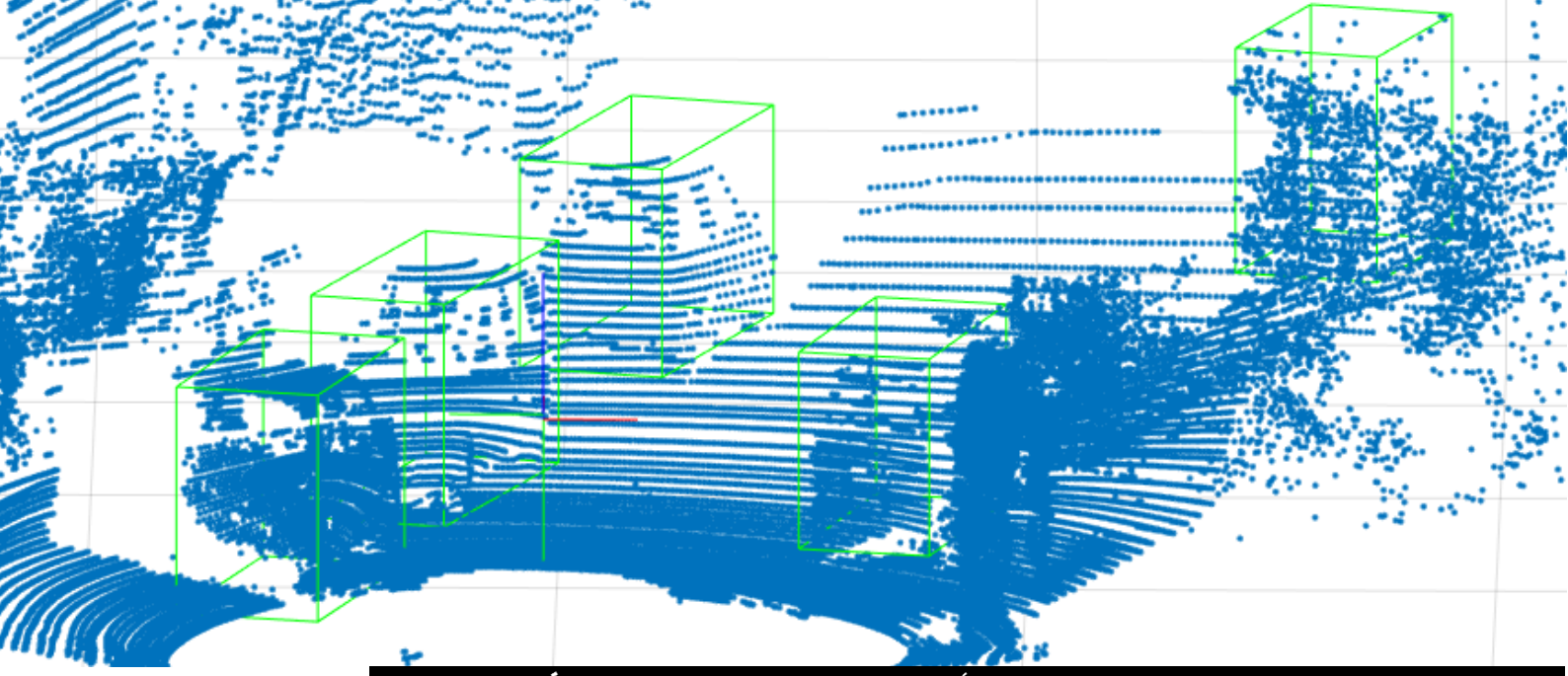


UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Template Matching Project

Project Report UN56 - P2020

Zihao Chen
Yiliang Huang
Thomas Vuillemin

Engineering and Management of Industrial Systems

Mr Zhi Yan
Researcher's Teacher

utbm
université de technologie
Belfort-Montbéliard

SUMMARY

Introduction

Project

On the UV UN56, it is asked the second half of the semester to do a project related to what is taught the first part of the semester. We then formed a 3 person group to realize this project.

The global goal

Nowadays, the AGV achieve automatic station alignment is a popular issue. With the help of the lidar, the AGV can successfully detect the environment in several sets of points around it. But it's a great challenge for the AGV to process the data so as to identify and understand its surroundings.

So, in the UN56 project, the objective is to find a solution to make the AGV be able to not only perceive the environment, but also distinguish and identify the different objects in the environment.

This project is divided in 3 groups. First of all, group 1 need to segment the point cloud detected by choosing a best-performance clustering algorithm. Then, group 2 should find a suitable method to recognize the clustering provided by group 1. And finally, group 3 will make the visualization of the two first groups' solution on Unity.

The task 2.2

The project assigned to this group is the second one. As mentioned in the previous part, our task is to find a suitable method to process the data obtained by the group 1. We need to use the existing knowledge in the database to compare with the clustering and to calculate the similarity between them. In this way, the teacher proposes us the method "template matching".

Since our project is a study for AGV, the template for us is the data of the characteristics of the object appeared in the station. We will use it to match the similar object in our environment in order to realize the function 'discrimination '.

Finally, we will also verify the object in order to prove that our programme is correct.

What is Lidar and template matching

Lidar is a method for measuring distances (ranging) by illuminating the target with laser light and measuring the reflection with a sensor. Differences in laser return times and wavelengths can then be used to make digital 3D representations of the target. It has terrestrial, air borne, and mobile applications.

Figure 1: A lidar

Figure 2: representation of a lidar environment

Template matching is a technique in digital image processing for finding small parts of an image which match a template image/model.

Combining lidar point cloud with template matching, a robot can recognize the surrounding objects, avoid the obstacle and find out its station.

Figure 3: Moving robot with lidar

Project resources

In this project, we received as a starting point a database compressed in an archive called '2.Project 2-data.zip'.There are, inside this archive, 2 folders, label and lidar.

Figure 4: The database

The text files

In folder label, there are 21 '.txt' files. All the text files are organised the same way. Each of those 21 text files is composed of values (numerical or strings). All of them represent object detection data (cluster) in KITTI label format. With the help of this website:

https://github.com/NVIDIA/DIGITS/blob/v4.0.0-rc.3/digits/extensions/data/objectDetection/README.md

From this website, we figured out those information:

All the text file contains a certain number of lines and each of those lines refers to a specific cluster and to all its description. The cluster description is composed of 16 types of value

- 1 --- The type of cluster.
- 2 --- The extent of truncated.
- 3 --- The occlusion state.
- 4 --- The observation angle of object.
- 5~8 --- The 2D bounding box of object.
- 9~11 --- The 3D object dimensions (height, width, length).
- 12~14 --- The 3D object location in camera coordinates (X, Y, Z).
- 15 --- The rotation ry around Y-axis in camera coordinates.
- 16 --- The score which represent the confidence in detection.

For our project, the values 2 to 8 and 16 will not be used since our task is to find the cluster's type thanks to values 9 to 15.

```
Truck 0.00 0 -1.57 599.41 156.40 629.75 189.25 2.85 2.63 12.34 0.47 1.49 69.44 -1.56
Car 0.00 0 1.85 387.63 181.54 423.81 203.12 1.67 1.87 3.69 -16.53 2.39 58.49 1.57
Cyclist 0.00 3 -1.65 676.60 163.95 688.98 193.93 1.86 0.60 2.02 4.59 1.32 45.84 -1.55
DontCare -1 -1 -10 503.89 169.71 590.61 190.13 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 511.35 174.96 527.81 187.45 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 532.37 176.35 542.68 185.27 -1 -1 -1 -1000 -1000 -1000 -10
DontCare -1 -1 -10 559.62 175.83 575.40 183.15 -1 -1 -1 -1000 -1000 -1000 -10
```

Figure 5: Text file number 5

The lines starting with the type "don't care" won't be analysed during this project, we will pretend that there are not here.

Since all the programming will be done on Matlab, we have to convert the category “type” of each lines on all the text file, otherwise the program will not execute. Thus, the type has to be converted from a string to a number using the lidar standard number assignment as shown below :

Class name (string in label file)	Class ID (number in database)
dontcare	0
car	1
van	2
truck	3
bus	4
pickup	5
vehicle-with-trailer	6
special-vehicle	7
person	8
person-fa	9
person?	10
people	11
cyclist	12
tram	13
person_sitting	14

Figure 6: Standard number assignment of the different types

The Excel files

In the ‘lidar’ file, there are 21 ‘.csv’ files which represent 21 different lidar point clouds. All the documents are organised the same, there is a big number of lines and all those lines are composed of 5 numbers:

- The number of the line.
- The X positions.
- The Y positions.
- The Z positions.
- The score of the numbers.

The number 2 to 5 are representing the cartesian position to a point. And this point comes from the lidar laser. All the lines then represent all the point that the lidar had detected.

For our project; the 5th column, the score number, won’t be used because too abstract.

seq	x	y	z	i
0	18.324	0.049	0.829	0
1	18.344	0.106	0.829	0
2	51.299	0.505	1.944	0
3	18.317	0.221	0.829	0
4	18.352	0.251	0.83	0.09
5	15.005	0.294	0.717	0.2
6	14.954	0.34	0.715	0.58
7	15.179	0.394	0.723	0
8	18.312	0.538	0.829	0
9	18.3	0.595	0.828	0
10	18.305	0.624	0.828	0
11	18.305	0.682	0.829	0
12	18.307	0.739	0.829	0
13	18.301	0.797	0.829	0

Figure 7: Example of a Excel document

Our method to solve the problem

After that all the data have been understood, we started to work on our project which is realizing a matching template to be able to determine the type of each cluster from the label's text files.

For the whole project, to work on the data and create the template matching, we have been using the software Matlab.

Visualization

Before to do any matching template, we had to visualize all the data we just received. Having only the numbers wasn't clear and at this point we were not able to know what we were talking about, since we only have number resources.

For the visualisation and even for the template matching, we decided we would only work on one document at the time So at the beginning of our code, we choose the name of a document and then we use this name to open the excel and text files corresponding.

The data's scene

In our first step, we have visualized the scene that the lidar had pictured. To reach that goal, we decided to extract all the lines and all the numbers of the excel file. To do so, we used the function 'dlmread' which can read a excel file and compile all the number contained in a matrix.

Afterwards, we get a matrix with all the value. The matrix has X lines and 5 columns, so the order is kept. Once all the document has been read, the function 'scatter3' is called. The function is calling 3 numbers (the X, Y and Z positions) and plot a dot at the position in the 3D word. To plot all the points, instead of importing one point, we imported all the data of each position in it. And the function returns us the plot of all the points which is the scene taken in picture by the lidar.

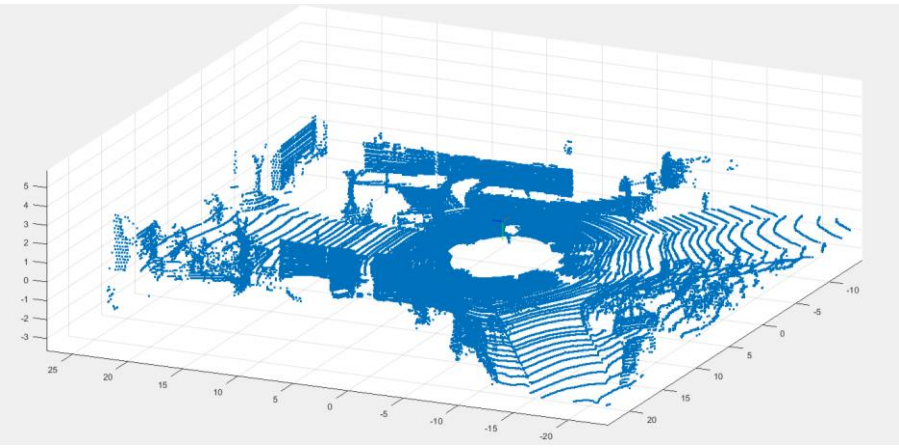


Figure 8: A lidar scene obtained on Matlab

The clusters' position

The second part of the visualization is to plot the box of each clusters of the document. The box is what we can call it "hit box", the area inside which the cluster is positioned. Then if we manage to plot the clusters' box, it will be simple to know where the cluster is located, since all the point of it would be inside the box.

Convert the value

To be able to plot the box, the necessary values are the cluster's location (X, Y and Z) and its' dimension (Height, width, length). All those values are inside the text file. So, the first step is to extract the value of the text file. To do so, we use again the function 'dlmread'. With this, all the value of each cluster is located in the same matrix.

Before to do anything with them, however, they have to be converted though. Indeed, the orientation and the position of the lidar and camera's frame are not the same, as you can see on the picture below.

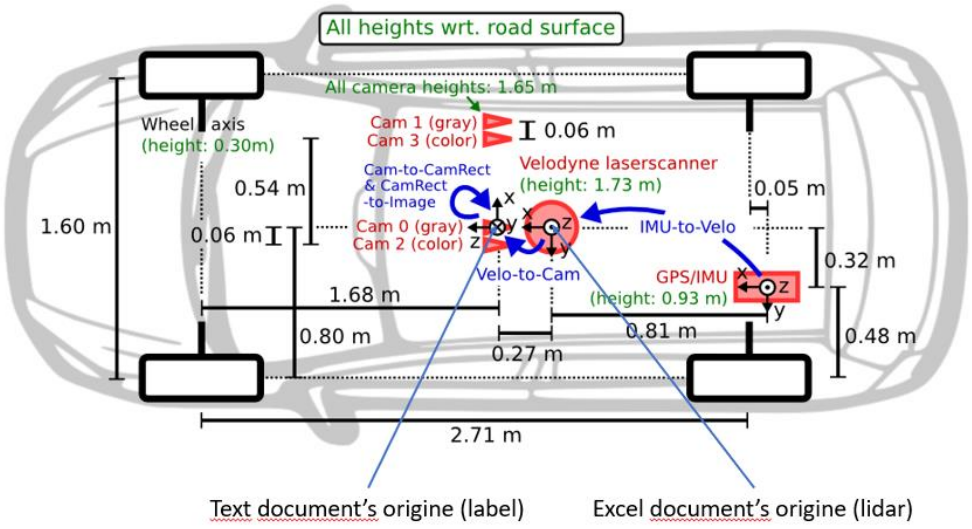


Figure 9: Cameras' position on the KITTY's car

To be able to exploit the value and then plot the box, the value of each location has to be converted first. Those value are putted in a matrix that we multiply by the transformation matrix, as learned in the UV MC55.

```
MT_cam_to_velo =

    0.0075    0.0148    0.9999    0.2729
   -1.0000    0.0007    0.0075   -0.0020
   -0.0006   -0.9999    0.0148   -0.0723
         0         0         0    1.0000
```

Figure 10: Transformation matrix

After the multiplication, the correct location of the cluster is obtained.

```
%Location from camera's frame
X = valeur(12);
Y = valeur(13);
Z = valeur(14);

Height = valeur(9);
Width  = valeur(10);
Length = valeur(11);

angle = -valeur(15);

%Location from lidar's frame
Loc = MT * [X;Y;Z;1];
```

Figure 11: Finding the correct locations and dimensions

Plot the clusters' box

Thanks to the last operation, the real location and the dimension of the cluster are known and exploitable. What we figure out is that the location given with the XYZ values represents the middle of the down face of the box.

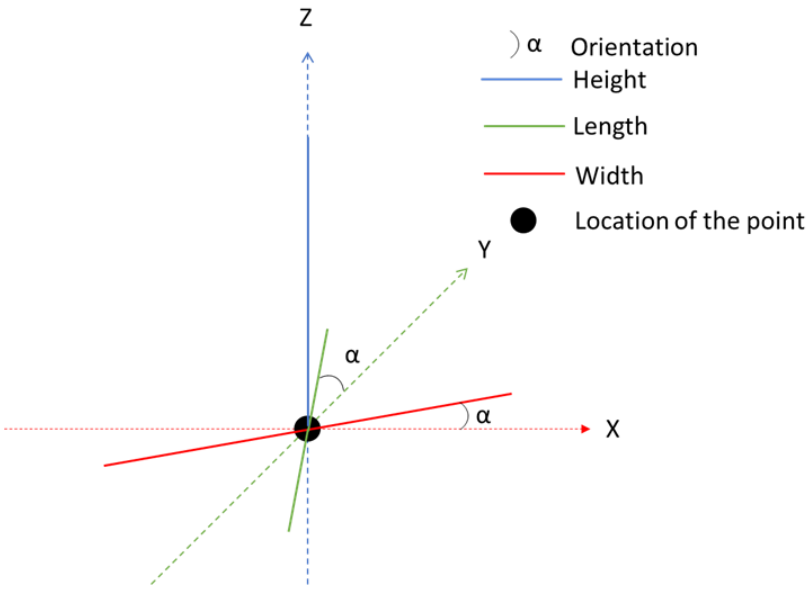


Figure 12: Representation of a point's dimension

From what we know, we have been able to find the position of the 4 corners of the down face of the box.

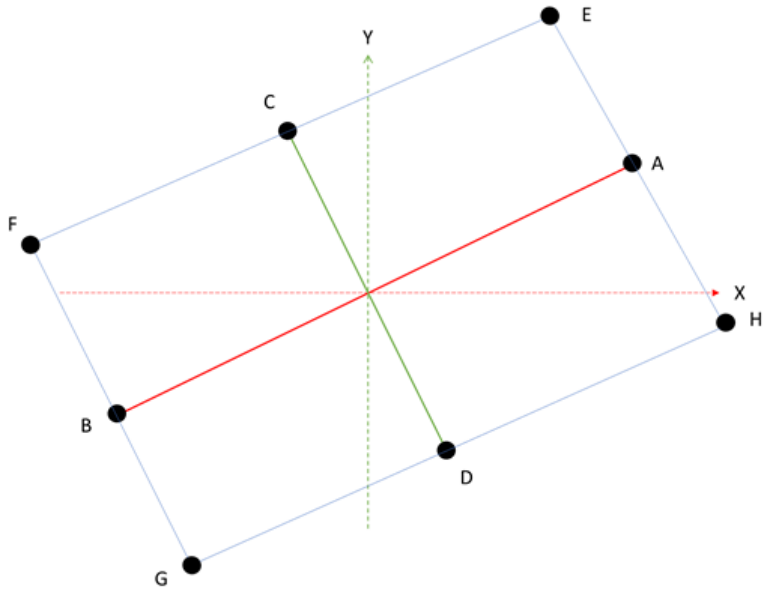


Figure 13: Representation of the 4 corners of the down face's box

With all the value we have, we calculated the position of the Point A to D and from those points, which are able to determine the 4 corners of the down face of the box, corresponding the point E to H.

```
        %X                      %Y
Pos = [Width/2*cos(angle)   ,  Width/2*sin(angle);   %PointA
       -Width/2*cos(angle)  , -Width/2*sin(angle);   %PointB
       -Length/2*sin(angle), Length/2*cos(angle);   %PointC
       Length/2*sin(angle)  ,-Length/2*cos(angle)]; %PointD

Box = [Pos(1,1)+Pos(3,1),Pos(1,2)+Pos(3,2);   %PointE
       Pos(2,1)+Pos(3,1),Pos(2,2)+Pos(3,2);   %PointF
       Pos(2,1)+Pos(4,1),Pos(2,2)+Pos(4,2);   %PointG
       Pos(1,1)+Pos(4,1),Pos(1,2)+Pos(4,2)]; %PointH
```

Figure 14: Code of the 4 corners X and Y positions finding

At that point, we have everything required to plot the box of the cluster. Thus, we use the function 'plot3' to plot the lines between the corners. the function 'plot3' requires a list of XYZ positions and it will draw a line between positions.

```
LocX = [Loc(1)+Box(1,1),Loc(1)+Box(2,1),Loc(1)+Box(3,1),Loc(1)+Box(4,1),Loc(1)+Box(1,1)];
LocY = [Loc(2)+Box(1,2),Loc(2)+Box(2,2),Loc(2)+Box(3,2),Loc(2)+Box(4,2),Loc(2)+Box(1,2)];
LocZ = [Loc(3),Loc(3),Loc(3),Loc(3),Loc(3)];
LocZ2 = [Loc(3)+Height,Loc(3)+Height,Loc(3)+Height,Loc(3)+Height,Loc(3)+Height];

plot3(LocX,LocY,LocZ,'g');
plot3(LocX,LocY,LocZ2,'g');

for j= 1:4
    plot3([LocX(j),LocX(j)],[LocY(j),LocY(j)],[Loc(3),Loc(3)+Height],'g');
end
```

Figure 15: Code of the draw of each box' lines

To draw the box, we will draw the lines of the down face with the positions calculated added to the locations. Afterwards, the same operation will be done for the top lines, only the height is changed, compare to the first plot3. And finally, the 4 vertical lines will be drawn with the For loop.

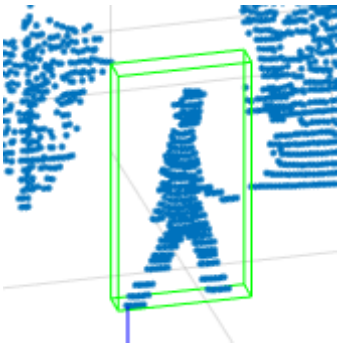


Figure 16: Example of a pedestrian box

With all that code, we are now able to draw one cluster. Since the documents often contains more than one cluster, we created a for loop in the main program which will call the program as much time as there are clusters on the document.

With this code done, we have been able to visualize all the clusters of each document.

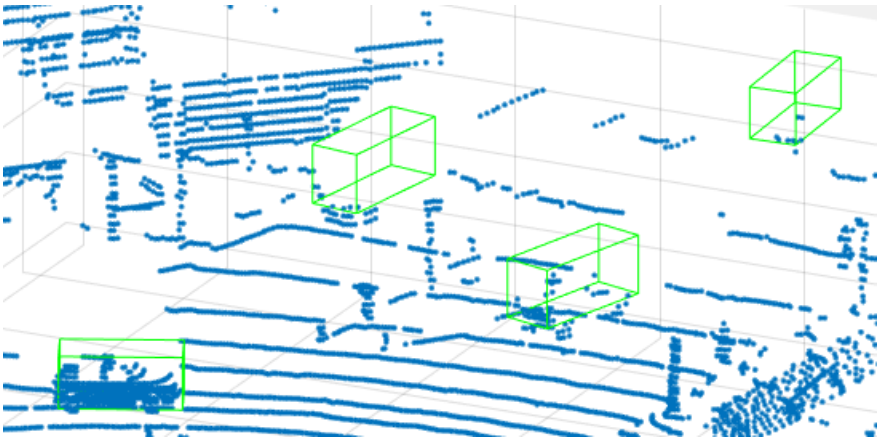


Figure 17: Result of the visualization

Template matching

With the previous part done, we are now able to clear see and identify by seeing them what is the kind of the cluster. It is now way more understandable what is going on with all those data we got at the beginning of the project.

In this part, the goal is to characterize all the types of cluster, compare the clusters from the text file to our characteristics, deduce what are their type and verify if our deduction is good or not.

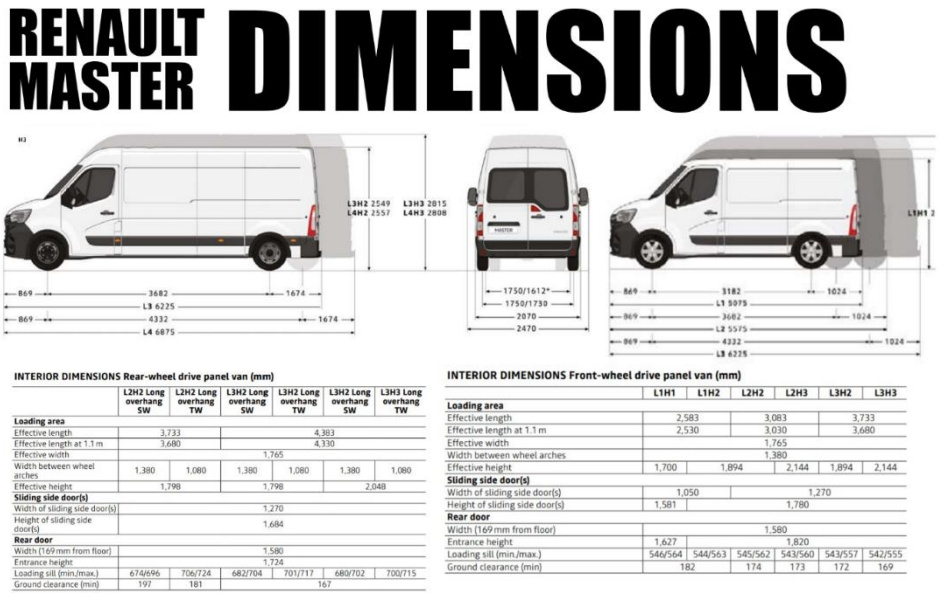
Even if the previous part, the visualization, won't help us to do the template matching, it will however help us to verify our results.

The characteristics

To realize template matching, we chose the dimension of object as the judging characteristic. We decided to look for the most common moving thing that a lidar could take in picture, which are:

- Car
- Van
- Truck
- Bus
- Pick up
- Special vehicle
- Person
- Cyclist
- Tram

For each of those type of object, we search their typical dimensions on internet (such as the manufacturer's manual or national standard), and we put them into a table.



INTERIOR DIMENSIONS Rear-wheel drive panel van (mm)

	L2H2 Long overhang SW	L2H2 Long overhang TW	L3H2 Long overhang SW	L3H2 Long overhang TW	L3H2 Long overhang SW	L3H3 Long overhang TW
Loading area						
Effective length	3,733		4,383			
Effective length at 1.1 m	3,680		4,330			
Effective width	1,765					
Width between wheel arches	1,380	1,080	1,380	1,080	1,380	1,080
Effective height	1,798		1,798		2,048	
Sliding side door(s)						
Width of sliding side door(s)	1,270					
Height of sliding side door(s)	1,684					
Rear door						
Width (169 mm from floor)	1,580					
Entrance height	1,724					
Loading sill (min./max.)	674/696	706/724	682/704	701/717	680/702	700/715
Ground clearance (min)	197	181	167			

INTERIOR DIMENSIONS Front-wheel drive panel van (mm)

	L1H1	L1H2	L2H2	L2H3	L3H2	L3H3
Loading area						
Effective length	2,583		3,083		3,733	
Effective length at 1.1 m	2,530		3,030		3,680	
Effective width	1,765					
Width between wheel arches	1,380					
Effective height	1,700	1,894		2,144	1,894	2,144
Sliding side door(s)						
Width of sliding side door(s)	1,050		1,270			
Height of sliding side door(s)	1,581	1,780				
Rear door						
Width (169 mm from floor)	1,580					
Entrance height	1,627	1,820				
Loading sill (min./max.)	546/564	544/563	545/562	543/560	543/557	542/555
Ground clearance (min)	182		174	173	172	169

Figure 18: Example of document which helped us finding the characteristics

The dimensions needed are the height, the width and the length. for 3 of them, we take the minimum value and the maximum value. For each of those type we associated the standard number that the lidar community assigned to them.

	Heigth		Width		Length		
Type	Min	Max	Min	Max	Min	Max	Number
Car	1,2	1,7	1,3	2,2	2,5	5	1
Van	2	2,8	1,8	2,5	4,5	6,9	2
Truck	2,6	4,4	2	2,9	5	17	3
Bus	3,6	4,4	2,7	2,9	14,9	18,2	4
pick-up	1,8	2	1,9	2	5,4	5,9	5
special-vehicle	3	3,6	2,3	2,8	7,6	8	7
person	1	2	0,2	1,5	0,2	1,5	8
cycliste	1,5	2	0,5	1,5	1,5	2,5	12
tram	3	3,6	2,4	2,9	12	48	13

Figure 19: Table of characteristics

The program

Now that all the cluster's types are characterized, we can move to the template matching part. The goal of this section is to identify the type of the cluster. With this program we would be able to know, with only the dimension of the cluster, what kind of cluster it is, with a degree of accuracy.

The comparison

To be able to know what is the type of each cluster, they will be compared to characteristics that we defined before.

Thus, the first step is to read and keep in memory all of those characteristics. Since they are all ordered in a excel file, we use the xlsread' to extract all the numbers of our excel document.

Afterwards, all the value is stored in a matrix and with it, we will compare them to each dimension of the cluster. It will go like this:

- Is the height of the cluster lower than the minimal height of the first type?
  - Yes, then take the value "height of the cluster divided by the minimal height of the type".
  - No, then next question.
- Is the height of the cluster higher than the maximal height of this first type?
  - Yes, then take the value "Minimal height of the type divided by the height of the cluster".
  - No, the height is between the minimal and maximal height of this type so the value is equal to one.

We repeat those 2 If statements for the width and the length to all the other types of cluster. After asking all those questions, we obtain one value for each dimension of each type of cluster. This value represents how far the dimension is to be part of the interval of the type of cluster's dimension studied.

- The value is equal to 1, the dimension is part of the interval.
- The value is lower than 1, it is not part of the interval. And as low the number is as far the dimension is to be part of this type of cluster.

3 numbers are obtained, to have a total score of the cluster compare to a specific type, we multiple all the 3 numbers together. A fourth number is obtained and with it we can know how different is the cluster from each type of cluster. We decided that from 0.8 to 1, the result is close enough to be considered as the type of cluster in question. If it is lower though, we consider that the differences are too big so the cluster cannot be part of the type of cluster researched.

With this program, it is now possible to know what is the type of the cluster researched and it is equal to the type with the highest score obtained. To know the type of all the cluster, the function is called in the main in the same For loop as the function show cluster.

```
%Show all the clusters, except 'Don't care' type
for i= 1:NbCluster
    if (valeur(i,1) ~= -1)
        fprintf('The %.0f cluster is ',i)
        typeOfCluster(valeur(i,9),valeur(i,10),valeur(i,11));
        showCluster(valeur(i,:),MT_cam_to_velo);
    end
end
```

Figure 20: For loop in the main code

If there is a case where more than one type of cluster obtains the highest score, the cluster will be assigned to the type with the highest number type (downed in the list of the cluster type).

The C# conversion

At this part of the project, the task 2.2 still needed some verification but the main structure and the main code were here. Since our project is part of a bigger one, it is important that all the tasks are able to be used all in once. To remember, the first task is to identify and find out the cluster in the excel file, the one from the lidar. The second task, the one we worked on, was to determine the class of the cluster previously found; And the last one, is to make a representation of all the result found by the two first groups.

Since all the result has to be regrouped in once and it as to be used by the last group, it is a good idea to convert all the program previously done on the language used by the last group. Since there are working on the representation on Unity, the code as to be in the language C#.

So, the program to find the type of each cluster of a text file as been converted in C#. The code can be found in the annexe part, as all the codes previously shown.

The verification

Four important indexes

In a classical binary matching model, there are 2 type of prediction: positive and negative. A prediction can be true and false according to the actual result, and so we got 4 kind of prediction result: 'True positive', 'False positive', 'True negative', 'False negative'.

The index 'accuracy', 'precision', 'recall' and 'F1' are based on them.

- Accuracy = $\frac{\text{True Positives+True Negative}}{\text{Total examples}}$
- Precision = $\frac{\text{True Positive}}{\text{True Positive+False Positive}}$
- Recall = $\frac{\text{True Positives}}{\text{True Positive+False Negative}}$
- F1 = $2\times\frac{Precision*Recall}{Precision+Recall}$

A good F1 score means that we have low false positives and low false negatives. An F1 score is considered perfect when it's 1.

How to do verification

In order to verify the accuracy of our program, we examined these indexes which are widely used in mechanic learning.

In task 2.2, to know whether the program's result is true or false, we check the result of each files manually, and we make a table for this.

		predicted								
		car	van	truck	bus	pick up	special	person	cycliste	tram
actual	Car	**true car**	false van	false truck	false bus	false pickup	false special	false person	false cycliste	false tram
	Van	false car	**true van**	false truck	false bus	false pickup	false special	false person	false cycliste	false tram
	Truck	false car	false van	**true truck**	false bus	false pickup	false special	false person	false cycliste	false tram
	Bus	false car	false van	false truck	**true bus**	false pickup	false special	false person	false cycliste	false tram
	Pick-Up	false car	false van	false truck	false bus	**true pickup**	false special	false person	false cycliste	false tram
	Special vehicle	false car	false van	false truck	false bus	false pickup	**true special**	false person	false cycliste	false tram
	person	false car	false van	false truck	false bus	false pickup	false special	**true person**	false cycliste	false tram
	cycliste	false car	false van	false truck	false bus	false pickup	false special	false person	**true cycliste**	false tram
	tram	false car	false van	false truck	false bus	false pickup	false special	false person	false cycliste	**true tram**

Figure 21: Confusion matrix

To check manually whether a prediction is 'true' or 'false', we compare the matching result and the data in folder 'label'. If an object is correctly matched, we called it 'true positive'. Otherwise, if it is matched as other types, we called it 'false positive'. In our program there is no direct prediction negative, so in the table there are only true and false. The following table is the result for each type.

		predicted									
		car	van	truck	bus	pick up	special	person	cycliste	tram	
actual	Car	43	1	0	0	0	0	0	0	0	NB clusters
	Van	0	3	0	0	0	0	0	0	0	65
	Truck	0	0	3	0	0	0	0	0	0	
	Bus	0	0	0	0	0	0	0	0	0	
	Pick-Up	0	0	0	0	0	0	0	0	0	
	Special vehicle	0	0	0	0	0	0	0	0	0	
	person	0	0	0	0	0	0	11	0	0	
	cycliste	0	0	0	0	0	0	0	2	0	
	tram	0	0	0	0	0	0	0	0	2	

Figure 22: Filled confusion matrix

With this confusion matrix, we calculated the four indexes:

	car	van	truck	bus	pick up	special	person	cycliste	tram
Accuracy	0,985	0,985	0,985	/	/	/	0,985	0,985	0,985
Precision	1	0,75	1	/	/	/	1	1	1
Recall	0,977	1	1	/	/	/	1	1	1
F1	0,989	0,857	1	/	/	/	1	1	1

Figure 23: Result obtain from the confusion matrix

For most type/class, all the indexes are close or equal to 1, which means our program has a satisfying matching accuracy.

However, our program seems not performing well with van.

3.Result

As seen on the last part, the program is able to define with a high accuracy the type of each cluster of the document analyzed. When the program is started, it will write on the windows commander the next line:

“The A cluster is a B with a score of C%”

Those 3 letters are:

- A: The number of the cluster. With this, we will be able the total number of clusters and for each of them their type.
- B: The type of the cluster. It has been identified and is return by the program.
- C: The score of obtained by the cluster. As seen previously, a score is obtained after comparing the cluster to every type. The highest score is then shown, it just has to be multiplied by 100 to have percentage.

```
The 1 cluster is a pedestrian with a score of 100.0%
The 2 cluster is a pedestrian with a score of 100.0%
The 3 cluster is a car with a score of 91.4%
The 4 cluster is a pedestrian with a score of 100.0%
The 5 cluster is a car with a score of 100.0%
The 6 cluster is a pedestrian with a score of 100.0%
```

Figure 24: The program answer

With this project and the 2 others finished, it will now be able for the lidar to search and identify the clusters in the picture it took. It will be as well able to know from what type is the cluster it previously found. And finally, it will be able to show its results with a representation on Unity.

With those 3 projects done, the AGV has now all the keys in hand to be able to understand the data it gets, locate itself according to its surrounding and find the station. Some calibration will be needed but the hardest part of the job is done.

4.Difficulties

At the beginning, we met a technical difficulty: the lidar was new things for us. We didn't know what software and how to treat the project resource (label files and lidar files).

We did spend some time on searching and exploring. Firstly, we tried to find tutorial from various forums. Since we had no idea on what is lidar, we didn't even know what was the key world for researching. Fortunately, we finally found out some useful tutorial, for example, from github we learned that the label files are in 'KITTI label format'. With these tutorials we gradually understood the project resource.

We met another difficulty when we tried to visualize and recognize objects in lidar point cloud. With the help of Matlab, we succeed visualize the cloud, and we knew the position of an object from label files. However, when we created a box to highlight the object, it was impossible to recognize what was it. The points seemed to be chaotic.

To deal with this we contacted professor Yan Zhi. Finally, we found out that there existed a problem when converting the coordinate of lidar point cloud. Mr. Yan then corrected the coordinate and sent us the files updated. As a result, we succeed the recognition of objects after spending about 2 weeks on it.

The last difficulty that we faced is that programming is not a easy task for the majority of the group. Most of us don't have solid bases on programming and/or are not really familiar with it. It took a lot of time and effort for the not programmer persons of the group to catch up with what has been done. And in addition, even if the task 2.2 hasn't been declared as a programming project by the teacher, at the end, 80% of the work has been to program the project. This proportion of programming in the project is quite big, which means that not programmer persons are quite disadvantage and behind the programmer persons. It would maybe be a good idea do explain more what expected in the project before we choose it.

5. Conclusion

We did learn a lot from this project.

Firstly, we started on lidar data treatment. We learned what is lidar, how to visualize lidar point cloud and what is KITTI label format. This gives us a global view on this new and popular technique.

Secondly, our ability of collecting and filtering information is also promoted during the project. As we talked about in the part 'difficulty', the hardest part of the project may be the start-up as we knew nothing about this before. A lot of research were done during a few weeks.

Lidar is now playing a more and more important role in our lives as the car's auto-driving technique (like Tesla's autopilot) is changing the automobile industry, and Apple puts lidar in their new iPad to enhance the function of Augmented Reality. After working on this project, we both have a clearer understanding on its potential power, which may help in the future study or work when we meet similar situation.

Table of figure

Annexes

Annexe 1: Main program

```
%TemplatMatching
clear, clc, close all;

document = '000011';

%Read the Text file
valeur = dlmread(strcat(document,'.txt'));
NbCluster = size(valeur,1);

%Difference rotation between the camera's frame and the world's frame
R_velo_to_cam = [7.533745000000e-03, -9.999714000000e-01, -6.166020000000e-04;
                 1.480249000000e-02,  7.280733000000e-04, -9.998902000000e-01;
                 9.998621000000e-01, 7.523790000000e-03, 1.480755000000e-02];
T_velo_to_cam = [-4.069766000000e-03;-7.631618000000e-02;-2.717806000000e-01];

MT_velo_to_cam = [R_velo_to_cam,T_velo_to_cam;0,0,0,1];
MT_cam_to_velo = inv(MT_velo_to_cam);

%Show all the lidar's point
PointCloud(document,MT_cam_to_velo);

%Show all the clusters, except 'Don't care' type
for i= 1:NbCluster
    if (valeur(i,1) ~= -1)
        fprintf('The %.0f cluster is ',i)
        typeOfCluster(valeur(i,9),valeur(i,10),valeur(i,11));
        showCluster(valeur(i,:),MT_cam_to_velo);
    end
end
```

Annexes 2: Point Cloud

```
function PointCloud (document,MT)

    %Read the Excel file
    value = dlmread(strcat(document,'.csv'), ',',1,0);

    %Show the points
    scatter3(value(:,2),value(:,3),value(:,4),'.');

    %Add the Wolrd's frame
    hold on
    plot3([0,1],[0,0],[0,0],'r');
    hold on
    plot3([0,0],[0,1],[0,0],'g');
    hold on
    plot3([0,0],[0,0],[0,1],'b');


    %Add the camera's frame
    hold on
    plot3([MT(1,4),MT(1,1)+MT(1,4)],[MT(2,4),MT(2,1)+MT(2,4)],[MT(3,4),MT(3,1)+MT(3,4)],'r');
    hold on
    plot3([MT(1,4),MT(1,2)+MT(1,4)],[MT(2,4),MT(2,2)+MT(2,4)],[MT(3,4),MT(3,2)+MT(3,4)],'g');
    hold on
    plot3([MT(1,4),MT(1,3)+MT(1,4)],[MT(2,4),MT(2,3)+MT(2,4)],[MT(3,4),MT(3,3)+MT(3,4)],'b');

end
```

Annexe 3: Show Cluster

```
function [Loc] = showCluster (valeur,MT)

    %Location from camera's frame
    X = valeur(12);
    Y = valeur(13);
    Z = valeur(14);

    Height = valeur(9);
    Width  = valeur(10);
    Length = valeur(11);

    angle = -valeur(15);

    %Location from lidar's frame
    Loc = MT * [X;Y;Z;1];

           %X                    %Y
    Pos = [Width/2*cos(angle)   ,  Width/2*sin(angle);  %PointA
           -Width/2*cos(angle) , -Width/2*sin(angle);  %PointB
           -Length/2*sin(angle), Length/2*cos(angle);  %PointC
           Length/2*sin(angle) ,-Length/2*cos(angle)]; %PointD

    Box = [Pos(1,1)+Pos(3,1),Pos(1,2)+Pos(3,2);  %PointE
           Pos(2,1)+Pos(3,1),Pos(2,2)+Pos(3,2);  %PointF
           Pos(2,1)+Pos(4,1),Pos(2,2)+Pos(4,2);  %PointG
           Pos(1,1)+Pos(4,1),Pos(1,2)+Pos(4,2)]; %PointH


    LocX = [Loc(1)+Box(1,1),Loc(1)+Box(2,1),Loc(1)+Box(3,1),Loc(1)+Box(4,1),Loc(1)+Box(1,1)];
    LocY = [Loc(2)+Box(1,2),Loc(2)+Box(2,2),Loc(2)+Box(3,2),Loc(2)+Box(4,2),Loc(2)+Box(1,2)];
    LocZ = [Loc(3),Loc(3),Loc(3),Loc(3),Loc(3)];
    LocZ2 = [Loc(3)+Height,Loc(3)+Height,Loc(3)+Height,Loc(3)+Height,Loc(3)+Height];

    plot3(LocX,LocY,LocZ,'g');
    plot3(LocX,LocY,LocZ2,'g');

    for j= 1:4
        plot3([LocX(j),LocX(j)],[LocY(j),LocY(j)],[Loc(3),Loc(3)+Height],'g');
    end
end
```

Annexe 4: Type of Cluster

```
function [] = typeOfCluster (Height,Width,Length)
    Dimension = [Height,Width,Length];

    clusterType = xlsread('clusterType.xlsx');

    nb = size(clusterType,1);
    result = zeros(nb,5);

    for i = 1:nb
        for j = 1:3
            if(Dimension(j) < clusterType(i,(j*2)-1))
                result(i,j) = Dimension(j)/clusterType(i,(j*2)-1);
            else
                if(Dimension(j) > clusterType(i,j*2))
                    result(i,j) = clusterType(i,j*2)/Dimension(j);
                else
                    result(i,j) = 1;
                end
            end
            result(i,5) = clusterType(i,7);
        end
    end

    maxValue = zeros(2,1);

    for i = 1:nb
        result(i,4) = result(i,1) * result(i,2) * result(i,3);
        if(result(i,4)>=maxValue(1))
            maxValue(1) = result(i,4);
            maxValue(2) = result(i,5);
        end
    end

    if(maxValue(1)<0.8)
        maxValue(2) = -1;
    end

    switch maxValue(2)
        case 1
            fprintf('a car with a score of %.1f%% \n', maxValue(1)*100)
        case 2
            fprintf('a van with a score of %.1f%% \n', maxValue(1)*100)
        case 3
            fprintf('a truck with a score of %.1f%% \n', maxValue(1)*100)
        case 4
            fprintf('a bus with a score of %.1f%% \n', maxValue(1)*100)
        case 5
            fprintf('a pick-up with a score of %.1f%% \n', maxValue(1)*100)
        case 7
            fprintf('a special-vehicle with a score of %.1f%% \n', maxValue(1)*100)
        case 8
            fprintf('a pedestrian with a score of %.1f%% \n', maxValue(1)*100)
        case 12
            fprintf('a cycliste with a score of %.1f%% \n', maxValue(1)*100)
        case 13
            fprintf('a tram with a score of %.1f%% \n', maxValue(1)*100)
        otherwise
            fprintf('undefined\n')
    end


end
```

Annexe 5: C# code

```
using System;
using System.Diagnostics;
using System.IO;

namespace TemplateMatching
{
    0 références
    class Program
    {
        0 références
        static void Main(string[] args)
        {
            string[] lines = File.ReadAllLines("textFiles/000016.txt");
            Double[,] arrayOfValue = ExtractValue(lines, 15);

            for (int i = 0; i < lines.Length; i++)
            {
                if (arrayOfValue[i, 0] != -1)
                {
                    Console.Write("The cluster " + (i + 1) + " is ");
                    TypeOfCluster(arrayOfValue[i, 8], arrayOfValue[i, 9], arrayOfValue[i, 10]);
                }
            }
        }

        1 référence
        static public void TypeOfCluster(Double height, Double width, Double length)
        {
            Double[] dimension = { height, width, length };

            string[] characteristics = File.ReadAllLines("textFiles/TypeOfCluster.txt");
            double[,] clusterCharacteristics = ExtractValue(characteristics, 7);

            Double[,] result = new Double[characteristics.Length, 5];

            for (int i = 0; i < characteristics.Length; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    if (dimension[j] < clusterCharacteristics[i, j * 2])
                    {
                        result[i, j] = dimension[j] / clusterCharacteristics[i, j * 2];
                    }
                    else if (dimension[j] > clusterCharacteristics[i, (j * 2) + 1])
                    {
                        result[i, j] = clusterCharacteristics[i, (j * 2) + 1] / dimension[j];
                    }
                    else
                    {
                        result[i, j] = 1;
                    }
                    result[i, 4] = clusterCharacteristics[i, 6];
                }
            }

            Double[] maxValue = new double[2];

            for (int i = 0; i < characteristics.Length; i++)
            {
                result[i, 3] = result[i, 0] * result[i, 1] * result[i, 2];
                if (result[i, 3] >= maxValue[0])
                {
                    maxValue[0] = result[i, 3];
                    maxValue[1] = result[i, 4];
                }
            }

            if (maxValue[0] < 0.8)
            {
                maxValue[1] = 0;
            }
```

```
            switch (maxValue[1])
            {
                case 1:
                    Console.WriteLine("a car with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 2:
                    Console.WriteLine("a van with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 3:
                    Console.WriteLine("a truck with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 4:
                    Console.WriteLine("a bus with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 5:
                    Console.WriteLine("a pick-up with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 7:
                    Console.WriteLine("a special-vehicle with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 8:
                    Console.WriteLine("a pedestrian with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 12:
                    Console.WriteLine("a cycliste with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                case 13:
                    Console.WriteLine("a tram with a score of " + Math.Round(maxValue[0] * 100, 1) + "%");
                    break;
                default:
                    Console.WriteLine("undefined");
                    break;
            }

        }

        2 références
        static public double[,] ExtractValue(string[] lines, int nbColumns)
        {
            double[,] arrayOfValue = new double[lines.Length, nbColumns];
            int i = 0, j = 0;

            foreach (String line in lines)
            {
                string[] split = line.Split(" ");
                foreach (string num in split)
                {
                    string tamp = num;
                    if (num.Contains('.'))
                    {
                        tamp = num.Replace('.', ',');
                    }
                    arrayOfValue[i, j] = System.Convert.ToDouble(tamp);
                    j++;
                }
                i++;
                j = 0;
            }

            return arrayOfValue;
        }
    }
}
```