Person Following Robot Behavior Using Deep Learning

Ignacio Condés[1] and José María Cañas[2]

[1] Universidad Rey Juan Carlos, ignacio.condes.m@gmail.com,
[2] Universidad Rey Juan Carlos, jmplaza@gsyc.es

Abstract. The proposed following behavior combines a *perception* and a *control* module. Perception module addresses the person detection on images using a pretrained TensorFlow SSD *Convolutional Neural Network*. It provides *robustness* over traditional methods, although care has to be put on not losing real-time operation. A *person tracker filter* has been included to alleviate the effect of false positives/negatives. It also extracts faces, which are also analyzed by a *FaceNet* CNN to reidentify the tracked individual. This perception module tells which person in the image has to be followed, even on poorly lightened scenarios. It is combined with depth readings to obtain the relative position of the person. The control module implements a case-based *PID* controller for a reactive smooth response, moving the robot towards the target person. The entire system has been experimentally validated[3] on a real Turtlebot2 robot, with an Asus Xtion RGBD camera.

Keywords: Computer Vision, neural networks, deep learning, robotic behavior

1 Introduction

In this project, we aim to establish a scope which covers two fields: *robotics* and *deep learning*.

In the last decades, robots have become a powerful allied to humans, as they are leveraged to perform all kind of tasks (hazardous explorations, personal assistance, cleaning, driving, ...). However, we aim to design them to perform these tasks on the most autonomous way possible, which means not requiring to be controlled on every action (with exceptions, such as surgeon robots). This requires to provide robots a certain intelligence and capabilities to correctly trigger the most suitable action for each possible input stimulus. For this purpose, we can find multiple approaches for different problems (navigation, conversation, ...).

Concretely, this article is dovetailed with *Computer Vision*, which involves connecting cameras to a robot, and taking advantage of this on an autonomous

[3] https://jderobot.org/Naxvm-tfg

way. Concretely, we will tackle the *person following* challenge, which is based on a behavior governed by a person *detection* system. Many approaches are already existing, such as color filters [1], or disparities filtering. However, promising *Deep Learning* techniques excel on their image processing variant, as they offer high quality results, accompanied by *robustness* facing lighting issues (Fig. 1). This is the set of techniques we have used on this project, so we will describe the basis firstly.

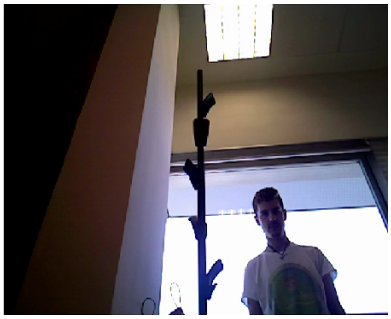

Fig. 1. Harsh lighting situation for *Computer Vision* algorithms.

In the image detection quandary, the main objective is to determine *presence and position* of a certain object in a given image. For this, we make use on this article of *deep learning* techniques, concretely *Convolutional Neural Networks* (CNNs).

This investigation has focused on exploring the synergies existing the two explained fields, making use of *deep learning* to automatically command a robot to follow a determined person. This is achieved making use of two different modules: a *perception* one, focused on the computer vision/deep learning tasks, and an *actuation* module, which implements a case-based behavior, depending on what the perceptive part senses and computes. The designed system has been experimentally validated, as it will be seen on section 6

2 State of the Art

There are not any other techniques for the moment, we have invented robotics and deep learning entirely.

3 Infrastructure

The system proposed on this article follows the structure represented in the Fig. 2. As it has been introduced, it is composed by two blocks. The first one

(*perception* block) is responsible for performing the *vision* tasks. Therefore, it determines, with the maximum possible precision, the position of every person found inside the image. It is here, as well, where it is discerned which of these persons is the one to be followed, in case it is seen. The second one (*actuation* block) is designed to read and process that output from the previous block, and send suitable movement commands to the robot.

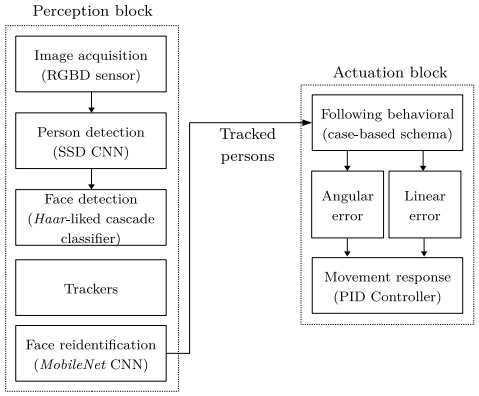

Fig. 2. General structure of the proposed system.

These mentioned blocks run on a computer, connected to both the RGBD sensor and the robot. The block processes are executed on a concurrent way, offering the advantage of a correct *timing adaptation*. Hence, the system is designed to run properly on different hardware specifications on the base PC.

4 Perception Module

This is the first block of the proposed system. As it name indicates, it is responsible for apprehend the incoming images (RGB and depth) from the sensor, and generate a significant output to be interpreted by the actuation module. Its structure follows a certain pipeline along the images, which will be described next.

4.1 Person Detection

On first place, the incoming RGB images are passed through an *object detection* module. It is powered by a CNN, which has a special architecture designed in

[2] for this purpose: SSD (*Single-Shot Multibox Detector*). This kind of detection technique stands out by its prediction speed, because *it performs a single feed-forward pass* of the image through the network, unlike the rest of *state-of-the-art* techniques. These other approaches perform successive feed-forward passes through the network, what makes them consequently much slower, as it can be seen on a timing experiment performed during this project, with different detection architectures (Table 1).

Architecture	**Base network**	**Dataset**	**Mean inference time** (ms)
ResNet	Inception	COCO	820.71
SSD	MobileNet	COCO	107.43
ResNet	101	COCO	786.49
ResNet	50	COCO	515.28
ResNet	101	COCO	63.97
Faster-RCNN	ImageNet	ILSVRC2014	703.99
Faster-RCNN	Inception	COCO	352.20
ResNet	50	COCO	793.87
SSD	MobileNet	COCO	102.85
ResNet	101	COCO	898.59
Inception	ResNet	OID	792.42
SSD Lite	**MobileNet**	**COCO**	**68.13**
ResNet	101	Kitti	111.29
Inception	ResNet	OID	667.76

Table 1. Timing performance tests for several detection models. The selected implementation appears in **boldface**.

On this structure, the desired image to input has to be reshaped to 300×300 px, which is a typical shape for this type of CNN architecture. As visualized on Fig. 3, it extracts a set of activation maps on its *base network* o Feature Extractor (*MobileNet*) in the selected network model, and position and class inferences are made later, on multiple image scales. In last place, a *Non Maximum Suppressor* is applied, retaining only the most confident detections, which are adjusted to a correct bounding box shape.

This system provides an accurate and efficient object detection, returning for each processed image:

- *Classes:* the detected classes (person, cell phone, airplane, dog ...) inferred for each detected object.
- *Scores:* the confidence $\in [0, 1]$ the network has on each object belonging to the decided class.
- *Boxes:* the coordinates of the rectangular *bounding box* which wraps the detected object, expressed as the coordinates of two opposite corners of it.

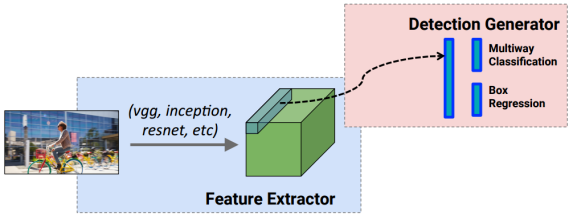

Fig. 3. Schema of the SSD architecture.

Although the used SSD model is capable to detect up to 80 object classes, the proposed system only retains those detection corresponding to *persons*, as it is what we are interested to follow.

Hence, this results on a light *person detection*, perfectly capable to work on a real-time operation on a standard level hardware. Our implementation is capable to handle different network models and architectures on a *plug and play* manner, just pointing the model file (in the specific TensorFlow `.pb` format) to it.

4.2 Face detection

Once the existing persons on the image have been detected, the next phase is to search their faces, in order to know whether any of them corresponds to the person to follow. In order to do this, we implement the classical Viola and Jones face detection algorithm (presented on [3]). This algorithm, which comprises *Haar* features (Fig. 4), is a simple algebraic method which takes advantage of the typical illumination pattern of a face (due to its physical shape) to detect promising regions of the input image to contain a face. To test a *Haar* feature on a specific grayscale image, the feature is slided through it, and a simple operation is performed (black pixels subtracted to white ones). If the result is positive along all the feature, the region where it has been applied has passed the test.

The image introduced in the system is divided into *regions*, which are passed through a *cascade* of tests, where the non-compliant regions are immediately discarded. The accepted ones pass to a slightly more complex feature each time, and the regions which pass all the features are supposed to contain a face. This progressive region dismiss makes it an efficient algorithm, capable of run simultaneously with the rest of processes. This entire process is performed with an OpenCV[4] method.

As the previous block detected persons inside the image, this face detection algorithm is only applied inside the instance of the detected persons, in order to avoid false positive detections.

[4] Open source image processing library.

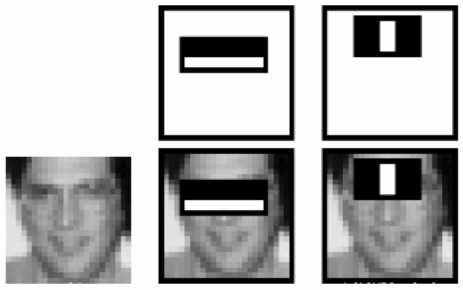

Fig. 4. *Haar* features applied on a face.

4.3 Person and face trackers

The previously described methods perform respective person and face detections, and although they behave in a robust way, the obtained output can suffer spurious components, due to lighting or occlusive conditions. As this can interfere with the desired following capability, the system palliates the effect of these false positive/negative detections implementing a time-spatial *trackers*, which filter the detection outputs (*candidate detections*). This filter attributes each person/face its coordinates inside the image, and takes into account the number of successive frames that a new response has been detected. If it surpasses a *patience* threshold, it is taken as reliable (*tracked detections*). Otherwise, it is taken as a spurious output. This way, a partial occlusion, which hampers a particular person detection, does not cause the elimination of that person, until it has not been lost for a while. The same happens with void detections which generally happen on a punctual way on a frame.

This tracking procedure can be observed in Fig. 5, in the *person detection* case, but it is applied in the same way for *face detection* in the proposed system.

4.4 Face reidentification

The previous trackers turn the spurious output of the detection systems into tracked and reliable detections. Hence, they can be used to perform *identification* tasks. For this purpose, a parallel CNN, *FaceNet* (presented in [4]), is used. This network *maps* a face image, extracting some key features, into a 128-dimensional Euclidean space, where faces are represented by what is called *embeddings* (feature vectors). The Euclidean L^2 distance (Eq. 1) existing between two of this embeddings stands for the *face similarity* between that faces.

$$d(\mathbf{f_1}, \mathbf{f_2}) = \sqrt{\sum_{i=1}^{128} (f_{1_i} - f_{2_i})^2} \tag{1}$$

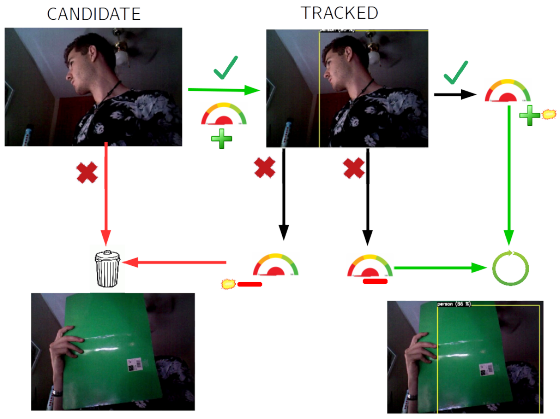

Fig. 5. Tracking process for a person detection.

Hence, we can consider that two embeddings belong to the same face if their distance is below a threshold, which has experimentally been set to 1.1.

The proposed solution makes use of this, computing on real time the embeddings of each detected (and tracked) face. Once this has been performed, it compares the similarity between these embeddings and the one corresponding to the target person (computed when the program starts, from a given image of the *person to be followed*). Something to consider is that, to avoid penalties on similarity due to lighting conditions, a previous blurring and *prewhiten* (on Eq. 2, with x as the color channel, μ as its mean and σ as its standard deviation) are performed on each face inputted on the *FaceNet*.

$$x' = \frac{x - \mu}{\sigma} \tag{2}$$

This can be checked in Fig. 5 , where it can be observed that the same face in different lighting situations yield embeddings with a distance lower than the threshold.

All this described pipeline allows the system to discern if the target person is being seen right now. In addition, it is not necessary to detect its face every time, as the person tracker is capable to infer that a new person detection on a near location to the last position of the target person corresponds to the new location of the tracked person, without needing to see its face.

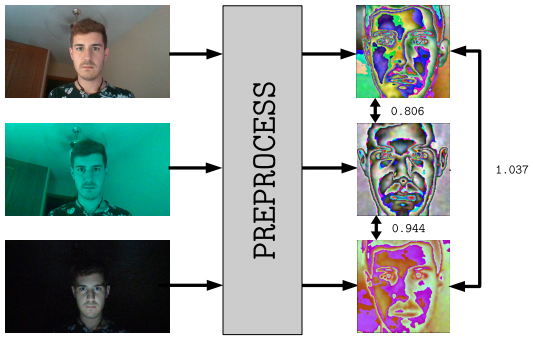

Fig. 6. Relative distances between the same face on different lighting situations. Color patterns on the right images are not representative, as they depend on the numeric range of the plotted image.

5 Actuation module

After the *perception* tasks, the system has obtained the maximum possible certainty about the persons present in the current RGB image, as well as their condition of being or not being the one to be followed.

The next functional block, the *actuation* one, is responsible of generating a suitable command to the robot motors, in order to move towards the followed person, in case it is being seen in the current frame. To do so, it follows its own pipeline, explained next.

5.1 Following behavioral

As it has been stated, the input for this element is the information yielded by the *perception* block: the *tracked persons* parameters (position, face, "is or not the followed person"). From this information, the new *actuation* block has to infer which attitude to take (depending on the state of the system on the last iteration). This is implemented with a *case-based* behavioral, which follows the *flow chart* represented on Fig. 7. It can be seen that the response depends on *mom* (the semantical name given to the followed person), as it can be *lost* or *tracked and followed*.

If the followed person is found among the tracked persons, the system follows it. It can observed as well that, as it was mentioned before, the system does not need a continuous face feedback to follow the target person, as it just needs to be still seen (that time-spatial continuity is provided by the previously described

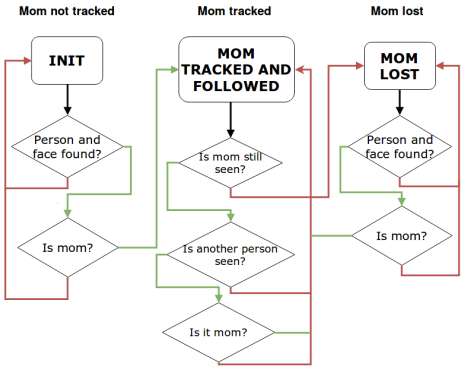

Fig. 7. *Flow chart* followed by the case-based behavioral, depending on the previous state.

trackers). In addition, if a new face is found, and it satisfies the criteria of similarity with the reference face, the *followed* role is swapped to that new person, which begins to be followed by the system.

5.2 Error computation

Once the system has recognized the target person into the image, in case it is being seen, it proceeds to an *error computation*, in order to determine the strength of the required movement to move towards that person. As the robot used on the proposed system moves in two dimensions, we perform a dual error computation:

- *Angular error:* the most accurate position for the system with respect to the person is to be aligned with it. This way, the person appears right in the horizontal center of the image. Hence, we can compute the *angular error* as the subtract of the center coordinates and the center of the person's bounding box, in the horizontal dimension. This process, represented in Fig. 8, will give a quantization of the necessary turn to be performed.

- *Linear error*: as the implemented sensor is a RGBD camera, it will provide the system a *depth* image. As it is aligned with the RGB image, the coordinates of the person inside the depth image are the same than the RGB ones. Hence, it allows to *locate* the person inside the depth map.

  In consequence, a 10×10 grid sampling of the depth values inside the person box (putting care on avoiding the margins, in order to measure only inside the person) allows to collect a serie of measured distances to the person. This system performs that process, and computes the *median* of that set of

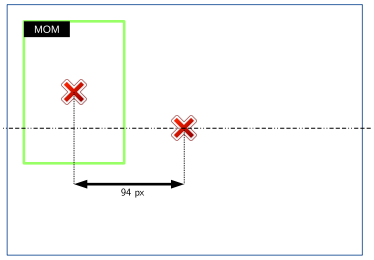

Fig. 8. Computation of the angular error between the system and the followed person.

measures, taking the result as the distance from the robot to the person, as seen on Fig. 9.

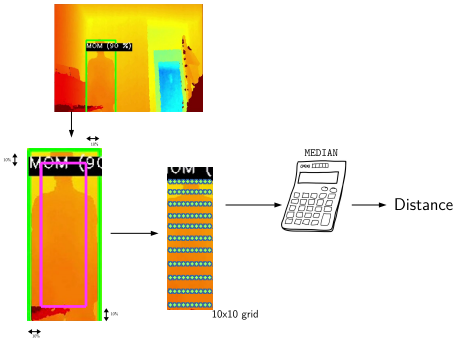

Fig. 9. Computation of the linear error between the system and the followed person.

So far, the system is capable to determine a numerical error value to measure the magnitude of the required response.

5.3 Movement response

The previous measures are required to determine the *relative position* of the robot and the target person. It is used as an input to compute the most suitable response.

As the system is not designed to move literally to the position of the person, but to just maintain a following behavior, a *dead zone* is established in each

dimension. These dead zones (illustrated on Fig.) will cause the robot not to move anymore towards the person, as the person is considered *under control* when it is placed inside.

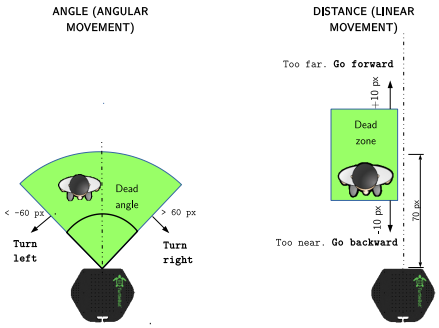

Fig. 10. Dead zones on each dimension, where the person is considered under control.

If the person is outside these dead zones, a physical response is required, with the purpose of *carrying* that person inside the dead zone. For this action, each dimension implements a *PID* controller, which establishes a *closed-loop* feedback system, as described on [5]. This allows to keep in mind previous responses, to achieve the optimum fitting in every iteration. This means, for example, to accelerate if the person is not going any closer, or to step hard on the brake if the person suddenly gets too close.

The implemented *PID* controllers (an angular and a linear one) have been experimentally tuned to obtain the most suitable parameters for our operation, obtaining the values in Table 2.

	Linear	**Angular**
k_p	2	7
k_d	0.1	0.5
k_i	3	10

Table 2. Optimal found values for the parameters in each PID controller.

This way, the system can output a speed value with a tight adjustment to the values required by the situation of the current iteration. The response obtained from this value is a *reactive* one. This means that each value results in a new movement command, avoiding to perform movements longer than one

iteration. For softness sake, what is sent to the motors is not this value, but the *mean* between it and the last sent one. This way, it results on a slightly longer convergence that helps to remove sudden movements.

The previously described pipeline is executed *once* per thread iteration, maintaining on concurrent threads each one of the two blocks (*perceptive* one and *actuative* one). This results on a new speed command to each one of the two available motors (angular and linear advances) on each iteration.

6 Experiments

This entire approach has been experimentally validated on a real system[5], composed by:

- Asus Xtion Pro Live RGBD sensor.
- Turtlebot2 robot.
- Standard laptop computer. Hardware:
  - CPU: Intel i5-4210U
  - RAM: 8 GB DDR3L
  - GPU: Nvidia 940M

The results have yielded a fine following system, which only follows to the indicated person, with a refresh rate of 10 movements per second (as the SSD CNN is the lightest possible model, as seen on Table 1. This helped substantially to minimize the time bottleneck). The tracking and reidentification process can be observed on the highest region of Fig. 12. The following process can be observed on the lowest region of Fig. 12.

7 Conclusions

In this paper we have presented a personal following behavioral, taking advantage of one of the possible synergies between both presented fields: *robotics* and *deep learning*, which are yielding a really promising results in these days.

As it has been explained, this system implements two concurrent blocks, performing operations related to *perception* and *actuation*. This leads into a fast

[5] Full video test available on https://www.youtube.com/watch?v=oKMR_QCT7EE

a) Frontal view b) Side view

Fig. 11. Experimental set where the system has been validated.

and efficient following system, capable of keeping the track on an individual person even without a continuous checking of its identity (its face, in this approach).

On an experimental point of view, the infrastructure allows to test different detection and reidentification models, in order to implement more robust ones to perform a slower but preciser following, or to connect the system to a different motor system. The source code of the entire system is available online[6]

This allows a simple robot to perform a successful tracking of a person, just knowing its face beforehand.

References

1. Calvo, R.: Comportamiento sigue persona con visión direccional, 2004.
2. Lui, W., Anguelov, D., Erhan, D., et al. SSD: Single-Shot Multibox Detector. *CoRR*, abs/1512.02325, 2015.
3. Viola, P., Jones, M. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I-511-I-518 vol.1, 2001.
4. Schroff, F., Kalenichenko, D., Philbin, J. *FaceNet*: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.038032, 2015.

[6] https://github.com/roboticsurjc-students/2017-tfg-nacho_condes

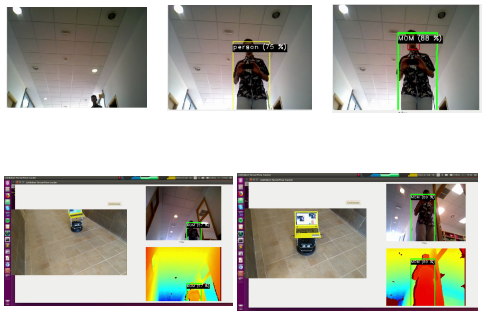

Fig. 12. Tracking and reidentification process for a person (up). Following process (down).

5. Åström, KJ., Murray, RM. Feedback Systems: An Introduction for Scientists and Engineers, 2004.