Interpersonal Distance Monitoring Using Deep Learning and Computer Vision

Rodrigo López de Toledo Soler

Author

Jaime Boal Martín Larrauri & Alvaro Jesús López López

Director

***Abstract*— In these weird times that we are living the last year and a half, keeping a safety distance between individuals to prevent Covid-19 from spreading has become one of the most effective measures to control de pandemic. Since we are in 2020 and technology is now our way of understanding everything, to ensure all these measures are being followed by the population, we need a way of making sure they are, and, since the number of people controlling is very small in comparison to the people that needs to be controlled, using all the supervisising methods we have, it should be at least studied. Using machine learning and the recent developments in computer vision this can be possibly handled and hopefully be useful not just for now, but for other future situations we may face. After training different models for detection, tracking and distance estimation between individuals, results show that, with some improvement, this objective can be achieved within the limits of real-time video. To achieve these results, three different versions of the state-of-the-art algorithm YOLO have been tested, then fed into a widely used tracking algorithm such as DeepSORT and finally, for the distance estimation part, we found the need of generating a completely synthetic dataset to feed color and depth information into a model and see if it can predict the distance between the individuals generated with a simulator. The results achieved are promising, but the road ahead is long and the room for improvement is big.**

Keywords — Deep learning, distance monitoring, CNN, people detection, tracking, PyTorch

I. Introduction

Computer vision is an area of study that is identified as a subfield of artificial intelligence and works in parallel with machine learning. The goal of computer vision is to understand the content of the digital images and therefore, videos, to mimic the ability of the human vision. For us humans, understanding an image is very easy, it comes natural and effortless, but for a computer it is extremely complicated because it is very difficult to understand the changes in lighting, position, orientation, and expression that we see so natural. In the early 1970s, computer vision was only an ambitious point in the agenda of giving robots a human intelligence. In the 1980s and 1990s, the focus shifted, researchers centered their efforts on more complex mathematical techniques to perform image analysis.

It is important to understand that image processing using common techniques such as image and contrast adjustment, histogram equalization and binarization is not computer vision. These techniques are very useful, but they are not computer vision because we cannot extract any analysis or knowledge from what is happening in the image.

Since the year 2010 there have been several tasks in the field that have been extensively researched and that have achieved success. Some of them have been retail[1], medical imaging[2], surveillance & biometrics[3], and motion capture[4].

The project follows the three stages proposed with some intermediate steps. It depicts the inference of the project proposed, but all the models involved, and some more, have been previously trained on different datasets and the best performing ones have been chosen for each stage, as described throughout the document.

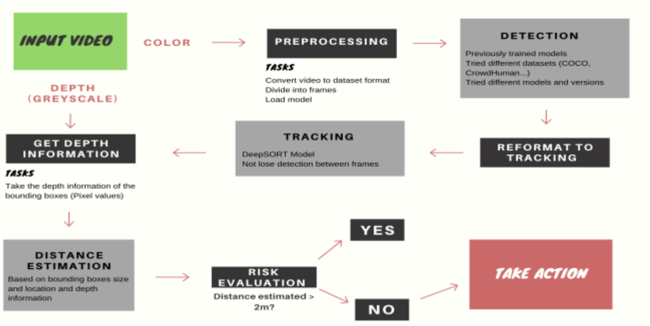

INPUT VIDEO
COLOR
PREPROCESSING
TASKS
Convert video to dataset format
Divide into frames
Load model
DETECTION
Previously trained models
Tried different datasets (COCO, CrowdHuman...)
Tried different models and versions
DEPTH (GREYSCALE)
GET DEPTH INFORMATION
TASKS
Take the depth information of the bounding boxes (Pixel values)
TRACKING
DeepSORT Model
Not lose detection between frames
REFORMAT TO TRACKING
DISTANCE ESTIMATION
Based on bounding boxes size and location and depth information
RISK EVALUATION
Distance estimated > 2m?
YES
NO
TAKE ACTION

Figure 1. Schema of the system proposed.

[1]https://www.forbes.com/sites/cognitiveworld/2019/08/22/revolutionizing-brick-and-mortar-retail-with-computer-vision/?sh=5a83412ad112

[2] P. Srinivasan and V. Srinivasna, "A Comprehensive Diagnostic Tool for Skin Cancer Using a Multifaceted Computer Vision Approach," 2020 7th International Conference on Soft Computing & Machine Intelligence (ISCMI), 2020, pp. 213-217, doi: 10.1109/ISCMI51676.2020.9311557.

[3]http://www.forbes.com/sites/forbestechcouncil/2019/09/23/four-ways-computer-vision-is-transforming-physical-security/?sh=530888c65846

[4] https://paperswithcode.com/paper/babel-bodies-action-and-behavior-with-english

II. Previous Work, Techniques and Research

Security, privacy, autonomous vehicles, facial recognition, biometrics. Those are just some of the fields where computer vision has been used and developed in the recent years. Object segmentation, object detection, and object classification have been the main tasks these problems have tackled.

In terms of image capturing, LiDAR and stereo vision are the main input sources used. LiDAR provides a point cloud that allows reconstructing the scene in 3D. Stereo vision uses two cameras with different perspectives to estimate depth. This information is useful in the last stage of the project.

While a lot of techniques and methods can be applied to the task we are tackling, the rest of the section focuses only on the ones that have been applied in this project.

A. *Traditional techniques*

Early models of sensory processing by the human brain is the inspiration behind neural networks [1]. By coding an algorithm that mimics processes of real human neurons, our newly created, artificial, neural network can 'learn'. A neuron receives input from external sources, or even from other units within the network. It weighs each input and, if the total output is above a threshold, the output is one. So, in relation to our problem, a neural network can solve a classification problem, if the target is binary or multi-class. In the scenario of video surveillance, neural networks perform low level processing of pixels and then perform classification on the outputs of these preprocessed pixels. This approach is highly adaptive, which makes it perfect for outdoor cameras processing.

The problems that are not linearly separable can be also classified with what is called a hidden layer. This, at a high level, is another threshold unit that its mission is to do partial classification. After that, the output level leverages each partial classification and comes up with a final prediction. These are called feed forward neural networks and are the first type of neural networks that appeared. The activation function can be changed into other function that can classify the objects more accordingly to what we want. In our case, for the distance estimation part, a linear activation function is needed since it is a regression problem.

In the early 2000s, two approaches were used to do a more robust and flexible tracking. These two techniques are Kalman filters and P2DHMM (Pseudo-2D Hidden Markov Model), that is actually used to feed a Kalman filter [2]. These two algorithms work together, making it possible to track people even if the background has moving objects. The main advantages of using a statistical model approach such as P2DHMM is that, a priori, it can recognize some human body shapes and learn the only the features relevant to the problem.

Kalman filters provide estimates of a variable given the measurement observed over time. It has two stages, prediction and update. The prediction stage estimates a new measurement from the output of the update stage. The filter is recursive, so a stopping constraint needs to be specified. These filters are widely used to track moving objects. This algorithm is the basis of the DeepSORT algorithm used for the tracking stage.

The Microsoft Research Vision Technology Group established in [3] the characteristics and rules that a real-life vision-based tracking system must comply with. They did that by focusing their research on depth information analysis and by clustering, they determined that we can extract shapes from that information.

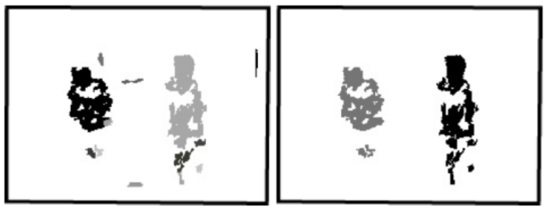


Figure 2. Results of the research [3].

Support Vector Machines (SVM) have also been used several times as classifiers for crowd monitoring tasks after doing training in the corresponding model [4]. First of all, an initial calibration of the scene is needed. This is used to build a density map of shapes within the image. A common method is to split the image in rectangles and calculate the density in each rectangle. After the image is divided, the density estimation of each rectangle and the design of the density map of the complete image are carried out.

B. *Modern techniques*

A very common method is semantic segmentation, and a technique developed by University of Cambridge researchers that has proven successful is de SegNet Encoder-Decoder. SegNet is a deep encoder-decoder consists of four layers of encoders and their corresponding layers of decoders, followed by a pixel classifier.

SegNet uses CNNs that, for our case, take an input image, assign importance to the different parameters, and differentiate one from another. While in traditional techniques the preprocessing is more archaic and tedious, the preprocessing required for CNNs is much lower. The difference with a common Neural Network is that, in NN, we can just take a matrix of pixel values and turn it into a 1D vector to use as input. In CNN we cannot do that because it can also capture the spatial and temporal characteristics of an image with the appropriate filters. The convolutional part is that the objective of the CNN is to reduce the images into a form that is easier to process without losing feature information to provide a good prediction. Depicted below is the common architecture of a Convolutional Neural Network. Based on this architecture, most modern models such as the ones used in this project like YOLO are based and it consist of a set of convolutional layers, with its corresponding pooling and normalization, the neck of the model, that serves as a connection between the convolutional backbone and the classifier, is a fully connected layer following the outputs of the convolutional layers and finally the classification output, where probabilities for each class are calculated in order to classify the input.

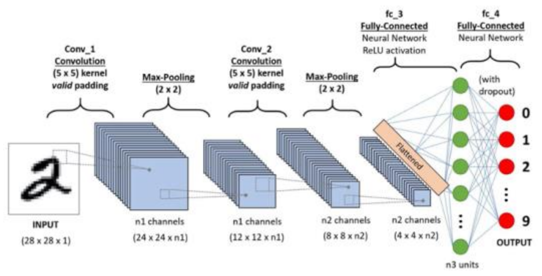


Figure 3. Architecture of a CNN.

For detectors, the basis for all the project, previous research has shown that the best approaches are region-based algorithms or regression-based algorithms. Combining the first one with CNNs we get R-CNNs. They solve the problem of selecting a huge number of regions in common CNNs by generating many region proposals by selective search and generating a feature vector classified later by an SVM. Selective search uses a greedy algorithm to recursively combine similar regions into larger ones and use them as final candidate regions. These are very computationally expensive so a variation of the Fast R-CNNs that improved the results by using a single-stage training algorithm that classifies and refine locations at the same time was proposed in [6][7]. Because of the computational requirements, regression-based algorithms appeared, like YOLO.

YOLO is a model developed for object detection that uses the target detection as a regression problem for a separate target box and its category [8]. A single neural network predicts the confidence of the box and the category. Initial YOLO implementations could only detect 49 objects and had high localization error. An evolution appropriately called YOLOv2 follows the following process, implementing batch normalization on all the convolutional layers and a high-resolution classifier [9]:

1) At first, the image is divided into a grid. If the object we want to track is in a grid, it is responsible for detecting the object. Each grid cell predicts the detection boxes and its confidence.

2) For each detection box, YOLO establishes 5 values , the coordinates of the center, height, width and confidence, all of these related to the box.

3) Prediction of the pedestrian probability for each grid

4) The probability is multiplied by the predictive value of each box, called IOU, which is the overlapping area between the prediction box and the ground truth box. This operation gives the confidence of the class as a result for this box.

There are some variations in the application of YOLO to detection. One of them is the YOLO-R network which is a recursive application of the YOLO algorithm. It combines convolutional layers alongside pooling layers (using the max criterion) in the same way as CNN and reorganization layers. This method increases the performance of YOLO by a small margin in terms of precision and recall [9]. The main advantage of YOLO is that it is faster than other methods, including Fast-RCNN. Continuous evolution in the algorithm is the main reason to start the project trying the YOLOv3 version and then, the following ones.

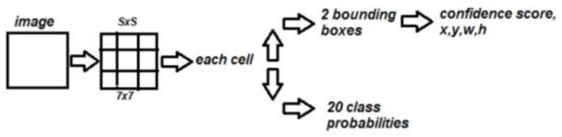


Figure 4. Simplistic Architecture of YOLO.

In terms of tracking, Kalman Filters are extensively used nowadays. So much, that the state-of-the-art algorithm in real-time tracking, DeepSORT is based on it [6][10]. The technique used in this algorithm is called tracking-by-detection. This technique divides the task into two parts, the first part is detecting the object and the second is associate detections of the same object in different time points. The first method for this technique is using graphs for tracking. Given two images in the same space in different time frames, the nodes of the graph are the detections, and the edges are the spatial and temporal differences between the images. Specifying some constraints and using different programming approaches the network flow can be solved, for example using a k-shortest path algorithm. A second method within this technique is using multiple cues. This approach uses data association to improve the robustness of the tracking system. These cues are used to combine different textures, shapes and depth coming from stereo systems. This tracking-by-detection is the preferred method nowadays in terms of usage and performance.

As a clarification, there is a tracker based on YOLO called ROLO. It implements the extensively explained YOLO architecture and attaches an LSTM network at the end. All the feature information extracted from the feature extractor (Backbone + Neck) is passed to the LSTM, which provides the bounding box predictions and the time and space information. It seems that the logical path for the project would be to use ROLO. Well, unfortunately, ROLO is, for now, only contemplated for single object detection and struggles when it must deal with data association [11].

Given that an introduction into the main methods used in this project has been made, let’s dive into the specifications of each of the models and the structure of the system.

III. DESCRIPTION OF THE SYSTEM

Given that this is a 3-stage project, it is easier and clearer if I divide this section into three, so I can explain clearly and with some degree of depth what is done in each of the steps. I will start by describing the thought process behind the election of the detection model, its features and characteristics, and how it will work in the context of this project. I’ll do the same for the following tracking model and distance estimation stage.

A. Detection stage

The growth of deep learning is what allowed us to have a modern approach to detection. As outlined before, it allows extracting more complex features of the images, getting a better representation of the feature space. There are basically two frameworks where all the previous techniques exposed are grouped in. These frameworks are region proposal algorithms, such as R-CNNs and the other framework consists in regression-based algorithms, such as YOLO. As mentioned earlier, R-CNN architecture generates the proposals of objects bounding boxes first, then does the feature extraction and finally carries out classification and localization. As main drawbacks, doing a selective search of locations is very computationally expensive and takes more time. However, its accuracy is better than regression-based algorithms as seen earlier, there are architectures such as Fast-RCNN and Faster-RCNN which can provide a faster training, but they still take considerably long. Essentially, these approaches are not very suitable for real-time applications, just by how they are built.

The regression/classification-based frameworks are different because they search for spaces in an image with high probabilities of containing an object. The main algorithm within this approach is YOLO and its different versions. Since the detection can be done in just one step, this algorithm is considerably faster than the different RCNNs algorithms. However, its accuracy is a slightly lower because it usually struggles with small objects that appear in groups.

In general, all applications where real-time processing is essential should be implemented with regression-based algorithms due to the improved speed they provide. Common tasks for this framework are autonomous vehicles. In this case it is very clear the importance of providing a real-time processing with the smallest delay possible. Taking into consideration the limitations and the advantages of all these different algorithms and the different use cases for each of them, it is logical that the optimal algorithm to use for our people detection in real-time use case is YOLO and its different versions.

The first model tried was YOLOv3. The only relevant information from the previous versions is that the backbone used was a 19-layer convolutional network and that the mAP is 48.1, as outlined in both the official website[5]and the paper [12]. The structure of this version, and the basis of the following ones is as follows [12].

1) ***The feature extractor****: To avoid the struggles of detecting smaller objects in the previous version, this one uses what they called Darknet-53[12]. It is a mix of convolutional layers and residual networks that works as follows: There are consecutive 3x3 and 1x1 convolution layers followed by a skip connection (the residual part), which prevents having to diminish the gradient too much when propagating information. The combination of the 53 convolutional layers, the residual layers and the detection head result in a total 106-layer fully convolutional architecture.*

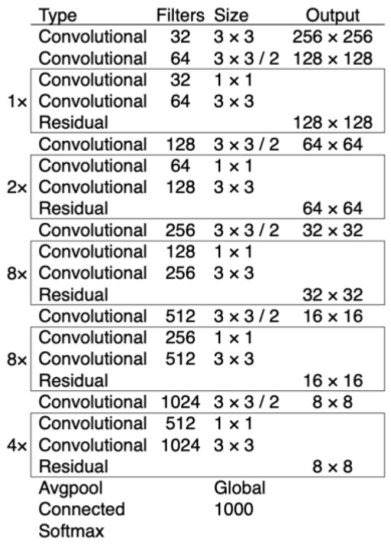
2) ***The multi-scale detector****: Derived from the name, the detector needs inputs with different scales. Depending on how the structure is defined, the detector takes the output of 3 different parts of the network and defines feature vectors of different dimensions (13x13, 26x26 and so on and so forth). The larger vectors will be used for the detection of larger objects and the smaller ones for small objects [12]. The different detection heads are placed alongside the model and the preceding convolutional network needs to have a linear activation function, let's not forget this is a regression-based algorithm. In the model used there are three detection heads in charge of providing detections for the image at three different scales. The shape of the kernel depends on the number of classes to predict, so for the different datasets, a different file is needed because the filters of the convolutional layer preceding the detection head needs to depend on the classes provided by the dataset. The formula for calculating it is the following:*

$$filters\ =\ \left(B * (5 + numclasses)\right)$$

$$where\ B\ is\ the\ number\ of\ anchors\ (YOLOv3\ uses\ 3)\ \textit{(1)}$$

The number 5 on equation (1) refers to the key parameters of the bounding boxes (height, width), (center_x, center_y) and the confidence score (1 per class).

3) The loss used in previous versions of YOLO was the sum of squared errors. It is very ineffective when doing multi-class classification. That is why in this version the authors decided to use Binary Cross-Entropy to measure loss in the predictions.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 5. Architecture of YOLOv3 [12].

YOLO works the following way. The objective of the network is to predict bounding boxes of each object and the probability of this object belonging to each possible class in the dataset. The input image is divided in a *SxS* grid, and, by the formula explained earlier, there are [B*(5+num_classes)] [12] filters.

As per the paper, the model predicts the class of the object whose center of the bounding box lies inside the grid cell. Therefore, the output of each forward pass through the network is a 3D tensor of dimension [S, S, [B*(5+num_classes)]].

Since the objects to detect are not squares, the authors use a term called "anchor boxes". They are predefined boxes with an aspect ratio set defined by a K-Means clustering on the entire dataset. YOLOv3 uses 3 anchor boxes per detection scale and the center of these boxes are the same as the centroid

[5]Yolo Official Website: https://pjreddie.com/darknet/yolo/

of the grid cells. In total, since there are 3 detection scales, YOLOv3 uses 9 anchor boxes.

There is a chance that after the single forward pass there are lots of bounding boxes for the same object (Same centroids). What we need is the box better suited for this object. The authors use what is called Non-Max Suppression to select the better bounding box. A threshold can be defined to just clear all the bounding boxes with confidence scores lower than this threshold. After this first stage, it ranks in the bounding boxes descending order and determines that the appropriate one is the one with highest confidence score. After eliminating the boxes for this grid cell, it needs to eliminate the rest by doing the IoU, Intersection over Union calculation, that measure the distance between the ground truth boxes and the predicted boxes.

$$IoU = Area\ of\ Overlap/Area\ of\ Union$$

If the overlap and the union is very similar (IoU approximately 1) it means that the bounding box predicted is very accurate. An IoU of more than 0.5 is considered a good prediction, so the first attempts with the models will be with that threshold and a 0.4 confidence threshold.

YOLOv4 and YOLOv5 work in a very similar way. Since, as showed in the results later, the best performing model and the one worth using is YOLOv5, here are the main improvements made for it, which, are very similar to YOLOv4 [13].

It uses a slightly different backbone called CSPDarknet53. It is very similar to the original Darknet53, the only difference is that it uses a partition strategy to separate the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy. This approach allows a larger gradient flow through the network. This architecture is more light-weighted and enhances the variability of the learnt features in the different layers [14].

The method used for parameter aggregation is PANet to sum the features and parameters from different levels. PANet is chosen because of its ability to preserve spatial information accurately, helping in proper localization of pixels for mask formation [13].

The main advantage between YOLOv5 [15] and the previous versions is that it is built natively in PyTorch, making the inference, parsing and building of the model much easier on the machine and, therefore, much quicker. Previous versions were built in C/C++ and when implementing them in Python, all the configuration files and parsing made the models much slower than YOLOv5 even though they are simpler and with less parameters (less accurate also).

There is some controversy about YOLOv5, because it has been published with that name even though none of the authors participated in the previous papers and, in fact, there is no YOLOv5 paper as of June 2021. However, its performance is good enough so it's worth using it.

B. *Tracking stage*

The next stage in the project is to provide some tracking within the frames to make sure that we do not lose the identification of the people that the previous algorithm detects. This can be seen as a "safety" measure because it is possible to feed an object detector directly into the distance estimator, if the detector performs accurately. However, adding a tracking mechanism in between those stages, allows the algorithm to "memorize" previous detections for a certain amount of time in case it loses a detection in a given frame, but the same person appears again in one of the next ones. It is a very efficient method to deal with occlusions.

In this project, the approach will be tracking-by-detection. This approach takes consecutive frames and tracks detections through them. This is different for example than re-identification, which attempts to match a detection in a given image with a previous detection. Re-identification is not a real-time approach. Since our input will be a video feed, tracking-by-detection is the obvious choice for the task.

The most popular tracking algorithm is DeepSORT. It is an evolution of the SORT (Simple Object Real Time Tracker) [51] algorithm. The main difference between the two algorithms is that DeepSORT replaces the Hungarian method association metric used by SORT with a CNN network.

The authors of the paper [10], decided to use the Mahalanobis distance, which allows them to take into consideration the uncertainty, and work better with distributions. The authors threshold the Mahalanobis distance metric at a 95% confidence interval.

$$\boldsymbol{d^{(1)}(i,j) = (d_j - y_i)^T S_i - 1 * (d_j - y_i)}[10]\ (2)$$

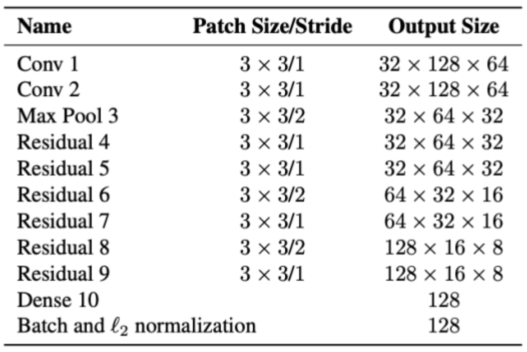
In equation (2), d_j denotes the j-th detection and S_i and y_i denotes the spatial information of the i-th track [10]. Despite the effectiveness of the filters and the associations provided by the Hungarian algorithm, it still struggles with occlusions and different viewpoints, that is why the authors [10] included a deep learning approach to the previously existing algorithm. Deep learning is used to fix this by introducing an appearance metric based on CNNs. The idea is to obtain a feature vector that accurately describes the images. After obtaining the feature vector the distance metric is updated like this:

$$\boldsymbol{c_{(i,j)} = \lambda\, d^{(1)}(i,j) + (1 - \lambda) d^{(2)}(i,j)}[10]\ (3)$$

In equation (3), $d^{(1)}$ is the Mahalanobis vector used before, $d^{(2)}$ is the cosine distance between the feature vectors of the current and previous image and $\boldsymbol{\lambda}$ is the weighing factor. The authors [10] claim that the importance of $d^{(2)}$ is much more than $d^{(1)}$ that they were able to achieve good results using $\boldsymbol{\lambda = 0}$, nullifying the Mahalanobis distance, when there is considerable camera motion, therefore more occlusions. Using this deep learning feature increased the power and accuracy of DeepSORT considerably. To compare consecutive frames in terms of appearance descriptors, the algorithm takes the minimum cosine distance between the i-th track and the j-th detection [10], as seen in equation (4), where R_i denotes the collection of appearance descriptions of the previous bounding boxes, and their tracks.

$$\boldsymbol{d^{(2)}(i,j) = min\{1 - r_j^T r_k^i \mid r_k^i \in R_i\}}[10]\ (4)$$

In summary, the Mahalanobis measure provides useful information for short term detection for bounding box localization while the appearance descriptor considers information that is very useful to recover predictions after some time. The CNN architecture used is like this:



Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Figure 6. Architecture of DeepSORT's CNN [10].

C. *Distance estimation stage*

For the last part of the project, the system needs to be able to estimate the distance between 2 individuals and be able to monitor it. There is very little research regarding DL approaches for this. There are a lot of examples of distance measuring, but they are almost all of them based on a parameter of distance to get depth and all that information is hardcoded and fixed for each camera and perspective. The closest approach to a deep learning one is this one that transforms the camera view into bird's view [16], a zenithal take of the same surroundings. The complication is that, before changing the perspective into a bird's view, you must lock the objects you want to monitor [16]. That is not practical at all for us because with that, the system would not be able to track new people entering the scene.

To clarify once again, the model to be built is some simple model with a couple convolutional layers and a linear output, meaning that we will use a regression approach to solve the problem. The model needs, first, a dataset of images. The previously used datasets are not valid for several reasons. The First and foremost, there are no annotations of the distance between the people in the images. Also, these are very complex datasets for this problem, at least at this stage where this task is relatively new. There are too many people, and it would be very difficult to estimate the distance between all of them. The goal is to simplify the problem and use simple images with only two individuals, and then see if the model is able to estimate the distance between them. The third requirement is some depth information, which the datasets do not provide.

To overcome these problems, a synthetic dataset was generated using CoppeliaSim [6], a robot simulator. With this Several scenarios will be generated where there are two people and then the simulator will measure the distance between them. These scenarios will be saved as images and the distances as the labels. This way I am going to construct a synthetic dataset. Using the simulator, the depth images can also be saved and can be used as part of the dataset. Since the previously trained detector is very accurate, in order to help this final model, the images generated from the simulator will be passed through the detector it will get the bounding boxes and put them on the depth images, this way, the model does not have to deal with colors or other information, it only needs to take the grayscale values of the depth and, with the help of the bounding boxes, estimate the distances provided in the labels. The following images explain this a little better.

The simulator used is CoppeliaSim, which is a simulator built for robot tasks, but it is open source, and it might be enough.

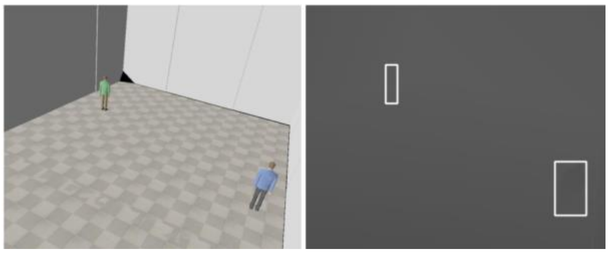
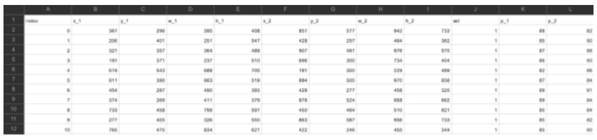


Figure 7. Scenario generated in the simulator and tagged with the detector

The first image in Figure 7 is the output of the simulator. As seen, it generates two people in random positions within the limits of the scenario. Those images are passed through the detector and the bounding boxes are "translated" to the depth image, as seen in the image on the right. No difference can be seen between the individuals and the background in the depth image. This is because the difference between pixel values is so small that it is not perceivable by the human eye, but, in theory, the computer is able to differentiate them, giving the information of the distance to the individual from the camera´s point. Ideally, this dataset would have several perspectives in order to have a model that generalizes correctly when the camera changes position but with the hardware available, creating a dataset that big is impossible. For now, this dataset is good enough for the objectives of this task.

After several attempts with different configurations of the simulator and the models for a long time creating a CSV dataset became an option. This dataset includes all the bounding box information and their centers pixels values to train only a DNN instead of a full CNN, this dataset is like this sample.



	A	B	C	D	E	F	G	H	I	J	K	L
1	index	x_1	y_1	w_1	h_1	x_2	y_2	w_2	h_2	dst	p_1	p_2
2	0	361	296	395	408	851	577	942	733	1	88	82
3	1	200	401	251	547	428	257	484	362	1	85	90
4	2	321	357	364	489	907	461	976	570	1	87	86
5	3	191	371	237	510	696	300	734	404	1	86	90
6	4	619	543	688	705	191	350	229	489	1	82	86
7	5	611	390	663	519	884	505	970	656	1	87	84
8	6	454	287	490	393	428	277	458	320	1	89	91
9	7	374	269	411	379	876	524	958	662	1	89	84
10	8	733	458	799	597	450	464	510	621	1	85	84
11	9	277	405	328	550	863	587	956	733	1	85	82
12	10	765	475	834	621	422	246	455	344	1	85	90

Figure 8. CSV format dataset

The distance estimation models are built on both of these datasets and depending on their performance it can be evaluated if it is possible to estimate the distance between individuals. This dataset contains the coordinates, heigh and width of both individuals and the pixel value of the center of the bounding box for each individual, representing some depth information.

While it would be possible to train the model without the depth information, it is useful because they work as a safety measure for when the bounding box doesn´t correspond to the parameters of height and distance of the individuals, for example, when one has an armed lifted, the bounding box reaches the full length of the arm, that could be at 2.2 meters,

[6] CoppeliaSim Developers: https://www.coppeliarobotics.com

way above the average height. Also, if there are partial occlusions, the bounding box of the partially occluded individual will start from what is visible, meaning that it would not represent the entire height of the person. For that, the depth information is valuable.

IV. Results

As in the last section, it is easier to present the results of the three different stages in order to evaluate each of them and be able to make decisions on these partial results.

A. *Detection stage*

For this stage several models have been trained and tried on several datasets (On the GPUs provided by Google Colab, normally a Tesla P100). I have tried YOLOv3, YOLOv4 and YOLOv5 (A smaller, medium, and larger version) all of them in COCO[17] and CrowdHuman[18] datasets and only YOLOv3 and YOLOv4 on KITTI[19], the reason behind that is that KITTI performed significantly worse than the other two on the first two models, so I decided to stop using the dataset. The reason behind the lack of performance is clearly that this dataset is designed for autonomous driving problems and all the images are from the same perspective (a car), so, when generalizing to another perspectives, results are much worse than the other datasets that have way more variability. An exploratory analysis of CrowdHuman Dataset vs. KITTI is provided next to understand that.

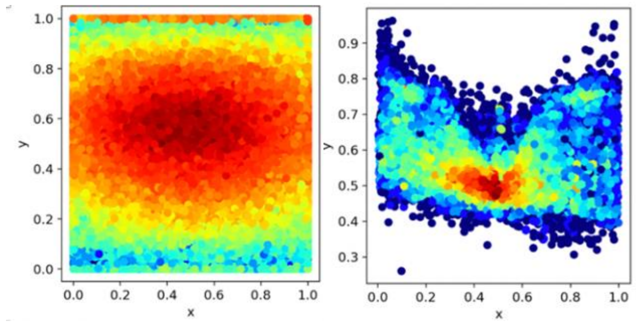


Figure 9. Exploratory Analysis of KITTI and CrowdHuman.

As seen in the images, the center coordinates of the bounding boxes of CrowdHuman (that all of them represent humans) are distributed way more heterogeneously through the picture dimensions than KITTI.

Training YOLOv5 in CrowdHuman provided the training results that are shown in Figure 10. These results, compared to the ones COCO provided are way better, since the model couldn´t get more than 0.65 mAP in any of the COCO models (YOLOv3 got 0.45 mAP), probably because it is a dataset with more classes than people, not like CrowdHuman that is dedicated to persons.

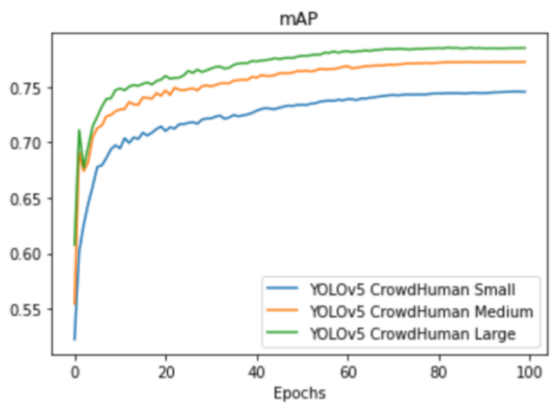


Figure 10. Training Results of CrowdHuman.

As seen, the larger model reaches almost 0.8 mAP (0.786 to be exact), and the medium one 0.764. While the larger model has more accuracy, taking into consideration the real-time objectives this project has, it seems better to use the medium sized model because is faster on inference. I have done a comparison of all the model tried including the number of detections and the FPS that each of them reaches.

All ratios shown in the table are calculated against the number of detections that the best model did, taking into consideration accuracy and speed. The ratio only measures the number of detections the model has been able to make in the set of images fed into it. All models were fed the same set of images.

TABLE I. Comparison Of Results of the Detection Stage

Model	Dataset	Ratio Of Detections	FPS
YOLOv3	COCO	0.44	37.1
	KITTI	0.27	
	CrowdHuman	0.79	
YOLOv4	COCO	0.5	25.3
	KITTI	0.03	
	CrowdHuman	0.63	
YOLOv5 - Small	COCO	0.48	78.4
	CrowdHuman	0.95	
YOLOv5 - Medium	COCO	0.46	53.0
	CrowdHuman	1	
YOLOv5 - Large	COCO	0.5	33.9
	CrowdHuman	1.02	

As aforementioned, when choosing the best performing model in this stage and using it to train the tracker after, it is needed to account for the accuracy and the processing speed. Taking in consideration that the limit of real-time processing is commonly put at 30 frames per second, any model performing below that number can be discarded. Because of the difference in accuracy, all YOLOv3 models were discarded and, because it is very much on the limit and not at all much better in accuracy, the YOLOv5 large model of CrowdHuman was discarded also. The last two models standing are the YOLOv5 small and medium models trained on CrowdHuman. I decided in favor of the medium sized one since it has a very wide margin in terms of FPS and is a little bit more accurate in large distance shots. In the next images are presented some images of this detector working and, in the conclusions, I provided links to all the results of all the models.

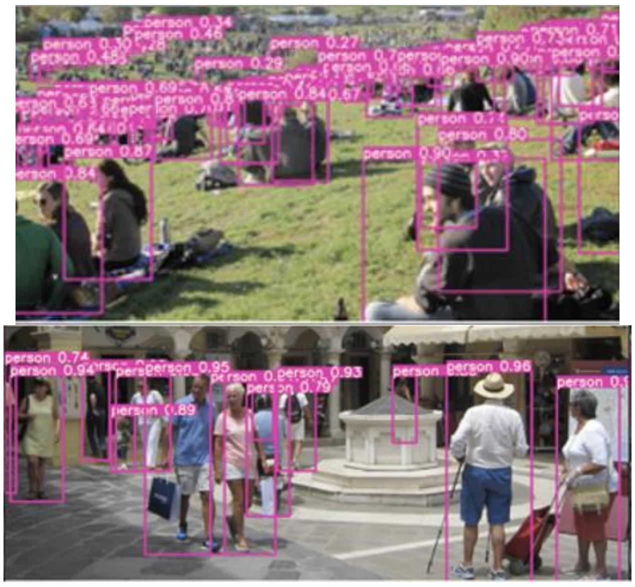


Figure 11. Inference examples of YOLOv5 trained on CrowdHuman.

B. *Tracking stage*

The authors in the paper evaluated the model using the MOT challenge [17]. This challenge is specifically designed to track objects in public locations, including people and the results were very good, as shown in the paper.

Taking this into consideration, the decision was to use the already trained model of DeepSORT and put it after the detectors I trained before. Training DeepSORT includes training also the detectors so, since the detectors are already trained, it makes a lot of sense to use the already pretrained DeepSORT model and use for inference the detector trained. The model provided by the authors allows selecting different parameters for the inference such as IoU, Confidence and Age of Tracks, which is a feature of DeepSORT that allows choosing the number of frames you want to keep a lost track´s information for. This allows the model to not overload the tracks matrix and keep the model lighter. It works in a very simple way, whenever a track is lost, because of detector mistake, occlusions etc., you can set the number of frames you want that track information to be kept so you can match it with another detection. For example, setting the age to 50 frames, it can “recover” the detection if that same object appears in the following 50 frames, but not after. If the age is very high, the inference time will increase. Personally, for this task, 60 frames age is chosen initially and then checking the inference time to see if it works or if it needs to be reduced. For this case, if the frame is lost for more than 60 frames, at 30 fps is 2 seconds, it is considered to be a detection that is going to appear either on a completely different location of the frame or it is not going to appear again at all after that time.

The results in terms of FPS for detection are shown below. I had some time and tried it also on the other models but, my previous conclusions stand.

TABLE II. Comparison Of Results of the Tracking Stage

Model	Version	FPS
YOLOv3	-	33.2
YOLOv4	-	22.1
YOLOv5	Small	69.0
	Medium	52.0
	Large	32.3

As seen in the table, there is not much difference when applying the tracker, except in the YOLOv5 - small model, that got its FPS reduced by 10 frames. The rest are similar, and we still maintain the 20 FPS margin from the real-time limit in the model selected as the best performing one. In the following images, forming two sequences are presented the results of the tracker in action on some videos of situations where it might be necessary the distance monitoring that has been talked about.

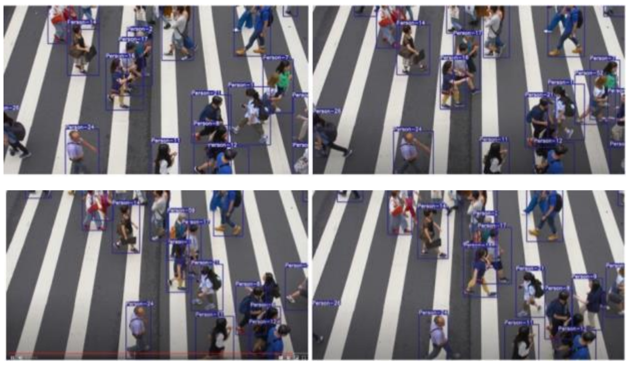


Figure 12. Sequence images of DeepSORT inference.

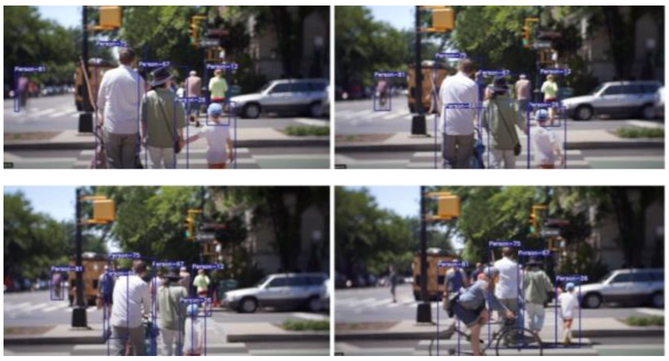


Figure 13. Sequence images of DeepSORT inference – 2.

C. *Distance estimation*

For this stage several tryouts were made also, including trying different configurations in the simulator like including walls, adjusting the range of the vision sensor, and generating the people with different orientations. For clarification, the dataset was generated with walls surrounding the scenario and carefully adjusting the range of the sensor. The individuals were generated in the 4 main orientations taking the camera as the point of view. Not only that but different configurations of the model were tried also. For example, transfer learning from different models such as InceptionV3 or MobileNetV2, simple models generated from scratch were also tried and a DNN model to work with the CSV dataset showed before. In the following lines are shown the two best performing models that also allows us to understand that the objective can be reached from two different paths.

The first one is using a CNN model. As seen in the curves of figure 14, the model is a bit unstable but much more precise, which for this problem, is a much better option because it is going to be able to differentiate better between pixels. The other models tried reached, the best ones, 1.6 MAE that, taking into consideration that MAE for this model is 0.93 meters in the training dataset and 1.2 in the test dataset, proves that this model and dataset is the best performing ones among the CNN models tried. It also does not overfit as the previous model did. All distances are in meters.

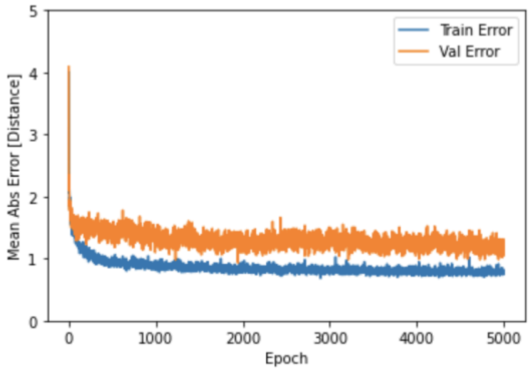


Figure 14. Training curves of the CNN model.

The architecture of the CNN model is as follows. Using a pretrained model like MobileNetv2 [20], the model used adds dropout and a flattening layer to the output of the pretrained model. It adds after it a 30-neuron fully connected layer and a 15-neuron fully connected after it. With a dropout layer afterwards, the models' output is a single neuron with a linear activation function. The objective for this architecture is to provide a little bit of adaptation from the pretrained model to the synthetic dataset used.

If we put in a scatter plot the predicted outputs against the supposed results, the following result shoes. As seen, the predictions follow the ground truth values, while not perfectly, with enough degree of accuracy. While it has some trouble predicting short distances, with some improvement, both in the model and in the dataset, the accuracy will increase automatically. As before, all distances are in meters.

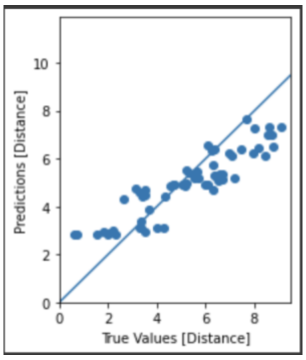


Figure 15. Scatter Plot of the CNN model on Test Dataset.

The other model is the DNN model. Even though this is a different approach, the structure is similar. The dense layers substitute the CNNs, and the output is linear once again. Training the model for 1000 epochs, the training curves are as follows:

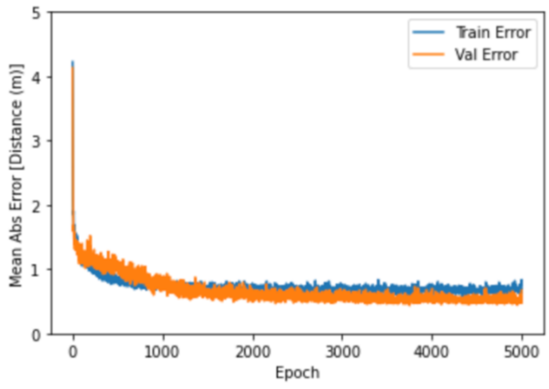


Figure 16. Training Curves of the DNN model.

Even though this model is affected by noise, it does not overfit and is more stable. It still shows a little bias on the scatter plot, but less than the previous model. It also has much less variance than the CNN model. In summary, it reduces the bias and the variance from the model with CNNs, meaning that is better than it. The only drawback it has is that it still suffers slightly when trying to predict either very long distance or very short ones, which is the real problem, although it deals better with short ones than long ones. These improvements, translated to a metric, mean that the MAE of this model is 0.6 meters, that, while not perfect, it is much less than previous models.

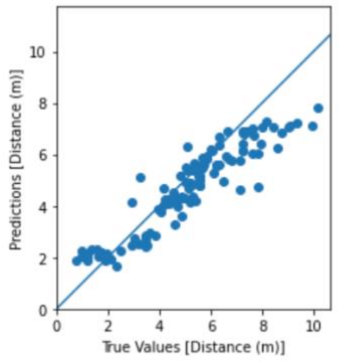


Figure 17. Scatter Plot of the DNN model on Test Dataset

In the table below a summary comparison of the models is presented.

TABLE III. COMPARISON OF RESULTS OF THE DISTANCE ESTIMATION STAGE

Model	**Training MAE (m)**	**Test MAE (m)**
CNN – Transfer Learning	0.93	1.2
CNN	1.6	2.1
DNN	0.78	0.6

V. Conclusions & Future Work

Considering the results presented, mainly in the last section, the monitorization of the interpersonal distance using deep learning while having a processing speed within the boundaries of real time (30 FPS) is possible. However, I´ll provide now a brief summary of the results of each section and the conclusions that can be drawn from them.

For the detection stage, 9 different detectors spread out in 4 different datasets (COCO, KITTI, Caltech and CrowdHuman) and 3 different models have been tried. Trying all of them in chronological order, I was able to discern why each of them perform the way they do. Starting with YOLOv3, it provided a relatively low accuracy for the COCO dataset but a very reasonable one for both KITTI and CrowdHuman. However, in inference, the COCO model was the second best performing one just behind CrowdHuman. KITTI was not able to generalize that well due to the nature of its dataset, as explained in the corresponding section. In terms of FPS, this model provided a speed of around 30 FPS, which is very much on the limit of real time.

The evolution of YOLOv3 into YOLOv4 should mean better results, but that was not the case. YOLOv4 model is a much deeper model that, in training, it's true that the results were better than YOLOv3, but in inference, that difference in accuracy was not tangible. In fact, there was only a slight improvement in the COCO dataset, but not in CrowdHuman or KITTI, the worst performing one clearly. In terms of FPS, since this is a much heavier model than YOLOv3, its performance was far from real time processing, so I decided to discard all YOLOv4 models for the next stages, they were less accurate and slower than YOLOv3.

Then YOLOv5 appeared in the scene and the results were all better than what we saw before. Not only in training, where the smaller model was already more accurate than all the previous models, but also on inference time. As explained before, YOLOv5 has the same structure and advances as YOLOv4, but it is much faster and precise because it is built natively on PyTorch, not in C++ like YOLOv4 and YOLOv3 model is. Reducing all the translating and parsing of documents and a simpler structure was enough to have much more accuracy and reduced the inference time considerably, reaching a 60 FPS processing speed for the model selected. In terms of datasets, CrowdHuman was the best performing one by a considerable distance, so I decided to stick with the YOLOv5 model trained on CrowdHuman dataset for the next stage.

So far, we have a very accurate model, that has some difficulties detecting people from very long distances and has some trouble with occlusions. However, it is very fast. For now, long distance detection is not a problem of much concern, as long as short distance detection is good, which it is. The occlusions, however, are a problem and we need to deal with them. That is why we incorporate a tracking mechanism. To try to avoid occlusions I relied on DeepSORT. It is based on Kalman Filters but incorporates a Deep Convolutional Network that works as an appearance descriptor between frames. Comparing the output of it to determine if a detection is very similar to one in the previous frame, we can "recover" lost detections some frames later.

Looking at the results and inference time of the YOLOv5 model + DeepSORT model, the outcome is very satisfactory. Occlusions are corrected in a lot of cases. Of course, this is not an exact science and between frames, as explained, one detection can occupy the space of another and be very similar. Unfortunately, some of those occlusions are not corrected, but that is rare and overall performance is very good. In terms of FPS, the processing speed is obviously reduced since the images pass through another model, but the FPS of the models selected are still above 50 FPS, a speed well above the limits stablished. Since the last stage is a very simple and light model, the processing speed is not expected to be reduced so, the real-time objective is reached with a considerable margin.

For the last stage, we encountered the problem the datasets we had used so far were not usable. To estimate the distance, we need some kind of distance annotations to train the model against them and check accuracy etc. There are no datasets like this so, in order to reach a deep learning solution for the problem, we decided to create a synthetic dataset. For this we used a simulator and generated a dataset of 600 images of 2 individuals with the distance between them measured and stored as the labels. This is very archaic, and it is not a state-of-the-art dataset but, as shown in the results, is good enough for now. While using a very simple dataset and a very simple DNN model, we reached an MAE of only 0.6 m, that is not excellent but is good enough, to check that estimating the distance is possible using Deep Learning. In the section following this one I´ll discuss how that can be improved.

With all these stages producing good results, I can determine that estimating the distance between individuals is possible using a Deep Learning approach and with real time processing speed. Of course, there is much work ahead, but this could be a first step towards a much more general and precise solution in the future.

A. Future work

In terms of detection and tracking, there is much to improve, but since the FPS is very good in the project, which was the main objective of these stages, it is a more researched field, and the part that needs more improvements is distance estimation, I´ll focus on what can we do to improve that last one.

To begin with, generating a better dataset is key. As mentioned, the simulator we used is not designed for this type of tasks. The right thing to do would be to use a photorealistic simulator in order to generate better images and depth information. For example, it only managed to provide a pixel value range between 80 and 92. That is not good enough and is not very precise since, ideally, a range between 0 and 255 is what the simulator can give. I was able to generate 600 images and create the dataset but having more time to generate a bigger dataset (or having a more powerful machine) could be of great help. Time was limited and what was generated was good enough.

Apart from that, which I believe would be enough to provide a very good solution, a deeper DNN model with a more complex structure could be of help also. Since we have more than 20 FPS of margin between the actual system with the selected model and the real time limit of 30 FPS, investing in a deeper model should improve the results accordingly, even if it slows the process a little bit.

As an end note, incorporating that last stage into the pipeline could be done more efficiently since, now, the input depth information is reworked into the 80-92 range of pixel values to be correctly inferenced. With a better dataset and better simulator knowledge, we could train the DNN with a 0-255 range and then, no reworking of the input depth information would be needed, but, for now, that is the way it can be done.

VI. References

[1] K. Sage and S. Young, "Security applications of computer vision," in IEEE Aerospace and Electronic Systems Magazine, vol. 14, no. 4, pp. 19-29, April 1999, doi: 10.1109/62.756080.

[2] G. Rigoll, B. Winterstein and S. Muller, "Robust person tracking in real scenarios with non-stationary background using a statistical computer vision approach," Proceedings Second IEEE Workshop on Visual Surveillance (VS'99) (Cat. No.98-89223), Fort Collins, CO, USA, 1999, pp. 41-47, doi: 10.1109/VS.1999.780267.

[3] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale and S. Shafer, "Multi-camera multi-person tracking for EasyLiving," Proceedings Third IEEE International Workshop on Visual Surveillance, Dublin, Ireland, 2000, pp. 3-10, doi: 10.1109/VS.2000.856852.

[4] X. Wu, G. Liang, K. K. Lee and Y. Xu, "Crowd Density Estimation Using Texture Analysis and Learning," 2006 IEEE International Conference on Robotics and Biomimetics, Kunming, 2006, pp. 214-219, doi: 10.1109/ROBIO.2006.340379.

[5] N. Deepika and V. V. Sajith Variyar, "Obstacle classification and detection for vision based navigation for autonomous driving," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 2092-2097, doi: 10.1109/ICACCI.2017.8126154.

[6] Joel Janai; Fatma Güney; Aseem Behl; Andreas Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art," in Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art , now, 2020.

[7] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE

[8] Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[9] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis, "Deep Learning for Computer Vision: A Brief Review", Computational Intelligence and Neuroscience, vol. 2018, Article ID 7068349, 13 pages, 2018. https://doi.org/10.1155/2018/7068349

[10] W. Lan, J. Dang, Y. Wang and S. Wang, "Pedestrian Detection Based on YOLO Network Model," 2018 IEEE International Conference on Mechatronics and Automation (ICMA), Changchun, 2018, pp. 1547-1551, doi: 10.1109/ICMA.2018.8484698

[11] N. Wojke, A. Bewley and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3645-3649, doi: 10.1109/ICIP.2017.8296962.

[12] G. Ning et al., "Spatially supervised recurrent convolutional neural networks for visual object tracking," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050867.

[13] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement" 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1804.02767.

[14] A. Bochkowsky, C. Y Wang and H.Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection”, 2020 ArXiv, 2004.10934

[15] C. Wang, H. Mark Liao, Y. Wu, P. Chen, J. Hsieh and I. Yeh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," 2020 IEEE/CVF

[16] Deepak Biria´s Project: https://blog.usejournal.com/social-distancing-ai-using-python-deep-learning-c26b20c9aa4c

[17] COCO Dataset: https://cocodataset.org/

[18] CrowdHuman Dataset: https://www.crowdhuman.org/

[19] KITTI Dataset: http://www.cvlibs.net/datasets/kitti/

[20] K. Dong, C. Zhou, Y. Ruan and Y. Li, "MobileNetV2 Model for Image Classification," 2020 2nd International Conference on Information Technology and Computer Application (ITCA), 2020, pp. 476-480, doi: 10.1109/ITCA52113.2020.00106.