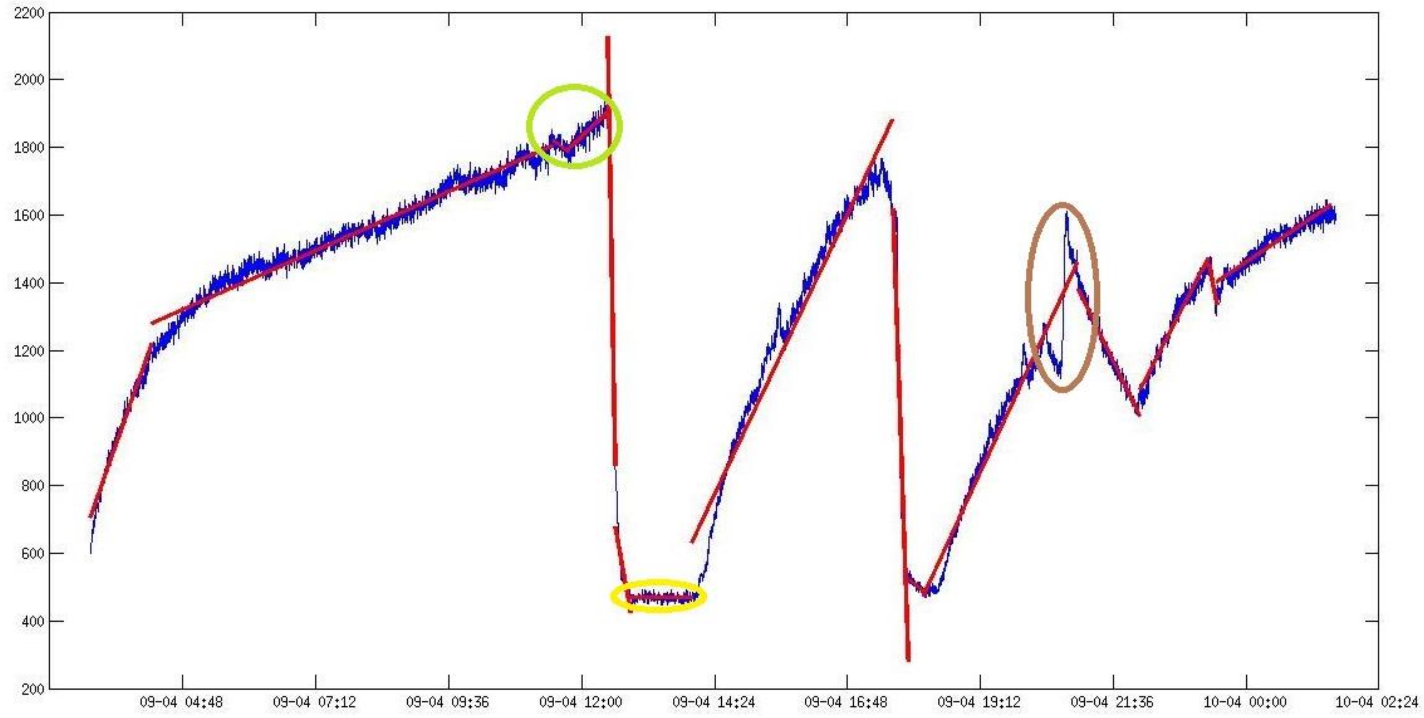

Figure 27

In this case it is possible to observe some errors. In the green circle, the abrupt data during the activity "sleep" creates a short segment with a slope that has been incorrectly identified as the activity "out". In the yellow circle, the CO2 concentration has achieved the outdoors level but the window was still open so there is no variation on the Co2 concentration. Therefore, it is not possible to identify any activity. Finally, the brown circle marks an activity "exercise" that has not been detected due to the short duration. The total score obtained is 83.2298 %, similar to the obtained with the example dataset.

Predicted Activities

Time: 09-Apr-2016 03:07:58 Activity: inside

Time: 09-Apr-2016 04:14:50 Activity: sleeping

Time: 09-Apr-2016 11:29:22 Activity: out

Time: 09-Apr-2016 11:46:05 Activity: inside

Time: 09-Apr-2016 12:27:53 Activity: window open

Time: 09-Apr-2016 12:36:14 Activity: out

Time: 09-Apr-2016 12:52:57 Activity: unknown

Time: 09-Apr-2016 13:59:48 Activity: inside

Time: 09-Apr-2016 17:37:05 Activity: window open

Time: 09-Apr-2016 17:53:48 Activity: out

Time: 09-Apr-2016 18:10:31 Activity: inside

Time: 09-Apr-2016 20:57:39 Activity: out

Time: 09-Apr-2016 22:04:31 Activity: inside

Time: 09-Apr-2016 23:19:44 Activity: out

Time: 09-Apr-2016 23:28:05 Activity: sleeping

Ground truth

time: beginning activity: inside
time: '09-Apr-2016 04:15' activity: sleeping

time: '09-Apr-2016 12:30' activity: window open
time: '09-Apr-2016 14:00' activity: inside

time: '09-Apr-2016 17:41' activity: window open

time: '09-Apr-2016 18:25' activity: inside

time: '09-Apr-2016 20:19' activity: out

time: '09-Apr-2016 20:36' activity: exercise

time: '09-Apr-2016 20:42' activity: out

time: '09-Apr-2016 22:00' activity: inside

time: '09-Apr-2016 23:17' activity: out

time: '09-Apr-2016 23:28' activity: sleeping

The last analysis will check how well every activity is detected separately. For this purpose, the function “score1activity” explained in section 4.2 will be used. This function gives the percentage of segments of **g** (identified activities for segments of 25 points) corresponding to one activity that match the segments in **G**. Furthermore gives the percentage of segments in **g** different from this activity that are also different in **G**.

	Main dataset		**Sleep example**		**Third example**	
	Correctly identified	**Correctly not identified**	**Correctly identified**	**Correctly not identified**	**Correctly identified**	**Correctly not identified**
Inside	87.5776	96.4029	66.0714	90.6463	98.2609	88.8889
Out	96.9325	90.8434	80.6180	95.7500	65.3846	95.6081
Exercise	76.9231	99.2920	0	100	0	99.6885
Window open	27.0833	98.6792	3.8462	99.4521	12.9032	99.3127
Sleep	99.4819	100	94.6602	91.6364	89.9329	100

Table 1

The main reason of failure is that the start and end of the activities are not the same in the ground truth and in the predicted activities. The main part of the components of **g** and **G** that do not match are situated in the transition from one activity to another. The reason for the 0 % identified in the activity “exercise” in “third example” is that there is only one segment in G that has not been identified. In “Sleep example” there is no segment related with this activity.

All the activities are identified independently with a precision from 65% to 99% except the activity “window open”. This happens because the duration of this activity is usually very short so by the time the components of **g** and **G** are starting to match, the activity has almost finished.

The activity sleep has very good results for the correctly not identified field. There are two reasons for this. The detected activity is usually shorter than the real activity. Also, as it is impossible to know for sure when the person enters deep sleep, the start time in the ground truth has been introduced looking at the predicted activities so the start time of this activity is the same in both.

3.2.14. SVM thresholds

In the section before, the thresholds that define the different activities have been obtained experimentally. Various datasets have been analysed and the program has been executed with various thresholds. The values used have been found to give the better results. However these values are adapted for the room that has been monitored and are not portable to a different room.

The aim of this section is introduce the usage of SVM thresholds in order to make this program portable. SVM (Support Vector Machines) are learning algorithms used for classification tasks. In an initial set of points, every point belongs to one of two possible categories. The SVM algorithm will create a classification model that will assign one of the two categories to any new point.

To create this model it needs a set of data (which will be called training data) along with the ground truth information. The ground truth information refers to the perfect classification of the training data (see section 4.1). Once the model has been build, any other data will be classified using this model.

In matlab, the function used to create this model is "svmtrain(trainingdata,groundtruth)" where **trainingdata** is a vector containing the data used to build the model and **groundtruth** is a cell containing a string in every component corresponding to the category of the same component in **trainingdata**. This function returns a matlab structure containing all the information about the SVM model. After the model is created, the function used to classify new data is "svmclassify(SVMModel,newdata)". **SVMModel** is the model created with the function "svmtrain" and **newdata** is a vector containing the new data to be classified. This returns a cell containing strings that correspond to the category associated to every component of **newdata.**

It can be deduced that the most important part for obtaining a good performance resides in create a good model. The quality of the model basically depends on the quality of the correspondence between the training data and the ground truth.

Generally, the SVM algorithms do not distinguish between more than two activities. For this program it is necessary to distinguish between 5 categories, which means that 4 thresholds are necessary. The activity "unknown" is not taken into account as it is not a real activity and its only purpose is to create a margin for the cases when the variation rate of the CO2 concentration is very small. Therefore, its limits will remain between -500 and 100. In order to use 4 SVM thresholds, 4 different models will be necessary. Each of this models will classify the new data between one activity and all the others.

The most difficult part is to obtain some data that corresponds perfectly with the ground truth. The ground truth about the activities can be achieved from the notes that were taken during the monitoring period. Nevertheless the data corresponding to the increasing or decreasing rates of CO2 is not perfectly accurate with the ground truth. The only way to obtain this data is through the program previously design or defining the segments manually and then calculating the slopes using linear regression. Any of these methods will have some errors because the ground truth information will not adapt perfectly and therefore the model will be distorted.

The option that has been found to give better results is to adapt the ground truth to the obtained increasing and decreasing rates. This means that instead of using the vector **G** containing the real activities, the vector **g** containing the predicted activities along with the vector **b** containing the slopes of these predicted activities will be used to create the SVM model.

The function “svmclassify” could be used to directly classify the slopes of the segments in different activities. However, it is also possible to classify a set of different slope values in order to find the threshold calculated by the SVM algorithm.

Another problem to solve is the necessity of two thresholds (lower and upper) for 3 of the 5 possible activities. The SVM function can only give one threshold. One solution consists on start the process classifying one of the activities that only requires one threshold and remove this activity from the training data so that the next activity also needs only one threshold.

For example, if the SVM model for the activity “window open” is created first, the result will be a SVM model containing the threshold that differentiates the activity “window open” from all the others. As this activity uses the lowest values of the slope, all the segments whose slope value is lower than the threshold will be classified as “window open”. Once the SVM model for this activity has been created, the next model will be for the activity “out”. The possible values for the slope of “out” are the lowest after the ones for “window open” so if the activity “window open” is removed from the training data, the activity “out” will only need one threshold (upper). Creating the SVM model for this activity now will give the upper threshold and the lower threshold coincides with the upper threshold of “open window”. The same process can be repeated starting from the activity “exercise” to find the two thresholds of the activity “inside”. Finally, the lower threshold of “inside” will be the upper threshold of “sleep”.

The function “deleteActivities” performs the elimination of the activities from the training data. It returns the variables **datem**, **b** and **a** after removing the information for the activity act.

```
function [daten, bn, an] = deleteActivities(datem, b, a, act)

k=1;
    for j=1:length(datem)
        if(a(j)~=act)
            daten{k}=datem{j};
            bn{k}=b{j};
            an(k)=a(j);
            k=k+1;
        end
    end
end
```

The function that creates the four SVM models is shown below:

```
function SVMModel = training(datem,b,a)

[daten, bn, an] = deleteActivities(datem, b, a, 3);
slopes25 = slope25m(daten,bn);
g = groundg(daten,an);
categories = oneactivity(1,g)';
SVMModel(1) = svmtrain(slopes25',categories);

[daten, bn, an] = deleteActivities(datem, b, a, 4);
slopes25 = slope25m(daten,bn);
g = groundg(daten,an);
categories = oneactivity(2,g)';
SVMModel(2) = svmtrain(slopes25',categories);

slopes25 = slope25m(datem,b);
g = groundg(datem,a);
categories = oneactivity(3,g)';
SVMModel(3) = svmtrain(slopes25',categories);

slopes25 = slope25m(datem,b);
g = groundg(datem,a);
categories = oneactivity(4,g)';
SVMModel(4) = svmtrain(slopes25',categories);

end
```

The first used function, “slope25m” (explained in appendix 2), divides all the data in segments of 25 points and returns a vector containing the slope of every segment. After that, the function “groundg” obtains the vector with the predicted activities, **g**, for this segments. It works exactly equal to the function “ground” shown in section 4.1 but only returns the result for **g.**

The next step obtains the cell with the correspondent categories that distinguish between the activity whose model is being created (called “current”) and the other 4 activities (called “rest”). This operation is perform by the function “oneactivity” (also explained in appendix 2). With this information it can create the SVM model for the current activity. The output of the function “training” is a cell grouping the 4 SVM models.

Once the SVM models are created it is possible to identify the calculated thresholds following the next steps:

- Create a vector containing numbers that will represent the possible slopes of the different activities.

```
rates = -40000:1:40000;
```

- Use the function “svmclassify” to assign a category (“current” or “rest”) to every component of **rates.** The value of the variable **i** is the number of the activity.

```
class = svmclassify(SVMModel(i),rates');
```

- Use the function “returnNumbers” (explained in in appendix 2) to convert the cell **class** to a vector with the number of the activity in the same positions than the category “current” in the cell.

```
act = returnNumbers(class,i);
```

- Plot the result to visualize the threshold

```
plot(rates,act)
```

For the activity 1, “inside”, the following graph is obtained:

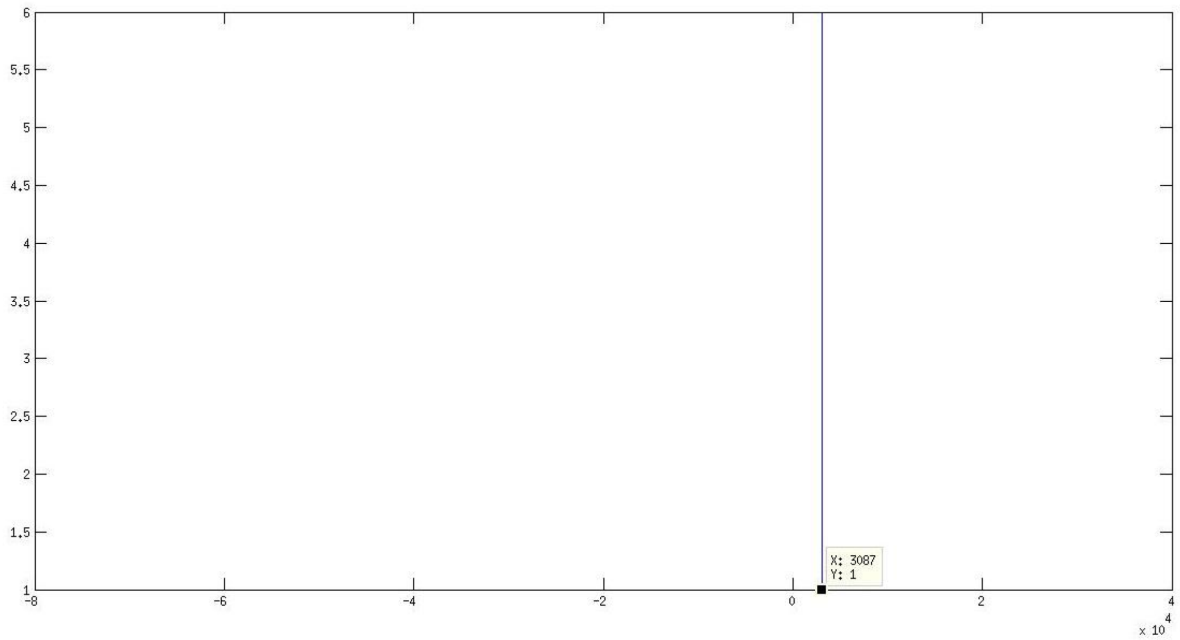

Figure 28

Following the same procedure for the other 4 activities, the following thresholds are obtained:

Activity	Threshold
1 “inside”	3087 (lower)
2 “out”	-876 (upper)
3 “exercise”	31155 (lower)
4 “window open”	-21584 (upper)
5 “sleep”	3087 (upper)

Table 2

These thresholds have been obtained using the dataset used during the explanation of all the other processess (the training data). To check the performance of the SVM thresholds another dataset should be used. The other two examples seen before will be used to compare the performance between the experimental thresholds and the SVM thresholds.

For the “sleep example”, the obtained graph now is:

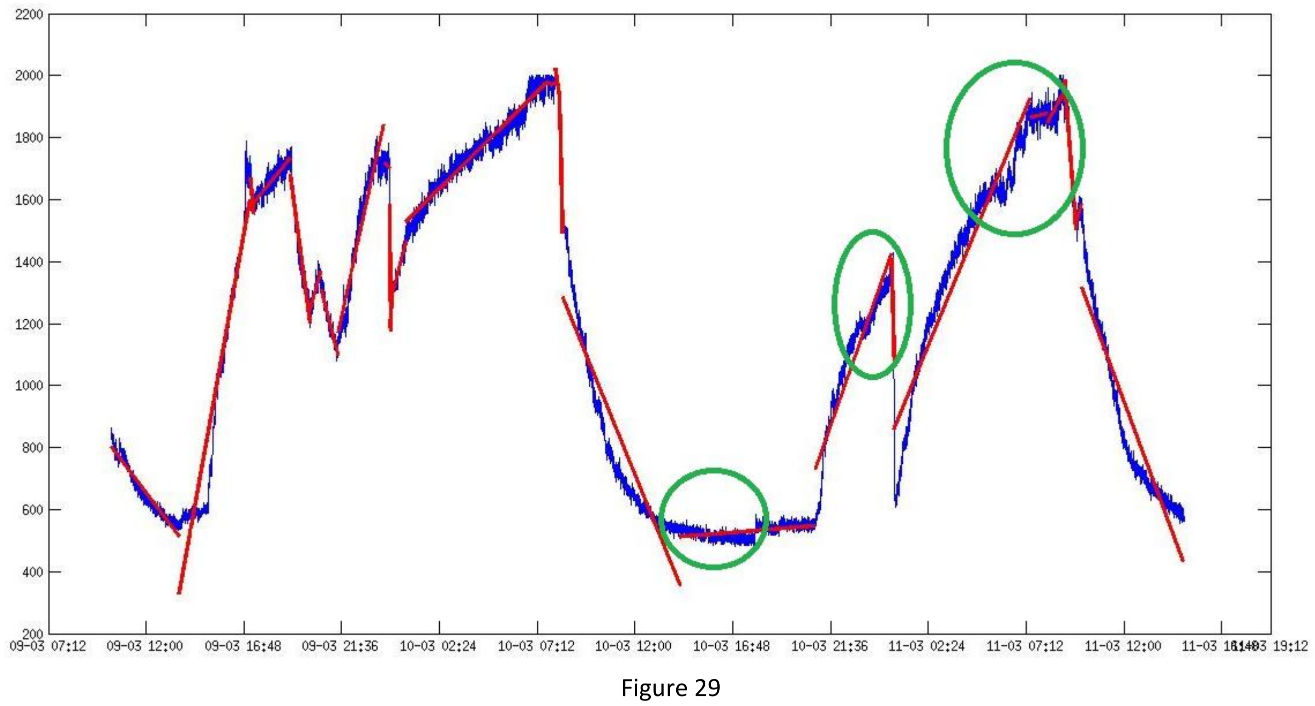

Figure 29

The performance has decreased, some new errors have appeared. They are marked with a green circle. The detection of the activity “sleep” is worse now, the two last circles correspond to parts of the activity “sleep” that have not been detected. The score has decrased from 78.5714% to 61.1111%.

From the individual analysis of the activities. It can be seen that the performance has decrease in all the activities with the exception of “window open”, where it is equal, and “inside”, where it has increased. The worst result is for “sleep” whose score has decrease around 50%.

	SVM Thresholds		Experimental thresholds	
	Correctly identified	Correctly not identified	Correctly identified	Correctly not identified
Inside	77.3810	76.3605	66.0714	90.6463
Out	65.1685	95.7500	80.6180	95.7500
Exercise	0	99.8677	0	100
Window open	3.8462	99.4521	3.8462	99.4521
Sleep	48.0583	95.2727	94.6602	91.6364

Table 3

However, “third example” will shows some different results:

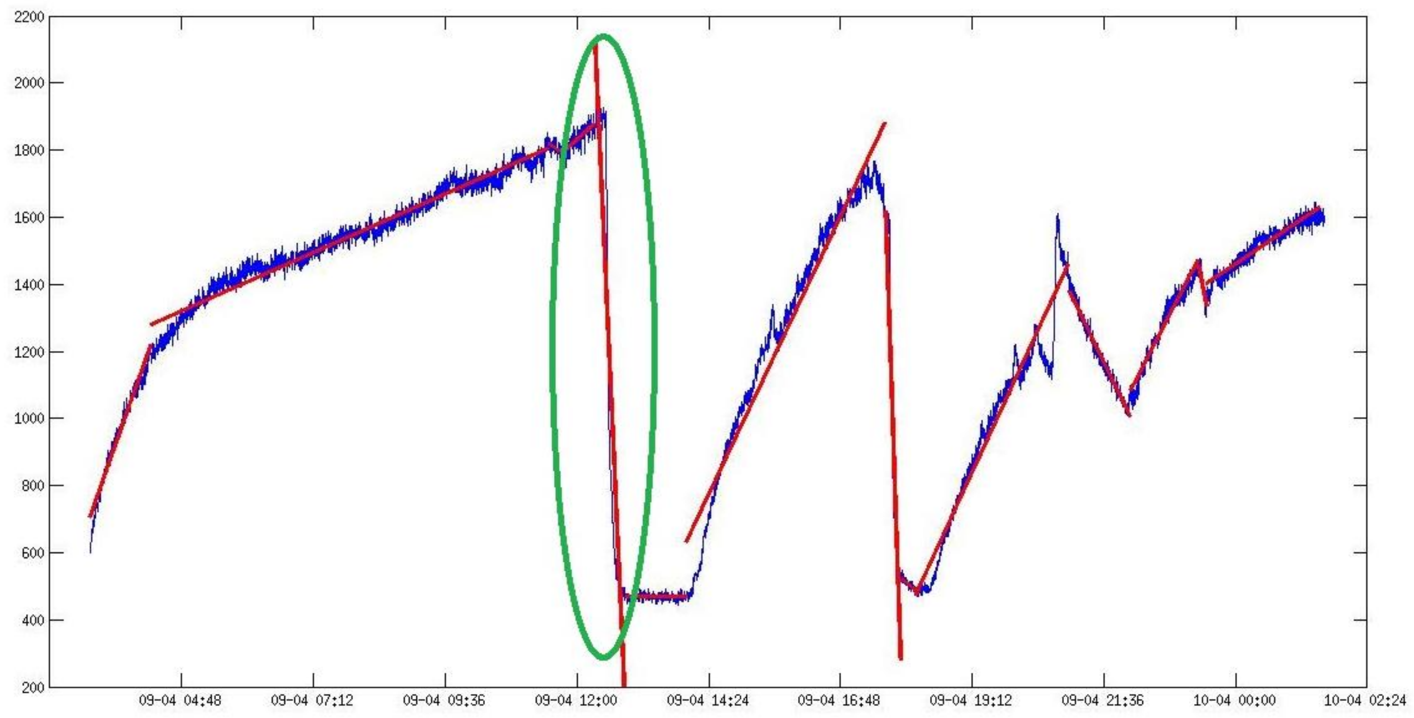

Figure 30

The only change is the activity “window open” marked in green. The detection has improved now and the erroneus activity “out” that was detected before just after this “window open” activity does not appear now. The general score obtained now is 84.4720%, slightly better than the 83.2298% obtained before.

The individual score of the activities is:

	SVM Thresholds		Experimental thresholds	
	Correctly identified	**Correctly not identified**	**Correctly identified**	**Correctly not identified**
Inside	98.2609	89.8551	98.2609	88.8889
Out	65.3846	96.9595	65.3846	95.6081
Exercise	0	100	0	99.6885
Window open	25.8065	98.6254	12.9032	99.3127
Sleep	89.9329	100	89.9329	100

Table 4

The accuracy detecting the activities has slightly increase. The main part is due to the increase of the accuracy in “window open”. From the graph, it can be observed that this activity is predicted during more time. As this activity is very short, the correctly identified field has doubled its value whereas the correctly not identified percentage has slightly decrease.

Using more data to create a better SVM model would improve these thresholds but the important idea is that the method to obtain these thresholds is not experimental and is more legitimate. They are obtained using complex algorithms that ensure that they will be optimal, instead of being the result of multiple experiments, which gives values that although they can be very similar, they are improbable to be the optimal. If the monitoring environment changes, this algorithm provides a method to determine the new optimal thresholds.

3. Ground truth and score

This section will deal with the performance of the activity detector analysing the number of correct activities predicted.

First, it is necessary to obtain the information of the real activities occurred inside the room in order to compare it with the predicted result. This real information about the activities is called ground truth and the objective of the activity detector program is to obtain a predicted information as close as possible to this ground truth.

3.1. Ground truth

In order to compare ground truth information with predicted information, the data will be divided in small segments containing 25 points each. Each of these segments will have a predicted activity and a ground truth activity associated that are stored in a vector. The number of matches between these two vectors will be the score obtained.

Three vectors are necessary for this process:

- **I**: once the original data has been divided in segments of 25 points, every component of this vector will contain the initial time (originally stored in date) of one segment of 25 points.
- **g**: after the calculated prediction segments have been divided, this vector will contain the predicted activity for every segment of 25 points.
- **G**: this vector contains the ground truth value of the activity that corresponds to every 25 points segment. This information has to be introduced by the user.

The function “ground” will create these three vectors:

```
function [l, g, G] = ground(datem, a)
j=1;
k=1;
while(j<=length(datem))
    N=ceil(length(datem{j})/25);
    i=1;
    while(i<=N)
        l(k) = datem{j}((25*(i-1))+1);
        g(k) = a(j);
        i=i+1;
        k=k+1;
    end
    j=j+1;
end
dateinput=1;
prev = 1;
while(dateinput<736696)
actinput=input('activity: ');
dateinput=datenum(input('time: '));
    if(dateinput<736696)
        I1 = find((l>=dateinput),1,'first');
        I2 = find((l<=dateinput),1,'last');
        d1 = abs(dateinput-l(I1));
        d2 = abs(dateinput-l(I2));
        if(d1>d2)
            G(prev:I2-1) = actinput;
            prev=I2;
        else
            G(prev:I1-1) = actinput;
            prev=I1;
        end
    else
        M=length(l);
        G(prev:M) = actinput;
    end
end
end
```

To create the vector **l**, it divides every vector of the cell **datem** in 25 point vectors and stores the first value of every of them. At the same time, the vector **g** stores the value of the input vector **a** that contains the activity associate for every vector (which represents a segment) inside **datem**.

To create the vector **G** it is necessary that the user introduces the information. First the function asks for the first activity, the initial time is supposed to be the first component of **l**. After this, the start time of the next activity has to be introduced. The algorithm calculates the difference between this time and the two closest values of **l**. **G** will be filled with the introduced activity from the index representing the initial time, to one value bellow the index representing the closest value of **l** to the initial time of the next activity. This last value will be the initial time for the next ground truth activity. To end this function a value of time further than 1-1-2007 (736696 in serial numeric value) has to be introduced.

3.2. Score

Once the ground truth activities and the predicted activities have been obtained for the segments of 25 points, they can be compared using a simple function to compare these vectors (**g** and **G**).

The function "score" will compare the two input vectors of equal length and will tell the percentage of matched components. This percentage will be the percentage of 25 points segments whose predicted activity corresponds with the ground truth activity. As the 25 points segments are small, this result can be interpreted as how much time the predicted activity corresponds with the real activity inside the room.

```
function sco = score(g, G)
sco=0;
N=length(g);
for i=1:N
    if(g(i)==G(i))
        sco=sco+1;
    end
end
   sco=(sco/N)*100;
end
```

Finally it can be useful to analyse the precision detecting a specific activity. For a certain activity, this is the number of segments identified with this activity whose real activity is the same. Also this refers to the number of segments identified with a different activity whose real activity is also different. A function based on the last one is created to obtain these percentages:

```
function [scoTrue, scoFalse] = score1activity(g,G,a)
scoTrue=0;
scoFalse=0;
M1=0;
M2=0;
N=length(g);
M1=length(find(G==a));
M2=length(find(G~=a));

for i=1:N
    if((G(i)==a) && g(i)==a)
        scoTrue=scoTrue+1;
    end

    if((G(i)~=a) && g(i)~=a)
        scoFalse=scoFalse+1;
    end
end
    scoTrue=(scoTrue/M1)*100;
    scoFalse=(scoFalse/M2)*100;
end
```

For a given activity **a,** the number of times that it appears in both **g** (predicted activities) and **G** (ground truth activities) for the same index over the total time of occurrences in **G** will be one of the output percentages (**scoTrue**). The other (**scoFalse**) is the number of times that an activity different from **a** appears in both **g** and **G** over the total times than an activity different from **a** occurs in **G**. This last percentage is important because it tells if the activity is predicted in segments that do not belong to this activity. An activity predicted in a very big amount of segments (more than the ground truth) will have a percentage of correct predictions very high (first percentage) but the percentage of segments where it should not be predicted will be very low. It is important that both percentages have a high value in order to determine that an activity is well predicted.

4. References:

1. B. J. Finlayson-Pitts, “Reaction of NO_2 with NaCl and atmospheric implications of NOCl formation”, 676-677 (1983).

2. Michael McGee. 2016, “CO2 EARTH”. Available at: https://www.co2.earth/. [Accessed 8 May 2016].

3. “The Engineering Toolbox”. Available at: http://www.engineeringtoolbox.com/co2-comfort-level-d_1024.html [Accessed 8 May 2016].

4. A.P. Jones, “Indoor air quality and health”, 4538-4553 (1999).

5. Yunoki T, Horiuchi M, Yano T, “Kinetics of excess CO2 output during and after intensive exercise”, 139-144 (1999).

6. Ni Zhu, Tom Diethe, Massimo Camplani, Lili Tao, Alison Burrows, Niall Twomey, Dritan Kaleshi, Majid Mirmehdi, Peter Flach, and Ian Craddock, “Bridging e-Health and the Internet of Things: The SPHERE Project”, 39-46. (2015)

7. John Gale, “Sources of CO_2”, 77-101 (2005)

8. Parisi RA, Edelman NH, Santiago TV, “Central respiratory carbon dioxide chemosensitivity does not decrease during sleep” 832-836 (1992)

Appendix 1 Linear regression

Linear regression is a method to obtain an approximate model for the relationship between a dependent variable and an independent variable so that any value of the dependent variable can be expressed as:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{1}$$

Here, y is the dependent variable, x is the independent variable, β_0 is the y point where the model crosses the y axis, β_1 is the slope of the linear model and ϵ is the error.

A set of n measured values can be represented as n linear equations like $= \beta_0 + \beta_1 x$. Using matrices, n linear equations can be represented like:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \tag{2}$$

Calling: $= \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$, $X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$ and $B = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$, it can be expressed as $Y = X \cdot B$

The objective is to determine the B matrix and the way to do it is using the least-squares regression. This is implemented in matlab with the operator \ , so $B = X\backslash Y$.

To check how good the model is calculated, the coefficient of determination, R^2 , can be used. This coefficient has value between 0 and 1 depending on how well the model approximates the different measures. The higher the value of R^2, the better the model is. R^2 can be calculated using:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \hat{y})^2} \tag{3}$$

Appendix 2 Other functions

During the implementation of the SVM thresholds, some functions to identify activities, to convert from cell to vectors or to obtain the data to analyse have been used. In this section the performance of these functions will be explained.

- **slope25m**

  This function divides the segments in smaller segments of 25 points and returns a vector that contains the slope of all these new segments.

```
function [slopes25] = slope25m(datem,b)

j=1;
k=1;
while(j<=length(datem))
    N=ceil(length(datem{j})/25);
    i=1;
    while(i<=N)
        slopes25(k) = b{j}(2);
        k=k+1;
        i=i+1;
    end
  j=j+1;

end
```

It uses the variable **datem** to determine the number of 25 points segments that can be obtained from every original segment. For every new segment associates the slope of the original segment and stores it in the vector **slopes25**.

- **oneactivity**

This function returns the cell (**f**) containing the categories that will be used by the SVM algorithm. Given the number of an activity (**a**) and a vector containing multiple number of activities (**b**), it returns a cell with the category "current" at the same position of **b** where the activity was **a.** In all the other positions, the category will be "rest".

```
function f = oneactivity(a, b)
for i=1:length(b)
    if(b(i)==a)
        f{i}='current';
    else
        f{i}='rest';
    end
end
end
```

- **returnNumbers**

This function is used to transform the output cell of the SVM function containing the categories assigned to a vector where these categories have been replaced for the number of the activity. It associates the numeric value of the current activity (**a**) to the components of the vector **f** whose index represents the category “current” in the input cell (**classified**).

```
function f = returnNumbers(classified,a)
N=length(classified);
for i=1:N
    if(strcmp(classified{i},'current'))
        f(i)=a;
    else
        f(i)=6;
    end
end
end
```

Appendix 3 Sensor

A CO2 sensor is used to obtain the CO2 concentration data. This sensor is the Cozir Co2 sensor, from Gas Sensing Solutions. It has a measure range of 0-2000 ppm and can also give results of temperature and humidity. Requires a voltage supply of 3.3 V and its power is 3.5 W. Uses a serial connection to interact with the user. It receives commands and returns information.

This sensor is connected to a RaspberryPi. A RaspebberyPi is very small and economic computer with a basic hardware. In this case, it executes an adapted Debian version called Raspbian. As it executes a complete operative system, it can be accessed through an ssh connection.

The CO2 sensor is connected to the pins of the Raspberry. A basic program written in python is necessary to communicate with the sensor and to store all the measures taken.

```
  import serial
  import time

  ser = serial.Serial(
      port='/dev/ttyAMA0',
      baudrate=9600,
      parity=serial.PARITY_NONE,
      stopbits=serial.STOPBITS_ONE,
      bytesize=serial.EIGHTBITS
  )

while 1:
 outfile = open('CO2_log.txt', 'a')
 ser.write('Z\r\n')
 response=ser.readline()
 outfile.write(time.strftime("%d/%m/%y") + ' ' + time.strftime("%H:%M:%S") + ' ')
 outfile.write(response + '\r\n')
 outfile.close()
 time.sleep(10)
```

First of all, the serial connection is created (ser). After this, the program enters in a loop that never ends. The function "outfile" opens a text file called CO2_log.txt and puts the pointer at the end of the file. If this file does not exist, it creates it. Then, it sends the command "Z" to the sensor. This command makes the sensor take a measure of the CO2 concentration which is read by the program using the function "readline". This value is written in the text file along with the time (using "strftime"). After this, the text file is closed and it waits 10 seconds to open it again.

Appendix 4 Software used

Filename/Algorithm/ Package	Supplier/Source/Author/ website	Use/Modifications made/ Student written
Activities.m	Matlab	Student-written
ActivitiesSVM.m	Matlab	Student-written
ActivityDetector.m	Matlab	Student-written
ActivityDetectorSVM.m	Matlab	Student-written
adddata.m	Matlab	Student-written
butfilter.m	Matlab	Student-written
coefi.m	Matlab	Student-written
deleteActivities.m	Matlab	Student-written
divide.m	Matlab	Student-written
ground.m	Matlab	Student-written
gorundg.m	Matlab	Student-written
oneactivity.m	Matlab	Student-written
RemoveErrors.m	Matlab	Student-written
results.m	Matlab	Student-written
returnNumbers.m	Matlab	Student-written
score.m	Matlab	Student-written
score1Activity.m	Matlab	Student-written
sleep.m	Matlab	Student-written
slope25.m	Matlab	Student-written
training.m	Matlab	Student-written
unify.m	Matlab	Student-written
unifyAct.m	Matlab	Student-written
GPSLogger	Google PlayStore	Obtain position coordinates
My Maps	Google	Create a personalized map
autoCO2_scrip.py	Python	Student-written