Bachelor in Computer Science
-
Application for Pedestrian Detection with Intel RealSense Camera Technologies

Saturday 11th January, 2020

Sean Achtatou
University of Luxembourg
Email: sean.achtatou.001@student.uni.lu

Francois Robinet
University of Luxembourg
Email: francois.robinet@uni.lu

Abstract

The following Bachelor Semester Project [BSP] paper has been written by Sean Achtatou; student in Bachelor in Computer Science; and supervised by Francois Robinet from the SnT section of Kierchberg, Luxembourg. This project is a mandatory research paper, for a semester in the Bachelor in Computer Science section, headed by Prof.Dr. Nicolas Guelfi in the University of Luxembourg.
The purpose of this project is to cover the use of the Artificial Intelligence technology for computer vision, to provide an application granting the possibility to detect multiples pedestrian, and their distance, with the manipulation of a "Intel RealSense" camera - which is a depth camera allowing to record color frame(s) and depth frame(s) (i.e. RGB-D). To achieve this project, there would be the reuse of an existing application in Python, named "YOLO", application applying the Convolutional Neural Networks [CNN]; belonging to the Deep Learning (DL) algorithms; to detect objects on single frame (pictures). "YOLO" would be united with a "Intel RealSense" camera to be used on real-time frames.The final produced application, would provide a way to understand if there is the possibility to be mounted on a car, for a possible use in a real situation. Furthermore, the project should cover the favorable and unfavorable aspects of the usage of the Artificial Intelligence for computer vision with the Convolutional Neural Networks, and which possibilities it should contribute to the humankind in their daily-life.

1. Introduction

Nowadays, the vast universe of Computer Science is evolving from days to days in different domains. For several years, the Computer Science have been interested in working with the Artificial Intelligence. This intensive interest from the Computer Scientist in this domain, have occurred due to the fact to an ambition of having the ability to help humankind to ease their daily life and routine.

The universe of Artificial Intelligence is growing, and getting more and more sophisticated with the time passing over. Indeed, the current Artificial Intelligence possibilities are really broad, it can be used for a multitudes of tasks; the scalability of those would extend from implementing a robotic unit being able to listen to a speech (by using voice recognition), and answering based on the given sentences; to a simple distinction between a dog or a cat picture; using one of the throng provided algorithms in Machine Learning or Deep Learning. However, Artificial Intelligence capabilities are still far from what we mind to achieve, for the entire Artificial Intelligence algorithms, the results obtained have still an inaccuracy rate (the inaccuracy is low but does exist) which would become problematic if it is used for a critical real-life situation such as health problems or autonomous cars.

This project paper highlights the exploitation of one of the numerous Artificial Intelligence algorithm brought up to date, and belonging to the Deep Learning aspect of Artificial Intelligence, namely the Convolutional Neural Networks. The Convolutional Neural Networks is an algorithm associating the convolution of data to Neural Networks for mainly use in computer vision, thanks to which it is possible be used to detect different objects or pedestrian on a frame (pictures, images).

The purpose of this project, being the production of an application for pedestrian-detection in real-time condition, there will be a reusability of an application using the Convolutional Neural Networks, named **You Only Look Once - YOLO**. It is an application which has already been designed to be used for detection of multiples objects and person on single frames (picture/image), therefore there will be a re-implementation of the **YOLO application**, to be thereafter able to handle processing on videos and real-life cameras, and furthermore only taking into consideration people instead of objects, as we are interested in pedestrian's detection.

2. Application for Pedestrian Detection with Intel RealSense Camera

2.1. Domains

2.1.1. Scientific. The major Scientific domain of this project is the study of the *Artificial Intelligence*. For this project, there is

Fig. 1. *You Only Look Once - YOLO logo.*

a focusing on the use of one of the Artificial Intelligence field, which is the Deep Learning with the Neural Networks, for the **Convolutional Neural Networks**. We will learn how the CNN is functioning to be able to produce and detect pedestrian on images.

2.1.2. Technical. The Technical domains of this project is the study of the *Computer vision and Image processing*. Along the project, there will be the possibility to work with the YOLO Convolutional Neural Networks to detect different pedestrian on frames by working on the processing of images.

2.2. Target Deliverables

2.2.1. Scientific Deliverables. The fundamental scientific deliverable is the production of multiples .csv files, incorporating the observation of different distance calculation using the Camera RealSense associated to the YOLO application. Indeed, there would be three .csv files to observe, and refer to fixed distance between the camera and a pedestrian, respectively: *five meters*, *ten meters* and t*wenty-five meters*. For each of the .csv files, there would be the introduction to different possibilities to calculate the distance of the pedestrian from the camera, and consequently observe which technique would be the optimal one to be used. Moreover, the pedestrian would execute, random action at different distance, movements in front of the RealSense Camera, to observe for possible issues with the YOLO application performance.

2.2.2. Technical Deliverables. There are multiples technical deliverables which would be possible to benefit. The first deliverable, is the possibility to employ the YOLO application to process input *pictures*, to detect pedestrian in the image, and saving the processed image in a folder to observe the results.

The second deliverable, is to operate on the YOLO application to process on *videos*, to detect pedestrian, and saving the processed video in a folder to observe the results.

The last but not the least deliverable, being the most important one, is to use the YOLO application and the Intel RealSense Camera combined, to process the detection of the pedestrian in *live*, and show their distance from the camera.

3. Pre-Requisites

3.1. Scientific Pre-Requisites

To understand the purpose and the direction of the project application, the need to take into consideration multiples scientific pre-requisites is consequently mandatory. Indeed, multiples scientific knowledge must be satisfied in order to avoid any miss-understanding of the contents quoted further during the project – the Artificial Intelligence, the Convolution Neural Networks, and the Intel RealSense scientific aspects are being approached in this section.

Note: *There is the possibility to get a perception of the Convolutional Neural Networks processing by the following documentation and PowerPoint presentation.*

3.1.1. Artificial Intelligence. First to first, knowledge about the *Artificial Intelligence* domain, on Machine Learning and Deep Learning algorithms is necessary to understand the project - mainly the Neural Networks process. Indeed, the using of the YOLO application, is being based on the use of the Convolutional Neural Networks algorithm - belonging to the Deep Learning algorithms – and is consequently approached during all the project long. Therefore, the Artificial Intelligence knowledge must be acquired.

3.1.2. Convolutional Neural Networks. Moreover, an apprehension of the *Convolution* process on images is needed. The Convolution being based on the processing of the pixels on images, it is actually being used by the Convolutional Neural Networks - linking Convolution and Neural Networking - as a starting point into the algorithm. As explained into the section for Artificial Intelligence pre-requisite, the YOLO application, is based on the use of the Convolutional Neural Networks. Thus, mastering the knowledge on Convolution must as well be satisfied.

3.1.3. Mathematical Background. Nevertheless, some basics *mathematical background* - at Bachelor level – is needed to be understood, in order to apprehend the mathematical aspects of the Convolutional Neural Networks, and to furthermore understand the YOLO application algorithm. The major important mathematics point to focus are the Linear Algebra, the Computational Science and the Intelligent Systems.

3.1.4. Intel RealSense Camera. At least but not last, the final project application, is being performed along the technology "Intel RealSense" - being provided a camera and a depth detection. In order to be able to benefit from this technology, cognition about the processing of an image being performed on a computer must have been acquired.

3.2. Technical Pre-Requisites

To apprehend the processing and production of the project application, multiple technical pre-requisites are mandatory.

Fig. 2. *Intel RealSense Camera model*

In this section, programming knowledge, operating system, Docker and YOLO application are approached.

3.2.1. Programming Language - Python. One of the first technical pre-requisites is the usage of a programming language and an Integrated Development Environment[IDE] having to be used through the running of the project production. Indeed, the YOLO application being exploited, has been implemented on the programming language being Python – Version 3.x. Therefore, the use of Python is mandatory to be able to provide the deliverables for this project.

3.2.2. Docker. Moreover, for the purpose of the pedestrian detection using Intel RealSense application scalability and production, the reusability is one of the main important characteristic to take into consideration. The use of a virtual environment can be considerate and allowing to rapidly work on our application. Hence, the use and the introduction of Docker can be considerate.

To clarify the use of such a feature, Docker is a software allowing to launch and execute, in isolated way, applications in containers. Those containers have the capability to virtualize different kind of environment, and reuse those environment ulteriorly, employing the parameters, settings and libraries having been set in the Docker container before the running. Therefore, the use of a Docker container is highly recommended for this project to avoid unnecessary time consumption.

3.2.3. Operating System. Nevertheless, it is necessary to take into consideration the particular type of operating system employed. The Windows operating system can be considerate as a judicious choice, however the possibilities of Windows to perform with new libraries and importation capabilities can be restricted and become rapidly a source of issues. Hence, the use of an operating system based on Linux has to be considered – Ubuntu, MacOS.

3.2.4. YOLO - Application. At least but not last, to achieve the production of an application for pedestrian detection using Intel RealSense camera, we need an application, using the Convolutional Neural Networks algorithm, implemented and trained to detect pedestrian. For this purpose, the application YOLO has been chosen due to its performance and high rate of accuracy to the detection of objects and humans on images.

The YOLO application has been implemented in Python Programming language, and is based on the use of multiples libraries, to provide an efficient and functioning algorithm. The fundamental libraries used are:

- tensorflow (machine and deep learning library by Google)
- numpy (extension of the Python programming language used to manipulate and operate on matrices and multi-dimensional arrays)
- open-cv (library by the Intel company, used for the processing of images in real-time)

Note : ***The scientific and technical deliverables have been produced and ran through a Docker on Ubuntu. All the necessary settings are provided in path "./Project[Pedestrian-Detection]/finalpedestriandetection-master/finalpedestriandetection-master". To run the Docker, directly launch the "run.sh" file from a Command Prompt line on Ubuntu***

4. Scientific Deliverables

4.1. Requirements

The Scientific deliverables have to fulfill multiples requirements, which have to be verified during the development of the project. Therefore, the Scientific deliverable requirements are mandatory to be explained and defined, before any possible usage of the provided technical deliverables of the project. The purpose of this section, is essentially to be aware and mindful of the possible limits with the use of the final application deliverables.

4.1.1. YOLO - Application. The first Scientific deliverable, as explained and introduced in the targeted deliverables section, is to provide a manner to demonstrate, validate and be aware whom the YOLO application, based on the use of the Convolutional Neural Networks, is behaving and running as intended. By dint of the production of a scientific deliverable, we should be able to generate a validation method, concerning the performance of the algorithm used by the YOLO application. We would study the distinct behavior of the Convolutional Neural Networks algorithms, and any possible issues which we would be encountered with during the use of the YOLO application, and how a possible resolution of these issues would be applied to improve the YOLO application.

4.1.2. Intel RealSense Camera. As soon as the YOLO application performance would have been analyzed, we would have to focus and operate on the Intel RealSense camera performance, as it would be used along with the YOLO application to produce the multiples technical deliverables.
Nevertheless, during this phase, we would have to cover two

distinct parts.
The first part, would be regarding about the method being used by the Intel RealSense camera to record the color and depth frames. Thus, we would have to analyses if the recorded frames are represented as they should be, as the both frames would be used later with the YOLO application to detect the pedestrian and their distance.
Having introduced the notion of distance, during the second phase, we would have to analyses the different distance performance that we can obtain, operating with the Intel RealSense camera features. We would have to introduce multiples technique to calculate the distance of a pedestrian from the Intel RealSense camera, and therefore analyzing which technique would be the optimal one to be picked.

4.2. Design

As explained in the requirements, the design part is focused on two main and important parts. The first part, is an observation on the YOLO application operation performance verification, in which we are explaining the processing of the YOLO application through the use of its algorithm with samples pictures. For the second part of the design, we are focused on the use of the Intel RealSense camera, being itself divided into two others sub-parts, respectively :

- *the frames analysis*
- *the distance calculation methods*

Nevertheless, since the goal of the project was to provide an application which would be used as a pedestrian detection mounted on a car, we first have to restrict the capabilities detection of the YOLO application. Indeed, the YOLO application was originally trained to be used on multiples objects – car, bicycle, dog, etc.
Therefore, we are allowing the algorithm only to detect pedestrian on the frames, and all others objects would be discarded.

Note: *All the next listed Python application have been executed from the Console Prompt with Python IDE (i.e. python application.py)*

4.2.1. YOLO - Performance. We had to be interested to the YOLO application performance. Observing if the model, produced by the Convolutional Neural Networks, and being used by the YOLO application, would be sufficiently accurate and could be further used with the Intel RealSense camera.

We had to implement a program in Python - "Processing_Image.py" – to process YOLO on images.
As said, the program is taking as input the path of a *.jpg/.png* image, which is passed into the YOLO application model. We would obtain an output image, displaying the detected pedestrians surrounded by a green rectangle, and being saved to the path *./data/demo_data/results/Result.jpg*. For this experience, we used the image *photo.jpg*.

Fig. 3. *photo.jpg*

Fig. 4. *photo.jpg passed through "Processing_Image.py" application*

4.2.2. Intel RealSense Camera - Recording. We had to get interested into the Intel RealSense camera capabilities to

record the frames of the environment, to know if it could be used with the YOLO application.
Based on the documentation of the product, the Intel RealSense camera have the possibility to record two frames at the same time. The first frame is the color frame of the environment being recorded, whereas the second frame is obtained by a laser, to calculate and produce a depth frame representing the distance of each pixel from the camera of the frame.

Fig. 5. *Intel RealSense Camera - Color Frame*

Fig. 6. *Intel RealSense Camera - Depth Frame*

4.2.3. Intel RealSense Camera w. YOLO Application – Distance. Once the Intel RealSense camera capabilities of recording the frames were satisfied, we would start to take into consideration the calculation of the distance of a pedestrian from the camera using the frames of the Intel RealSense camera. We had to work on the implementation of three Python files, respectively:

- *Application_Frames_Recording.py*
- *Application_Frames_Processing.py*
- *Application_Video_Creation.py*

The *Application_Frames_Recording.py* Python file, would record the frames (color frame, depth frame) obtained by the Intel RealSense Camera, and save these into folders to path *./data/DataRealCameraSense/Recording/InputFrameFolder*. Moreover, the application would display the color and depth frame, to comprehend the frames being recorded.

Note: *The application is requesting an input distance (5,10,25) to save the frames for diverse distance, to the corresponding folder.*

The Python file *Application_Frames_Processing.py*, would take as input the frames saved previously, and process separately both of these (color frame, depth frame). For each color frame, in accordance to its depth frame (must not be disparate), the color frame would be processed into the YOLO application model, to detect a possible pedestrian. And display, on an output frame, the detected pedestrian surrounded by a green rectangle.
Whereas, the depth frame would be used to calculate the distance of the detected pedestrians on the output frame. It would be cropping, based on the position and dimension of the green rectangles existing in the output frame, the depth frame accordingly. Each resulting frames would be saved into folders to path *./data/DataRealCameraSenseRecording/OutputYOLOProcess*

We had to validate the performance of the algorithm of the YOLO application model by applying it to different distance from the camera; reminder the purpose of this project is to observe if the final application could be used on a car. Applying the association of the YOLO application and the Intel RealSense camera to different distance - five, ten and twenty-five meters - is mandatory. Moreover, we additionally provided multiples frames in diverse position and poses of a pedestrian in front of the camera – front, side, ball, star – to observe any possible issues with the YOLO application during the use of the Intel RealSense Camera.

Fig. 7. *Result frame for 5m*

Note: *The application is requesting an input distance (5,10,25) in accordance to which saved folder must be processed*

The Python file *Application_Video_Creation.py*, would process the grouping of the entire resulted frame saved previously.

Fig. 8. *Result frame for 10m*

Fig. 9. *Result frame for 25m*

The final deliverable would be the production of viewable video displaying the detected pedestrian accompany with their distance.

4.2.4. Intel RealSense Camera w. YOLO Application – Quantile. We have discussed concerning the distance from the Intel RealSense Camera, however did not focused on the calculation of the distance based on the obtained Intel RealSense frames – color frame and depth frame. To be used on a car, the final project application should be capable to calculate and predict the distance, of the detected pedestrian(s), from the Intel RealSense Camera at any distance.

We had to use two techniques - reminder a picture is a set of pixels. With the depth frame, obtained after the execution of *Application_Frames_Recording.py*, we first had to crop the depth frame based, on the position and dimension, of the detected pedestrian green rectangle on the color frame.

Once the depth frame was cropped, we had used primitively the average of the cropped depth frame pixels' value. Later on, we discovered the calculation of the distance of a pedestrian would be improved based on the introduction of the quantile method.

The quantile is a method taking a percentage of as set of values, and returning only the ones situated under a given threshold. As illustration, assuming we would like only to retrieve thirty percent of the smallest value of an array a, then,

$$b = quantile(a, 0.3) \tag{1}$$

would return the average of the thirty percent of the smallest value, and,

$$a[a < b] \tag{2}$$

would return the thirty percent of the smallest values of the set a.

Note: *We can test the quantile method with different percentage and small values by executing the application "Test_Quantile.py" in path "./YOLOv3_TensorFlow-master"*

Thus, for each depth frame, we did use the quantile method with multiples threshold – from ten to ninety percent – on different pedestrian(s) position, to calculate the distance of the pedestrian(s) from the Intel RealSense Camera.

The resulted frame obtained for each distance were being saved in a .csv file in path *./data/Data_RealCameraSense_Recording/Output_YOLO_Process* with info :

- *frame number*
- *detection box (rectangle) position*
- *quantile for ten percents*
- *quantile for twenty percents*
- ...
- *quantile for ninety percents*

4.3. Production

The production section is about the observation and analysis of all the collected data obtained from the design section. Each sub-section of the design is being details about the information obtained from the YOLO application and the Intel RealSense camera.

4.3.1. YOLO - Analysis. As we can observe, by the resulted frame obtained from the YOLO application, we would assume, that the YOLO application performance to detect pedestrian(s) is highly accurate and would be used directly with the Intel RealSense camera. Indeed, from the output frame obtained, we can see that YOLO is actually recognizing a person into the picture and drawing a red rectangle around it. Moreover, the YOLO application contain a feature allowing to predict, with a probability, the detection of a pedestrian and displaying it on the rectangle.
However, we have not only worked on one frame, but have observed that there exist multiples use cases - which are discussed in more details during the *Intel RealSense Camera – Distance – Analysis* - in which the algorithm is not performing great on the frames.
Nevertheless, in the case we are working with only a single picture, in which the pedestrian is well positioned on the

frame, and looking at the camera, the performance of the YOLO application is highly accurate.
We would have got interested into every steps process of the YOLO application algorithm, using the Convolutional Neural Networks, by analyzing the architecture. However, the purpose of the project being the creation of an application combining the YOLO application and an Intel RealSense camera for pedestrian(s) detection, we did not invest time into analyzing the YOLO architecture.

4.3.2. Intel RealSense Camera - Analysis. After having recorded multiples samples frames with the Intel RealSense camera, we can observe that the obtained frames are appropriate. This can be verified by displaying the actual pixels contained into both of the frames and analyzing each of the values they contain. Moreover, by superposing both of the frames together, we can see that they are perfectly concordant to each other, meaning that the capture of the frames has been correctly done.

Nevertheless, we know how the composition of a color frame is done; three color channels (Red, Green, Blue) with pixels going from zero to two-hundred and fifty-five. However, we have to details the depth frame composition, to be sure about the information it contains, and if it can be used to calculate a distance of a pedestrian from the Intel RealSense camera.

As we can observe, the depth frame is represented into grayscale, using the three RGB channels, however for each pixel, the same values on each color channels are used (i.e. 18,18,18). The dark part of the frame (three channels pixels with values approaching zero) is representing the pedestrian(s), object(s) or environment being near from the Intel RealSense camera. And, the light part of the frame (three channels pixels with values approaching two-hundred and fifty-five) the more far it is from the Intel RealSense camera.
Nevertheless, we can see that the sky is represented in black into the depth frame, would it mean the sky is near the camera? Hopefully not, this situation can be explained by the fact that the Intel RealSense laser cannot provide a distance since the sky distance is infinitely too far and not calculable by the laser.

4.3.3. Intel RealSense Camera w. YOLO Application (Distance) – Analysis. We are analyzing multiples frames, obtained by the Intel RealSense Camera and going through the YOLO application, at different distance to observe the performance of their association – we are naming it Y-I application.
The purpose of analyzing their performance at different distance, is due to the fact that we have to keep it mind the final application has to be used on a car to detect pedestrian(s), and brake, based on this detection. Indeed, if we assume a pedestrian can only be detected by the Intel RealSense at a specific distance, and at a certain car speed, we can easily calculate the time the car would have to react at braking, before

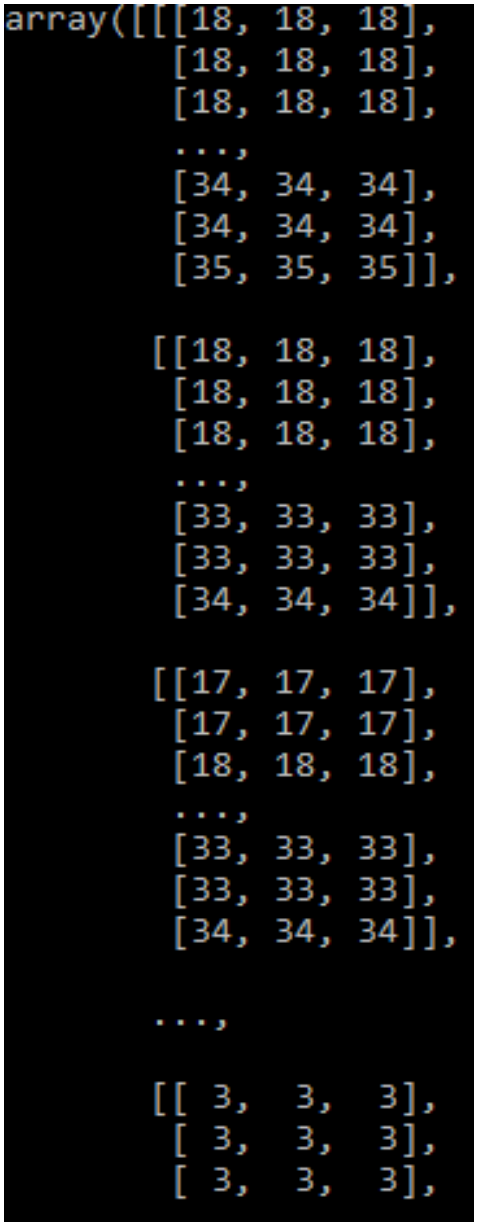

Fig. 10. *Display of each pixels for the Depth Frame*

hitting the pedestrian(s). Assuming that the car is going at the speed of thirty kilometers per hour.

$$\frac{5m}{30_{KM/H}/3,6} = 0,60 secondes \tag{3}$$

$$\frac{10m}{30_{KM/H}/3,6} = 1,20 secondes \tag{4}$$

$$\frac{25m}{30_{KM/H}/3,6} = 3 secondes \tag{5}$$

However, we calculated only the time that the car has to react before hitting the brake. If we want to avoid hitting the

pedestrian, the car should be fully stopped. A deceleration typical of an average car is about the square of six meters per second. Thus, with this information we can calculate the time the car would actually have to take to fully stop.

$$\frac{30_{KM/H}/3,6}{6} = 1,38 secondes \quad (6)$$

Consequently, we need to be sure the Y-I application is capable to detect pedestrian(s) from at least fifteen meters.

We are starting with a distance of five meters from the Intel RealSense camera, in which we have taken multiples frames of a pedestrian into different position – standing, knee, ball and side.

Fig. 11. *Frame with standing pedestrian at 5m*

Fig. 12. *Frame with knee pedestrian at 5m*

We can observe that the Y-I application is being performing when the pedestrian is standing, on knee, or on the side. However, when the pedestrian is positioned as a ball, the Y-I application is not detecting it at all. This issue of detection is due to a lack of training for the model of the Yolo application, it means it has been trained for pedestrian which are showing well their head and body, however when one of these parts are missing, the detection is beginning to be non-effective.

We are now focusing on the distance of ten meters, in which we have as well taken multiples frames of a pedestrian with different position – standing, knee, ball and side.

Fig. 13. *Frame with side pedestrian at 5m*

Fig. 14. *Frame with ball pedestrian at 5m*

We can observe as well the Y-I application is being effective when the pedestrian is standing, on knee, as a star or on the side. However, we are having the identical issue then related before, when the pedestrian is positioned as a ball, the Y-I application is not detecting it at all.

We are finally focusing on the distance of twenty-five meters, having taken as well multiples frames with different position – standing, knee, ball and side.

We can observe that, as usual, the Y-I application is being effective when the pedestrian is standing, on knee, as a star or on the side. However, once more, when the pedestrian is positioned as a ball, the Y-I application is not detecting it at all.

4.3.4. Intel RealSense Camera w. YOLO Application (Quantile) – Analysis. As explained in the design section, we have to calculate, at least to predict, the distance of the pedestrian(s) from the Intel RealSense camera. This is mandatory since, in the case the Intel RealSense camera is being mounted on a car, the car should brake or avoid the pedestrian based on its distance from the car.

To calculate the distance of a pedestrian from the Intel RealSense camera, we are thinking about calculating the average values of the pixels contained into the detected box of the pedestrian. However, as we can observe, since the box is not countering the pedestrian itself, but displaying globally its position on the frame, a lot of noisy pixels are

Fig. 15. *Frame with standing pedestrian at 10m*

Fig. 16. *Frame with knee pedestrian at 10m*

Fig. 17. *Frame with side pedestrian at 10m*

Fig. 18. *Frame with ball pedestrian at 10m*

present, particularly the background pixels. Indeed, by using the average of the pixels, the background pixels, which are far away from the camera, will be in majority in the frame compared to the pixels of the actual pedestrian, and consequently provide a false distance of the pedestrian when we are calculating the average.

To solve this issue, we are using a quantile method. The quantile method is taking, from as set of pixels, the percentage of the pixels having the minimum values. In our frame, the pixels having the minimum values are belonging to the objects near to the Intel RealSense camera. Consequently, we don't have to deal with the background pixels anymore, and can calculate the average of the actual pixels belonging to the pedestrian. Therefore, the distance from the Intel RealSense camera obtained is partially correctly calculated.

Nevertheless, we don't know which percentage of pixels must be taken from the detected box to obtain the finest distance of the pedestrian from the Intel RealSense camera compared to the real distance. However, we can figure it out by calculating the distance using different quantile for ten to ninety percent of the pixels and observe the obtained results. Moreover, we are calculating the distance of the pedestrian using the different quantile at different distance - five, ten, twenty-five meters - and using the different position used during "Intel RealSense Camera – YOLO Application - Distance – Analysis".

For every frame, we are taking the quantile which is leading to the distance having value nearest to the real distance the pedestrian is on the frame. As we can observe into the obtained .csv files, each frame is leading to different quantile, therefore we must take the average of the obtained quantile since the distance must be predictable for each situation. We can observe that for five meters, the ideal quantile is "0.7", for ten meters the ideal quantile is "0.5", and for twenty-five meters the ideal quantile is "0.3".

We can observe that the quantile percentage for each individual distance are being very diverse. However, the distance of the pedestrian must be as well predictable for each distance, thus we are taking the average of the quantiles, offering us as final quantile to be used for the distance calculation of "0.5".

Note: *The observation of the quantiles results can be retrieved in the annex "Performance Yolo + IntelRealSense.pdf" file.*

4.4. Assessment

As we have observed by the result evaluated during the production, the YOLO application, and the Intel RealSense camera, are fulfilled to be combined and operated for

Fig. 19. *Frame with standing pedestrian at 25m*

Fig. 21. *Frame with side pedestrian at 25m*

Fig. 20. *Frame with knee pedestrian at 25m*

Fig. 22. *Frame with ball pedestrian at 25m*

pedestrian detection, under conditions.

Indeed, we have observed that the fundamental problem with the YOLO application is the lack of training of the model, using the Convolutional Neural Networks, on the pedestrian. This is consequently leading to a miss-detection of the pedestrian on the frame, and therefore this issue has to be considered during the final application creation. By calculating the number of recorded frames, based on the number of frames having detected the pedestrian in the .csv file, for each distance (i.e. 5m, 10m, 25m), we get a performance of detection for the YOLO application between ninety-six to ninety-nine percent.

$$\frac{1501}{1562} = 0.96 \tag{7}$$

$$\frac{1770}{1815} = 0.97 \tag{8}$$

$$\frac{3350}{3357} = 0.99 \tag{9}$$

Nevertheless, the YOLO application performance is adequately correct and satisfying to be employed, despite the fact it does not satisfy all the requirements.

Moreover, we have seen that the Intel RealSense camera is having issues on the depth frame. Indeed, the depth frame obtained is having difficulties to establish a distance pixel with environment being too far from the Intel RealSense camera (i.e. sky) on the frame. However, the detection by the YOLO application being focused on the pedestrian itself; and furthermore using a quantile; the distance calculation is always focused on the actual pedestrian and not on the background. Therefore, the problem is being solved. Consequently, the Intel RealSense Camera has satisfied our requirements.

5. Technical Deliverables

5.1. Requirements

After the verification of the performance for the YOLO application and the Intel RealSense Camera, we are prepared to provide technical deliverables. In this section, we are enunciating the multiples deliverables which are determined to be produced, by the association of the YOLO application and the Intel RealSense camera. The final application would be a combination of multiples sub-applications, these possible to be employed in order to predict pedestrians on various supports: *image*, *video frames* and *live frames*.

5.1.1. Image Processing. The first technical deliverable would be the possibility to perform *Image processing*. The application should be able to be used, in order to detect pedestrian, on an image inputted. The application should be elementary to the usage, produce an image representing the position of the

pedestrian on the image with a green rectangle and displaying it is a pedestrian which has been detected, and save the results into a folder.

5.1.2. Video Processing. Our second technical deliverable is the possibility to perform *Video processing*. The application, such as *Image processing*, should be capable to be used, in order to detect possible pedestrian, although, this time, on an inputted video. The application, should likewise be simple to be used, produce a video representing the position of the pedestrian on the video, and save the results in a folder.

5.1.3. Live Processing. The last, but not least, technical deliverable, is the most important one. Considering that, the association of the YOLO application and the Intel RealSense camera, must be able to be used on a car, the processing of the frames must be possible to be achieved on live.

Consequently, a *Live video processing* application would be capable to process, by using the YOLO application in live, the frame of the Intel RealSense camera, to detect pedestrian(s). Moreover, the application would calculate and display the distance of the pedestrian on the frame, therefore the car would behave accordingly based on the distance obtained from the pedestrian(s).

5.2. Design

As introduced in the requirements section of the technical deliverables, we are focused on the production of three distinct deliverables which would be usable based on the use of the YOLO application and the Intel RealSense Camera. Among those deliverables, we have the production of an application implemented to detect pedestrian(s) on images. We have the production of an application implemented to detect pedestrian(s) in a video. And the most interesting one, being the objective of this project - we would know if it is possible to use the association of the YOLO application and the Intel RealSense Camera on a car - is the production of an application implemented to detect pedestrian(s) and display their distance from the camera in live.

Note: *All the listed Python application have been executed from the Console Prompt with Python IDE (i.e. python application.py **argument**)*

5.2.1. Image Processing Application. The image processing application has already been introduced during the scientific deliverable to analyses the performance of the YOLO application. However, it is being used as well in the technical deliverable, due to the fact it is using the YOLO application has benefit and produce satisfactory results.

We had to implement a program in Python - *Processing_Image.py* – to process the YOLO application on images.

As said, the program is taking as input the path of a *.jpg/.png* image, which is passed into the YOLO application model. We would obtain an output image, displaying the detected pedestrian(s) surrounded by a green rectangle, and being saved to the path *./data/demo_data/results/Result.jpg*.

5.2.2. Video Processing Application. For the video processing application, the operation is slightly different than for the image processing application. Indeed, the application would take this time, as input, a video of type *.mp4/.avi* and read the frame of the video one by one.

For each of these frame being read, they would be injected in the YOLO application model, which would detect possible pedestrian(s) on the frames, and surround each of the pedestrian(s) with a green rectangle.

Moreover, for each of the frames having been processed by the YOLO application, they would be regrouped in a new video, that would be viewable after having processed all the frame of the original video, and would be saved in the path *./YOLOv3_TensorFlow-master*.

5.2.3. Live Processing Application. Finally, the most important deliverable is the live video processing application. As remember, this application would be using the association of the YOLO application and the Intel RealSense Camera to detect pedestrian and display their distance on the frame(s) in live.

For the application to be running, it would require a wired USB connection of a camera directly connected to the computer. The application would research for the existence of a possible depth camera (i.e. Intel RealSense Camera) connected, which has the possibility to produce a color frame and a depth frame (if there is no camera recognized by the application which such features, the application will close).

Once a depth camera would have been recognized, the application would be start to be running and displaying the final frame in a window on the screen.

Note: *the application can be stopped at any time by pressing the "q" key. The speed of execution of the application per frame is displayed on the top left corner of the window screen in green. Unfortunately, the speed of execution is highly dependent to the CPU/GPU power of the computer. For us, since our computer power was limited, it would take two seconds per frame to be processed*

The final frame is obtained by following multiples steps.
At first, the application would wait to obtain a set of frames (color frame, depth frame) from the connected depth camera.

If the application would successfully have obtained a set of frame, it would be using as starting point the color frame, that would be injected in the YOLO application model to detect any possible pedestrian(s) on the frame, and display a surrounding green rectangle around any pedestrian detected.

If this detection of the pedestrian on the color frame has been succeeded, thus the depth frame would be used to calculate the distance of the detected pedestrian(s) on the color frame. For every green rectangle surrounding the detected pedestrian(s), the depth frame would each time, be cropped based on the position and dimension of the green rectangle, and the quantile method would be used to obtain the predicted distance of each pedestrian. The predicted distance would be displayed on the bottom left, for each of the pedestrian, on their corresponding detected green rectangle.

Lastly, we would have the possibility to combine every frame obtained, in a video, saved in the path *./YOLOv3_TensorFlow-master*, and would be viewable .

5.3. Production

The production of the technical deliverables is about the observation, explanation and awareness of the data obtained by processing the different implemented application – images, videos and live. We would explain which category of information we should obtain by using those deliverables applications.

5.3.1. Image Processing Application - Performance. For the image processing application, we are using two different images to be injected into the application. The first image (photo2.jpg) is a pedestrian having been taken in photo directly in front of the camera, whereas the second image (photo3.jpg) is a pedestrian doing a pose leaned against a window.

Fig. 23. *photo2.jpg*

Fig. 24. *photo2.jpg passed through "Processing_Image.py" application*

Fig. 25. *photo3.jpg*

As we can observe with the *photo2.jpg*, we are obtaining an image which is displaying perfectly the position of the pedestrian surrounded with a green rectangle on the frame, associated to the type of object having been detected at the top - *person*. Moreover, we are obtaining the probability of detection of the pedestrian, being displayed at the top of the green rectangle surrounding the pedestrian(s) in the frame. Allowing us to observe the performance capability of the YOLO application to detect with a certain percentage of probability the type pedestrian.

Nonetheless, when injecting the *photo3.jpg* into the application, we are obtaining a special case. Indeed, we can observe, that we are having a detection of a pedestrian in two distinct positions on the frame.
The first detection obtained, is based on the actual pedestrian posing in front of the camera. However, the application has managed to detect a second pedestrian – which is actually the same one – into the reflection of the window. Therefore, we

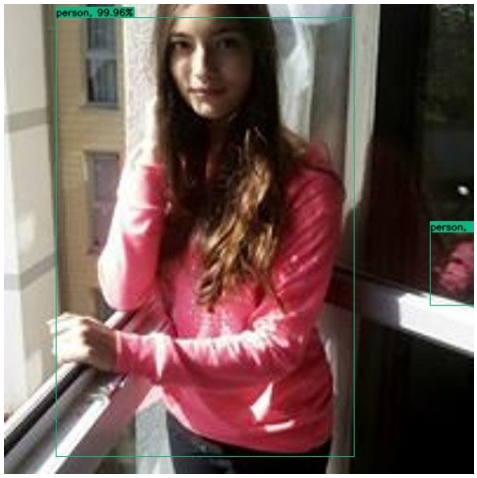

Fig. 26. *photo3.jpg passed through "Processing_Image.py" application*

can affirm that the application performance is highly efficient to detect pedestrian in special cases, even when they are not directly in sight.
Nevertheless, this type of detection could not be always helpful, in the case a car would be faced with a mirror, how would it react?

5.3.2. Video Processing Application - Performance. For the video processing application, the process is barely more complex. We are using as sample, a video having recorded multiples pedestrian(s) dancing and moving in front of a camera – *video.mp4*.
As we can observe, the application is being performing to detect multiples pedestrian at the same time, and displaying for each of them their precise position on the frame, by surrounding them with a green rectangle. Nevertheless, by going through the flow of the video, we can observe that some pedestrians are not being detected when they are placed into specific position. Indeed, when the pedestrians are laying on the floor, the application is beginning to have difficulties to detect them – by looking at the probabilities of the detection rapidly decreasing –, the detection starts flickering, nay does not detect the pedestrians at all.

5.3.3. Live Processing Application - Performance. For the last, but not the least, we are observing the process of the live video processing application. Unfortunately, for this case, the creation of a video recording the live was optional. Due to this constraint, the following observation have been deduced directly during the execution of the application.
Nevertheless, as first observation, we can see that the application is being efficiently performing to detect the pedestrian(s) in front of the camera, such as for the video processing application. However, we should keep in mind, based on the observation of the scientific section, that the performance of the application is being variable based on the position of the pedestrian.

As second observation, we can furthermore see that the distance of the pedestrian(s) - which has been calculated using the quantile method – predicted and displayed into their respective detected green rectangle, are close to the real distance measured from the camera to the pedestrian(s).

Nonetheless, as explained in the design, since our computer power is limited, the live processing of the frames is heavily slow (two seconds per frame). Leading to an application being badly reactive to a possible pedestrian detection. However, this issue is rapidly solvable by the introduction and use of a superior computer having access to an improved CPU/GPU.

5.4. Assessment

As we have observed during the observation of the production of our technical deliverables, the use of the YOLO application, or its association with the Intel RealSense Camera can gather interesting data.

Indeed, we have observed the application *Processing_Image.py*, based on the use of the YOLO application, is producing fascinating results. We have seen that the application has managed to recognize and detect a pedestrian over the reflection of itself on a window.
However, this situation would become quite critical, assuming that the application would be used on a car to detect pedestrian(s). Surely, if a car would find itself in front of a window, reflecting multiples pedestrian(s), being actually situated on the side of the road, the car would likely brake to avoid the un-existing pedestrian(s), and end up being stuck on the road.
Nevertheless, considering that the application which have to be mounted on a car, is the *IntelRealSense-Cam_Yolo_Application.py*. As the application has to detect the pedestrian(s) on live, this situation would be solved by using the depth frame (explained further down)

With the application *Processing_Video.py*, based on the usage of the YOLO application as well, we have observed that the application is capable to detect multiples pedestrian(s) synchronously on a same frame. Indeed, the pedestrian(s) in the streets are infrequently walking lonely, many of them are present on the roads. Consequently, the capabilities of the application to detect multiples pedestrians is mandatory, to be used on a car.
However, having already observed it in the scientific section, the detection capabilities of the YOLO application are limited, and are reverberated on our application. Indeed, we have observed, the pedestrian(s) in a specific position (i.e. laying, on the side) are being hard to be detected, nay not totally detected. This issue can be explained by a limited training on the model of the Convolutional Neural Networks used by the YOLO application. To solve this issue, a bigger dataset of pedestrian(s) (more pedestrian(s) laying on the floor) should have been introduced and trained into the Convolutional

Neural Networks.

Finally, for the *IntelRealSenseCam_Yolo_Application.py* application, the deliverable regroups the capability of the YOLO application to detect pedestrian(s), expanded to the calculation of the distance of the detected pedestrian(s) from the camera with the Intel RealSense technology. The capability of detection of the pedestrian(s) by YOLO has yet been introduced by the two last deliverables applications, therefore we would focus on the quantile.
We have observed that the quantile, which has been chosen during the scientific section, have been confirmed to be the optimal one. Indeed, we have seen the distance of the pedestrian(s) predicted, and displayed on the frame, were being close to the reality. Therefore, we are confident that the distance of the pedestrian(s) displayed on the frame(s) would consistently be legitimate.
Moreover, this application is solving an issue having been observed during the first deliverable application. Indeed, we have seen that *Process_Image.py* is capable to detect pedestrian(s) on the reflection of a window/mirror, and would be an issue if mounted on a car. Nonetheless, with *IntelRealSenseCam_Yolo_Application.py*, due to the usage of a depth frame, our application would be capable to predict the real distance of the pedestrian(s) (i.e. the distance calculated would be the distance of the car from the window/mirror and not the actual pedestrian(s), being given that the surface of a window/mirror is equal, consequently the application would be aware it is in presence of a reflection).
Nevertheless, every deliverables application produced has fulfilled our technical requirements, and would consequently be usable on a car.

6. Conclusion

We have gathered a volume of fascinating data of the capabilities and performance from the YOLO application. We have seen the advantages and inconvenient of using the Convolutional Neural Networks to predict the detection of pedestrian(s) on frame(s), and explained which would be the possible solutions to adopt in order to get rid of the issues encountered to produce an improved performance of the application.

Likewise, we have seen the Intel RealSense Camera functionalities and features (i.e. depth frame) are interesting to be introduced, by dint of we managed to be able to predict the distance of the pedestrian(s) on the frame(s) with a precision of the distance approaching the reality.

The objective of this project was to observe if it would have been possible to associated both of these technologies together – *YOLO and Intel RealSense Camera* -, and being usable on live mounted on a car to detect pedestrian(s) and their distance on the roads.
After a numerous of observations, our final application *IntelRealSenseCam_Yolo_Application.py*, associating both of these technologies, fulfilled our objective requirements, under some limited conditions which has to be considerate during its execution.

In conclusion, we certainly state that the association of YOLO and the Intel RealSense Camera to detect pedestrian(s) and their distance from an autonomous car, is likely to be usable in society. Despite the fact that adjustments and improvements have to be done on the application, the fundamentals performance are existing in the application, and is apt to be applied in society.

APPENDIX

Fig. 27. *You Only Look Once - YOLO logo.*

Fig. 28. *Intel RealSense Camera model*

```
array([[[18, 18, 18],
        [18, 18, 18],
        [18, 18, 18],
        ...,
        [34, 34, 34],
        [34, 34, 34],
        [35, 35, 35]],

       [[18, 18, 18],
        [18, 18, 18],
        [18, 18, 18],
        ...,
        [33, 33, 33],
        [33, 33, 33],
        [34, 34, 34]],

       [[17, 17, 17],
        [17, 17, 17],
        [18, 18, 18],
        ...,
        [33, 33, 33],
        [33, 33, 33],
        [34, 34, 34]],

       ...,

       [[ 3,  3,  3],
        [ 3,  3,  3],
        [ 3,  3,  3],
```

Fig. 29. *Display of each pixels for the Depth Frame*

Fig. 30. *photo.jpg*

Fig. 31. *photo.jpg passed through "Processing_Image.py" application*

Fig. 32. *photo3.jpg*

Fig. 33. *photo3.jpg passed through "Processing_Image.py" application*

Fig. 34. *photo2.jpg*

Fig. 35. *photo2.jpg passed through "Processing_Image.py" application*

Fig. 36. *Intel RealSense Camera - Color Frame*

Fig. 37. *Intel RealSense Camera - Depth Frame*

Fig. 38. *Frame with standing pedestrian at 5m*

Fig. 39. *Frame with knee pedestrian at 5m*

Fig. 40. *Frame with side pedestrian at 5m*

Fig. 41. *Frame with ball pedestrian at 5m*

Fig. 42. *Frame with standing pedestrian at 10m*

Fig. 43. *Frame with knee pedestrian at 10m*

Fig. 44. *Frame with side pedestrian at 10m*

Fig. 45. *Frame with ball pedestrian at 10m*

Fig. 46. *Frame with standing pedestrian at 25m*

Fig. 47. *Frame with knee pedestrian at 25m*

Fig. 48. *Frame with side pedestrian at 25m*

Fig. 49. *Frame with ball pedestrian at 25m*

References

[1] http://neuralnetworksanddeeplearning.com/chap1.html#complete_zero
[2] https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/
[3] https://www.researchgate.net/figure/Illustration-of-a-fully-connected-neural-network_fig2_333336147
[4] https://www.docker.com/
[5] https://www.docker.com/get-started
[6] https://en.wikipedia.org/wiki/Docker_(software)
[7] https://pjreddie.com/darknet/yolo/
[8] https://github.com/wizyoung/YOLOv3_TensorFlow
[9] https://www.intel.fr/content/www/fr/fr/architecture-and-technology/realsense-overview.html
[10] https://en.wikipedia.org/wiki/Intel_RealSense
[11] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets
[12] https://victorzhou.com/blog/intro-to-cnns-part-1
[13] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
[14] https://www.youtube.com/watch?v=vT1JzLTH4G4
[15] https://pjreddie.com/darknet/yolo
[16] https://github.com/wizyoung/YOLOv3_TensorFlow
[17] https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
[18] https://en.wikipedia.org/wiki/Activation_function