electronics

MDPI

Article

Robust Person Identification and Following in a Mobile Robot Based on Deep Learning and Optical Tracking

Ignacio Condés [1], Jesús Fernández-Conde [2], Eduardo Perdices [1] and José M. Cañas [2,*]

[1] JdeRobot Organization, Alcorcón, 28922 Madrid, Spain; 100396879@alumnos.uc3m.es (I.C.); eperdices@gsyc.urjc.es (E.P.)
[2] Signal Theory, Communications, Telematics Systems and Computation Department, Fuenlabrada Engineering School, Rey Juan Carlos University, Fuenlabrada, 28942 Madrid, Spain; jesus.fernandez@urjc.es
* Correspondence: josemaria.plaza@urjc.es

Abstract: There is an exciting synergy between deep learning and robotics, combining the perception skills a deep learning system can achieve with the wide variety of physical responses a robot can perform. This article describes an embedded system integrated into a mobile robot capable of identifying and following a specific person reliably based on a convolutional neural network pipeline. In addition, the design incorporates an optical tracking system for supporting the inferences of the neural networks, allowing the determination of the position of a person using an RGB depth camera. The system runs on an NVIDIA Jetson TX2 board, an embedded System-on-Module capable of performing computationally demanding tasks onboard and handling the complexity needed to run a solid tracking and following algorithm. A robotic mobile base with the Jetson TX2 board attached receives velocity orders to move the system toward the target. The proposed approach has been validated on a mobile robotic platform that successfully follows a determined person, relying on the robustness of the combination of deep learning with optical tracking for working efficiently in a real environment.

Keywords: deep learning; mobile robot; person identification; person following; optical tracking

Citation: Condés, I.; Fernández-Conde, J.; Perdices, E.; Cañas, J.M. Robust Person Identification and Following in a Mobile Robot Based on Deep Learning and Optical Tracking. *Electronics* **2023**, *12*, 4424. https://doi.org/10.3390/electronics12214424

Academic Editor: Giuseppe Prencipe

Received: 26 September 2023
Revised: 21 October 2023
Accepted: 25 October 2023
Published: 27 October 2023

1. Introduction

Digital cameras and high-resolution sensors have become much more affordable to produce over the past few years, making them more accessible to consumers. Nowadays, in addition to high-quality web cameras or even driving assistance cameras in cars, everyone has at least two cameras on their cell phone. This fact has contributed to a strong drive for computer vision research, combined with an improvement in hardware performance. There are many possibilities outside industrial environments for camera applications, such as fancy image modifications or autonomous driving.

The latest times have been notoriously active in this field because of the massive use of deep learning for addressing high-complexity tasks, such as language understanding [1], speech recognition [2], and computer vision problems, which are linked to the growing interest in computer vision. This extensive use was spurred by the ImageNet classification contest, where a large, deep convolutional neural network [3] achieved an overwhelming victory over other approaches [4]. This discovery, along with the significant advances in computing power and parallel computing, has stimulated the usage of these technologies, which show an outstanding performance with the currently available means.

In addition, robotics applications have provided useful assistance in daily tasks. This research field takes more interest when the behavior of a robot tends to emulate humans or even pets (e.g., https://www.engadget.com/2018/01/08/new-sony-aibo-first-impressions/, accessed on 25 October 2023), with the advantage of no people being exposed to significant risk, or, in a less gloomy scenario, without the human body's physical

limitations. This requires refined (and complex) behavior. In this sense, two main branches have emerged in robotics:

- Teleoperated robots: They are capable of performing specific actions, remotely controlled by a human operator. This type is mainly used in hazardous or high-precision environments [5,6]. Some advances have been made in improving the teleoperation function, implementing feedback from the robot, such as haptic feedback [7], or VR (Virtual Reality) sensation, to allow persons to sense the environment as if they were in the robot's position;
- Autonomous robots: These robots are much more complex machines, distinguished for implementing a response by themselves, independently of any remote operator. This is crucial in some scenarios where there exist factors (such as the time elapsed performing an action or the cost of a control link on the robot) with considerable weight in the design [8]. The state-of-the-art techniques for these robots try to emulate human behavior, so robots' actions can be performed autonomously with a certain degree of intelligence.

There are outstanding synergies between robotics and artificial intelligence. As it is explored in this work, the knowledge of these two fields is combined to implement a mobile robot capable of reliably identifying a certain person and reactively navigating toward that person, using visual perception based on deep learning.

This article presents an embedded system for following a person, making use of a pipeline of three convolutional neural networks: a Single-Shot Multibox Detector [9] for person detection, faced [10] for face detection, and FaceNet [11] for face recognition. Additionally, it features an optical tracking system for supporting the inferences of neural networks, allowing one to determine the position of a person using an RGB-D (Red-Green-Blue-Depth) camera. The optical motion tracker, integrated into a multi-threaded software architecture, is based on a combination of the Lucas–Kanade visual tracker [12] and the Shi–Tomasi corner detector [13].

The system comprises two main components: the perception block, in charge of processing the images from the camera, and the actuation block, which moves the robotic base according to the relative position of the person to be followed. The integration of this system in social robots [14], which are intended to follow a person at home or in a hospital, is bound to be very beneficial.

The novel content of the presented system is the integration of the following characteristics:

- The proposed system integrates three neural networks for person identification. These networks perform inferences over the images captured by the RGB-D sensor, which is attached to the system as the sensing source of the robot. The inferences are devoted to detecting the different persons in the scene, as well as distinguishing them by employing a discriminant feature: their faces. All detection and identification tasks are based on neural networks, achieving high robustness and reliability;
- The system includes a person tracker based on optical flow. This tracker aims to guess the trajectories followed by each person detected by the robot, allowing the robot to follow the person. At the same time, the neural network yields a new update, as it takes considerably less time to predict the person's displacement. As a result, the robustness of the entire system is improved compared to a version governed exclusively by neural inferences, which are sensitive to visual occlusions. The introduction of the tracker soothes the robot's movements, ensuring more robustness in its observable behavior;
- The final system is based on a commercial off-the-shelf (COTS) low-consumption System-on-Module (SoM) mounted on a battery-powered mobile base. This embedded solution features a high-performance Graphical Processing Unit (GPU). This assembly can operate independently without requiring an external computer to perform deep learning inferences or run algorithms in parallel.

The rest of the article is organized as follows. Section 2 discusses the background and the main related work. Section 3 addresses the system's functional description, including the software architectural scheme. Section 4 is related to evaluating the performance of the different parts of the system. We will end by summarizing the main conclusions in Section 5.

2. Related Work

Regarding the implementation of a mobile robot with a camera that can follow a specific human, there are multiple approaches to this problem, and various solutions have already been proposed.

2.1. Visual Person Detection

One of the most common visual person detection approaches is the Viola-Jones detector [15]. This algorithm relies on a rigid body model, which fits a specific shape. This shape can be typically distinguished through pixel intensity levels on a grayscale image. Although this method was initially designed to detect faces, the rigid body model allows generalizing its usage for detecting different objects, such as persons. For this purpose, several spatial filters called Haar features are introduced, used across the image to look for the intensity pattern of each template, which should resemble a part of the rigid body. Since this detector provides a weak decision, several filters (previously chosen in a training process) are combined into a boosted cascade. A person is detected if the weighted combination of several filters is triggered inside a particular area, which is decided to contain a person potentially [16].

The open-source standard image processing library, OpenCV, includes pre-trained models (https://github.com/opencv/opencv/blob/master/data/haarcascades, accessed on 25 October 2023), which can be directly used with their Viola-Jones implementation. Scale invariance can be achieved by evaluating the image at multiple scales at runtime.

Another common approach for person detection is based on HoG (Histograms of Gradients) [17]. This method computes local features utilizing the intensity gradients across the image, quantizing them according to their orientations, and creating a histogram of oriented gradients for an image block. These gradients are collected in 64×128 windows and treated as features. These features are evaluated by a linear SVM (Support Vector Machine), which is trained to classify a window as a person/non-person. This detector will perform best when the person to be detected stands in that specific pose.

These methods have the advantage of using image gradients, which can be computed efficiently, represented compactly using a histogram, and provide reasonable performance. Their main drawback is the generalization capability, as successful detection highly depends on the person's pose.

Consequently, detection techniques have moved towards the spreading paradigm of deep learning, especially the most salient tools in image processing: CNNs (convolutional neural networks). CNNs are based on standard neural networks, which combine many neurons or perceptrons organized into layers. These perceptrons implement simple, non-linear operations that allow abstract feature extraction (after a proper training process), gaining in complexity when the number of internal layers increases. When a neural network is composed of many hidden layers (in addition to the input/output ones), it is placed into the deep learning paradigm.

In the case of object detection networks, although the output varies depending on the implementation, it is generally composed of a set of (location, probability) tuples, one for each class the network is capable of detecting. In the activation map of an object detection network, the map presents higher values in the regions with a high probability of containing the object of the class it is designed for. On a convolutional layer, each neuron computes several activation maps across the dimensions of the input data.

One possible application of this concept is focused on region-based convolutional neural networks (R-CNNs) [18], which require a previous step on the image called region

proposal. This step is devoted to finding potential regions on the image to contain an object. The main challenge is to label these regions according to the objects contained inside, reducing the problem to a classification task. However, finding these regions and iterating over them makes the process too slow for real-time requirements. A notable effort has been made in later works [19,20] to reduce this computation time.

An outstanding object detection architecture is the Single-Shot Multibox Detector (SSD) [9]. The main benefit of this architecture is that it embeds the required computations in a single neural network, reducing the complexity requiring external region proposals. This reduces the computational time when the network processes an image. This architecture can be split into several stages, namely:

1. Reshape: the first task to be addressed by the network is to reshape the input image(s) to a fixed size on which the rest of the layers work. In the case of an SSD detector, this shape is n $\times$ 300 $\times$ 300 $\times$ 3 (n is the size of the input batch, as n images can be evaluated simultaneously by the neural network). This image size offers a good trade-off between performance and computational load;
2. Base network: this first group of layers is reused from a typical image classification model, such as VGG-16 [21]. The first layers of this architecture are utilized in this design, truncated before the first classification layer. This way, the network can leverage the feature maps from the classification network to find objects inside the input image. Following the first part of the network, several convolutional layers are appended, decreasing in size, to predict detections at multiple scales. The base network can be different from VGG-16, such as a MobileNet [22], highly optimized for running on low-end devices limited in computing power;
3. Box predictors: for each layer in the base network, an image convolution is performed, generating a small set (typically three or four) of fixed-size anchors, with varying aspect ratios for each cell on a grid over the activation map. These maps have different sizes, so the system can detect objects in different scales. The anchors are then convolved with small filters (one per depth channel), yielding confidence scores for each known class and offsets for the generated bounding box. These scores are passed through a softmax operation that compresses them into a probability vector;
4. Postprocessor: as several detections might be triggered in the same area for different classes and scales, a non-maximum-suppression [23] operation is performed at the output of the network to retain the best boxes under the combined criteria of detection score and Intersection over Union (IoU) score, which measures the overlapping quality between two bounding boxes.

Another interesting approach is the YOLO (You Only Look Once) system [24]. Its main advantage is its inference speed since it performs a single analysis on the entire image, dividing it into a grid of cells. Each cell predicts up to 5 boxes containing an objectness score (the predicted IoU of the proposal for an object, regardless of its class), the coordinates of the bounding box, and a probability for the object belonging to each class. Although this design runs faster than other methods, it performs poorly when detecting small objects.

This design was revisited in YOLO9000 [25], introducing several improvements such as batch normalization at the input of the convolutional layers or the concept of anchor boxes: the box proposals follow a fixed set of aspect ratios, chosen previously using clustering on a training set. Limiting the proposal shapes to 5 fixed sizes improves the performance while maintaining a high IoU metric. The selected anchors seem like a reasonable shape for most objects the network aims to detect. Additionally, the number of deep layers has been increased from 26 layers to 30, and semantic modeling is performed on the labels across different datasets, allowing the network to be trained in other datasets under a standard semantic structure called WordTree.

The latest improvement of YOLO, YOLOv3 [26], relies on residual networks [27], which confront the problem of vanishing gradients when the networks become deeper. The stacking of several layers results in gradients diminishing their value until the arithmetical precision of the machine cannot handle it. The gradients are canceled, hindering the

training process, as the first layers' parameters take substantially longer to converge. The residual networks added in this design revision add shortcut connections across the layers, focusing the backpropagation gradients on the differences between the input and the output of the layer. The combination of these residual layers and convolutional ones allows the training of much deeper architectures (53 convolutional layers), capable of yielding a higher generalization.

As in the SSD detectors, the YOLO architecture performs multi-scale detections, using three scales for splitting the feature maps into cell grids. A similar k-means clustering is performed on the COCO dataset, selecting nine anchor sizes instead of 5 and grouping them in 3 scales. Now, on each of the cells, nine anchor bounding boxes are fit (3 anchor shapes $\times$ 3 scales). This aims to improve the poor performance of the previous version when dealing with small objects and produce better generalization: in the R-CNN and the SSD, the anchor shapes are hand-picked. These changes, with tuning on the error function, conform to the YOLOv3 improvements over the previous versions.

For each (anchor, cell, scale) combination, YOLOv3 predicts:

- The coordinates of the object within the anchor;
- Objectness score, computed by logistic regression to maximize the probability of overlap with a ground truth bounding box concerning any other prior anchor;
- 80 scores, as the original implementation is trained in the COCO dataset, which contains 80 classes. These classes might be overlapping (e.g., "woman" and "person"). Thus, these scores are computed by independent logistic classifiers and are not passed through a softmax operation.

2.2. Person Identification

In a controlled environment, where the only present person is the one to be followed, a person detection system could be enough to follow that person. However, in a real scenario, there might be several people inside the robot's field of vision. This problem can be approached by employing a distinguishing feature of the person of interest provided beforehand. One example is [28], which computes the color distribution of the person of interest and later compares this distribution with the ones belonging to the different persons using the Bhattacharyya coefficient [29], a measurement of similarity between two probability distributions.

This metric can be applied to compute the similarity between the color histograms of the reference person and the detected one. However, this system can be deceived by replicating the color distribution of the person of interest: wearing similar clothes helps to reduce the distance between the histograms, leaving a chance to confuse the person to follow with another one.

A more robust approach involves using the face of the person as the discriminant feature, as its uniqueness makes it an excellent reference to identify the detected person. As summarized in [30], several applications extract facial landmarks from the morphology of a given face, using them to recognize the face, comparing it with a set of known faces, and estimating the identity based on the distance to each known face. Some open-source libraries, such as dlib and OpenCV, provide algorithms to perform these processes.

The intuition behind these methods is to project the image of the face into a lower-dimensional space, allowing extraction of significant features from each face. These features must be consistent across different poses and lighting conditions for the same face. A beneficial transformation when a dimensionality reduction is pursued is Principal Component Analysis (PCA), a linear transformation that can be implemented to deal with the face recognition problem [31].

Neural networks can be leveraged in order to achieve better performance: as PCA is a linear operation, it could be learned by a single-layer neural network. Thus, the introduction of deep networks can yield exciting results. The most relevant approach so far uses deep convolutional networks for performing this process, implementing an

architecture called FaceNet [11], which is partially based on the Inception [32] module, designed to reduce the number of parameters in a neural network.

This network computes an embedding, a projection of the input face image into a point in a 128-dimensional hyper-sphere. This allows translating the identification into linear algebra terms, such as distance between two faces, and clustering and applying unsupervised algorithms. These networks can be trained using a triplet loss function inspired by the work in [33]. Given a training sample (anchor), a positive example (same class as the anchor), and a negative example (different class than the anchor), the network is tuned to maximize the anchor-negative embedding distance and minimize at the same time the anchor-positive one.

The algorithms described above perform the operations on the face image. Thus, face detection is required for previously cropping the face of the person to be identified. One interesting approach using this technique is "faced" [10]. This is a custom small ensemble of two neural networks responsible for detecting faces and correcting the bounding boxes found. As the system's main objective is speed, the central detector architecture is based on YOLO, and the second correction stage raises the precision achieved by the detector, achieving better results than the classical Haar approach.

2.3. Person Following

Several approaches have been developed to pursue the challenge of following a person. Once the visual perception algorithms are established, the final output of the pipeline issues a movement command for the robot to move toward the desired point. Mobile robots can be classified according to their locomotion capabilities. A robot is holonomic if the number of its controllable degrees of freedom is equal to its total degrees of freedom. In the case of a holonomic robot, the navigation process is simplified, as the robot can instantaneously move to the desired target. However, a non-holonomic robot must perform maneuvers to move toward a point, as the controllable degrees of freedom are lower than the total. An interesting classification of some existing people following algorithms and their applications is shown in [34].

Some approaches leverage the detected objects to estimate the relative homography of the orthogonal planes, allowing a partial knowledge of the robot's environment and the tracing of a safe path toward the person. Other approaches act without a path planning component, implementing reactive behavior [35]. In these approaches, the vector between the image's center and the person's center is used to command the robot's movements.

2.4. Embedded Deployment

Generally, robotic systems integrating a person detection module are deployed using laptops connected to robots, as the implementation in an autonomous robot imposes a power limitation on the algorithms to be deployed. However, the increasing interest in real-time computer vision applications has fostered the development of specific low-power embedded devices to be incorporated into mobile systems. The extensive usage of devices such as Arduino or Raspberry Pi has led to embedded robotics systems capable of running simple vision and navigation algorithms at a low cost.

Unfortunately, the requirements for running more complex algorithms, such as those based on neural networks, require the next tier in power terms. The appropriate device could be an Application-Specific Integrated Circuit (ASIC), as the custom design would lead to a very tight performance optimization. However, the objective is to run the algorithms on existing software frameworks, requiring the usage of general-purpose computers instead.

The most remarkable advance in this scope is the appearance of Jetson devices manufactured by NVIDIA. These development boards are SoM computers running a tailored version of Linux. The fundamental feature of these systems is that they include a high-performance GPU featuring a low-level parallel computation library named CUDA, as well as several toolkits, such as TensorRT (https://developer.nvidia.com/tensorrt, accessed on 25 October 2023), designed to optimize the software implementations for the plethora

of possibilities to be designed on this board. Its size and power consumption make this system a remarkable choice to be included in an autonomous robot.

2.5. Related Systems

A cost-effective approach for human-centered autonomous navigation in the context of domestic robotic assistance is proposed in [36]. The core of their robot assistive solution relies on the idea that keeping the platform oriented towards the subject permits the robot to continuously check its status when the robot is moving and avoiding obstacles typically present in a realistic indoor environment. To this end, they first set up a real-time visual perception pipeline that reliably provides the person's coordinates in the robot reference frame using an RGB-D camera. However, they do not integrate a deep neural network to recognize a specific user in the visual perception pipeline.

A person-tracking method exploiting the RGB and depth data that can overcome the deceptive challenge introduced by similar-appearing distractors is presented in [37]. The proposed system is robust to different challenges of occlusion and distraction by the uniform crowd and thus provides a resilient perception ability to the robot. The pioneer3 AT robot is the mobile platform with an RGB-D camera placed one meter above the robot and a laptop with a GPU as the control unit.

In [38], a system that implements the follow-person behavior in a mobile robot is presented. Its perception module addresses the person detection on images using a pre-trained TensorFlow SSD Convolutional Neural Network, which provides robustness even in harsh lighting conditions. It uses Haar features for face detection, including a FaceNet CNN to identify the target person. The TurtleBot-2 robot was used in the experiments, including an RGB-D sensor and a regular laptop equipped with an NVIDIA GPU attached to the robot's structure as the onboard computer.

In [39], an embedded system for following a person is presented, using a pipeline of convolutional neural networks. In addition, it features an optical tracking system for supporting the inferences of neural networks, allowing the robot to determine the position of a person using an RGB-D camera. This research extends that work by enhancing the performance of the optical motion tracker, now integrated into a multi-threaded software architecture. New experiments are performed to validate the improved technique.

In summary, none of the systems considered above can robustly identify and follow a specific person by combining a pipeline of CNNs and an optical tracker in an embedded SoM, keeping the power consumption and the economic cost of the system low. This is accomplished by a hardware/software architecture that integrates COTS components and a multithreaded software application.

3. System Design

The functional architecture of the presented system can be observed in Figure 1. It has been logically divided into two main components or modules: the Perception module and the Actuation module.

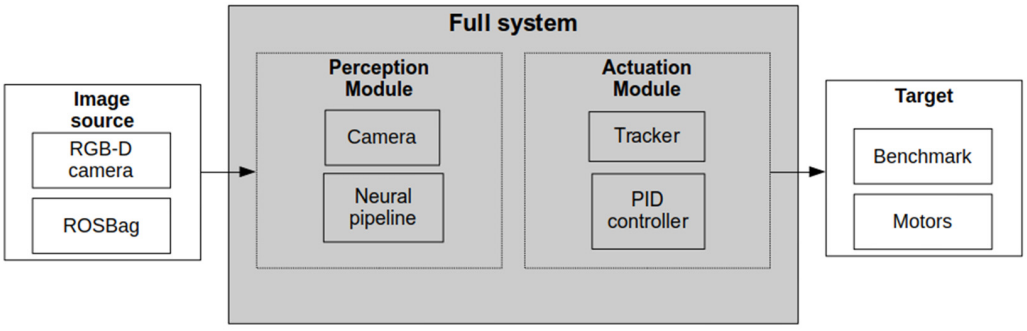


Figure 1. Functional architecture of the system.

The position and identification of the individual who needs to be followed are the two items that the perception module must be able to deduce from the image. These data are used as input by the Actuation module, which decides what action must be taken to move the robot in the person's direction. Whenever the person's position changes, these movements must be reactive and occur quickly.

3.1. Perception Module

This module encompasses the data received by the robot from its perception sensors (the camera, in this case) and its subsequent processing to determine the location of the person to be followed. It comprises the following components (see Figure 1): camera and neural pipeline. These components are detailed in the following subsections.

3.1.1. Camera

The robot's visual sensor is an RGB-D camera, specifically the ASUS Xtion Pro Live. This device yields two simultaneous images: RGB and depth images. The ROS controller for the camera, OpenNI2 (https://structure.io/openni, accessed on 25 October 2023), fetches the image and registered depth map from the camera, making this information available through several ROS topics.

As ROS follows publisher–subscriber semantics, once the driver is up and running, any application may subscribe to the topics in order to receive all the published messages. In our Camera module, two subscribers are deployed to retrieve the latest (RGB, depth) pair asynchronously. These images are then converted into the standard image format in the OpenCV library and ready to be used by other components. Additionally, the Camera module can retrieve images from a recorded ROSBag instead of the online camera to perform objective testing and benchmarks.

Recording an ROSBag allows saving in a single file the messages read from several topics when it is recorded. Later, the ROSBag can be played again to recover the messages from the topics in the same order they were recorded. This is useful to obtain objective metrics of other software components in unit tests.

3.1.2. Neural Pipeline

The captured images are passed through a pipeline of three neural networks, which provide the capability of detecting the persons in the scene and identifying which one is the one to be followed. The complex problem of determining the identity and location of the person of interest has been decomposed into three tasks, which are all addressed using the corresponding deep-learning techniques:

1. Person detection: several deep learning object detector models with different base network architectures and depths have been tried. Only those architectures that produce a good performance on a mobile (low power) device with an adequately fast inference time are considered. The SSD model [9], which employs a MobileNet for feature extraction, and the tiny YOLOv3 [26] model have been evaluated for this purpose. These models have already been trained and are openly accessible on GitHub (https://github.com/mystic123/tensorflow-yolo-v3, accessed on 25 October 2023) and on the TensorFlow Model Zoo [40];
2. Face detection: this issue can be addressed using a detecting neural network. The accepted solution is a single-class detection system because the earlier-mentioned models are unsuitable for detecting faces. The two-stage neural network used by the network developed in faced [10] is capable of detecting faces. Based on a class-specific neural network, its YOLOv2 detector ensures a fast and effective identification. A video sequence comparing the accuracy of this system against a classical Haar cascade approach [15] can be seen at https://github.com/iitzco/faced (accessed on 25 October 2023);
3. Face identification: once a person's face has been detected, it can be used as a discriminating characteristic to establish their identification. A publicly accessible implemen-

tation of TensorFlow (https://github.com/davidsandberg/facenet, accessed on 25 October 2023) has been utilized to carry out the identification for this task using the neural network FaceNet [11]. This deep identification method converts the image of a face into a projection (or embedding) of a 128-dimensional vector. This transformation is learned after a triplet-loss training procedure that projects similar faces as closely as possible while separating different faces as much as possible. As a channel-wise normalization step is carried out before running the picture through the network, it provides identical projections when two photographs of the same face are analyzed despite varying illumination circumstances.

Utilizing the flexibility provided by deep learning techniques, this pipeline of three neural networks extracts human positions, face detections, and face projections from a single image. Its functionality is depicted in Figure 2.

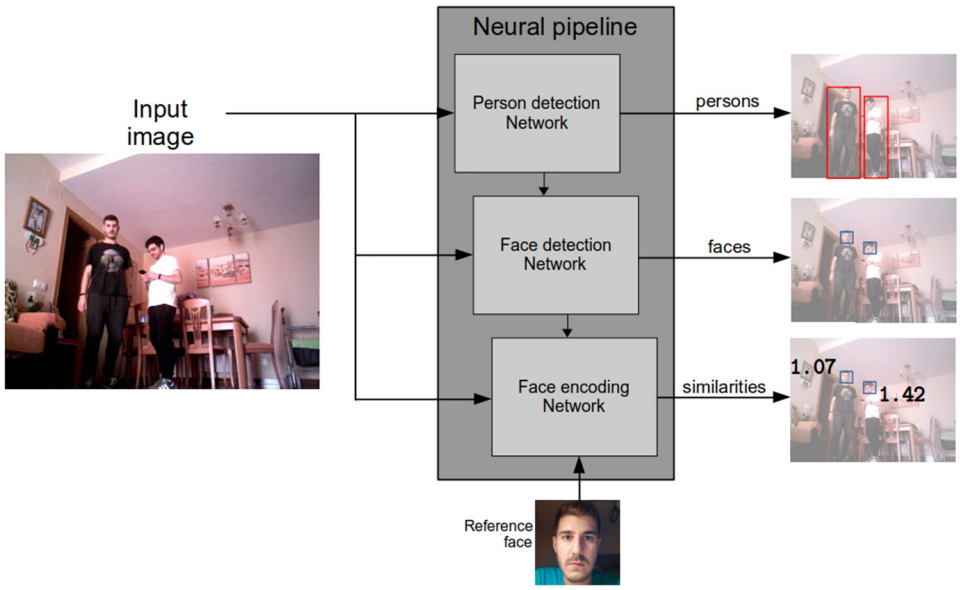


Figure 2. Pipeline of the three neural networks used to detect persons, faces, and similarities with the reference face.

Once the inference pipeline has been designed and implemented, it can take advantage of the optimization libraries of the Jetson TX2 board, using the TensorRT library for this purpose. In this way, several segments from the architecture of a given network can be modified according to specific parameters:

- MSS (Minimum Segment Size): A segment is chosen to be replaced by the TensorRT optimization if its value rises beyond a certain threshold. Increasing this value makes the optimizer more discriminating and only optimizes the network's most heavily loaded portions. Low values can result in an abnormally high overhead, which would lead to inferior performance than if the original graph had been used;
- MCE (Maximum Cached Engines): TensorRT preserves a runtime cache of its engines to speed up loading them into the GPU. As there is relatively little memory available to create the cache, this parameter modifies the number of engines cached;
- Precision mode: The trained neural networks' weights and parameters are typically treated as 64-bit floating point values. The operations are substantially lighter when the precision is decreased to 32-bit or 16-bit, achieving comparable results. A more extreme method lowers the accuracy to 8-bit integers while conducting an additional quantization step because the range can only hold 256 entries. The quantization step analyses the segment, computing the numeric range of its weights.

Experimental tuning of these parameters, looking for an optimization of the inference time, needs to consider that the enhanced models of the three neural networks have to

share the limited available memory onboard. Thus, special attention has to be paid to the memory footprint that an excessive runtime optimization might cause, as it would lead to a substantial penalization if the system cache were utilized to store the models.

3.2. Actuation Module

This module consists of the optical motion tracker and the Proportional-Integral-Derivative (PID) controllers (see Figure 1).

3.2.1. Optical Motion Tracker

The neural pipeline in the perception module outputs reliable inferences with a particular refresh rate, namely k frames. The system could experience a significant delay when the movement is executed if k is set to a too high value. Moving erratically, as a result, could increase the likelihood of losing the reference person. An optical motion tracker component is included in the system to prevent this. Its role is to estimate the person's movement during the previous k frames while the neural pipeline works on the subsequent detection.

In this fashion, the currently detected persons can be tracked along the image. At the same time, they wander until the neural pipeline outputs the latest predictions, which determine the actual new position of the persons. To fulfill this requirement, the tracking method has to be able to run at a higher rate than k, preferably with a considerably lower inference time. This way, the system counts on a slow, reliable detection system backed up by a fast-tracking system devoted to guessing the movements between detections.

A Lucas–Kanade visual tracker [12] has been selected for this task. This method uses a differential approach and calculates the motion field between the images captured at two separate time instants [41]. This algorithm relies on the fact that in a video sequence, the intensity remains almost constant within a specific pixel neighborhood for small changes in space and time.

In the case of this work, the objective is not to compute the entire optical flow (it would be an unnecessary consumption of scarce computational resources). The estimation can be limited to the pixels inside and surrounding the persons in the scene. Furthermore, given its texture, one can notice the existence of more informative regions inside the person than others. Object corners will usually be the best choice to be tracked, given their ease of identification and the fact that they provide more motion information than other areas (aperture problem).

To detect these corners, a Harris corner detector can be used. A corner response can be computed, yielding a score depending on the eigenvalues and their ratio. A modification of this algorithm, known as the Shi–Tomasi corner detector [13], improves the performance of the corner detector by changing the corner response computation. One advantage of this methodology is its invariance to rotation, as it works using the eigenvalues that automatically align to the highest variation directions. However, one important thing to mention as a flaw is the variance to scale: the relative size of the corner concerning the window size influences the eigenvalues.

The Harris/Shi–Tomasi approach yields a reliable result while taking a low time to execute. Combining these two methods provides a fast methodology to estimate the movement of a region using exclusively algebraic calculations on the pixel intensities. As these computations are bounded in complexity, the iteration time is around five times faster than the neural pipeline. Thus, the simultaneous combination of both algorithms allows tracking the movements of the persons during k frames until the next neural update arrives.

As the OpenCV implementation of Lucas–Kanade identifies the points found in both frames, the average displacement of all the points can be computed. This allows the shift of the bounding box of that person using the computed displacement vector. Additionally, it can be rescaled if the person moves closer or further from the camera, using the distribution of the points in the previous and current frames. As shown in Figure 3, the Shi–Tomasi

corner detector finds a set of corners (keypoints) in the frame t. These points are distributed with a given mean: the centroid of the cloud, represented with an "x", besides a standard deviation pair (σ^t_x, σ^t_y). In the next frame, some new keypoints are found (yellow), whereas other keypoints from the previous frame are successfully identified (green). These points are helpful in computing the new centroid and deviations pair (σ^{t+1}_x, σ^{t+1}_y). The remaining points from t (red) are not used since they cannot be located on t + 1. With this information, the person box can be updated accordingly.

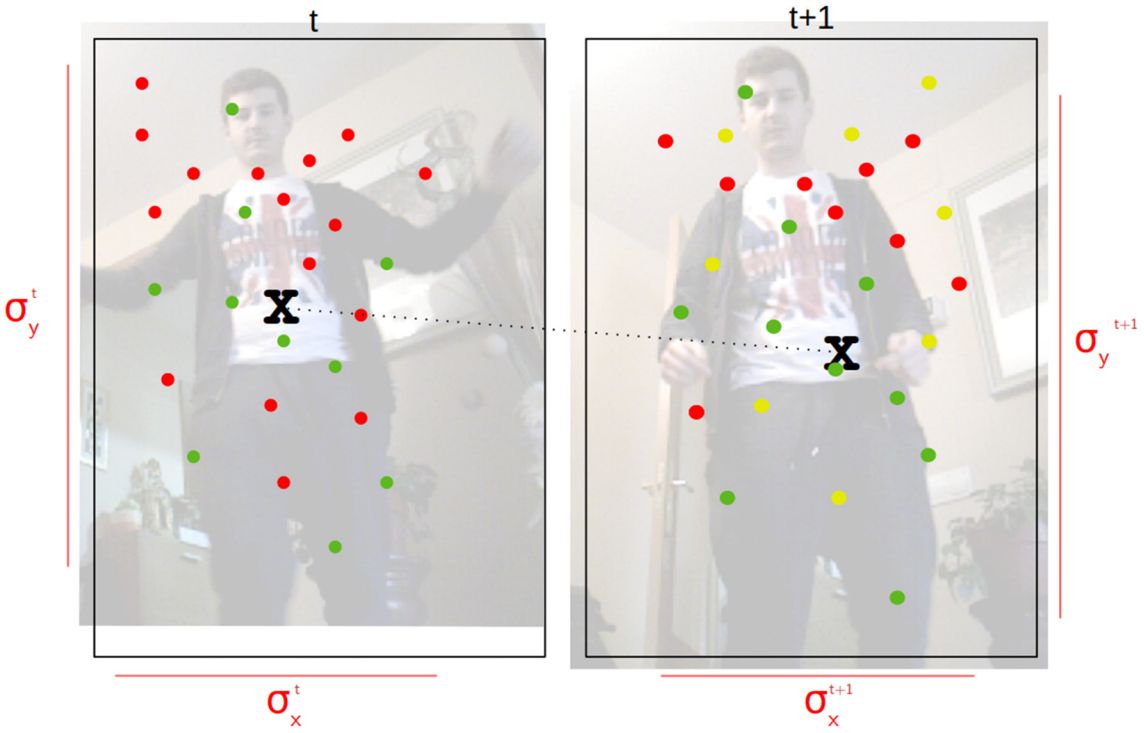


Figure 3. Update of the Lucas–Kanade tracker from frame t to frame t + 1. The green points are correctly detected in both frames, while the red and yellow points are only detected in t and t + 1, respectively. The green points determine the new centroid and the size deformation of the box.

Incorporating this motion tracker enhances the system's robustness since the system's output will not depend only on neural detections. This improves the performance as partial occlusions might cause some detections to be discarded momentarily. The introduction of the tracker can alleviate this effect, as the person will be kept as detected for several frames even if the neural pipeline does not detect it, and its position will be tracked during several frames P.

This number of frames introduces a hysteresis in the tracker, as a person has to be lost for P frames in a row to be discarded. In the same way, detection has to be maintained during P frames to be joined with the tracked persons. This element is introduced in pursuit of stability in complicated scenarios. In such cases, a detection flickering is observable, which could lead to the robot's erratic movement.

3.2.2. PID Controllers

The PID Controllers are responsible for converting the reference person's position data into velocity instructions. The strategy put into practice includes a reactive controller that moves the robot, intending to preserve the person inside safe areas. As the robot offers two degrees of freedom (rotation speed and linear speed), these safe areas fit the following description:

- Angular zone: The reference person has to be placed at the horizontal center of the image, with a margin of ± 50 pixels on the sides;

- Linear zone: The reference person has to be placed at a distance of 1 m from the robot front, with a distance margin of ±30 cm.

  These regions, which can be configured in the system, can be visualized in Figure 4.

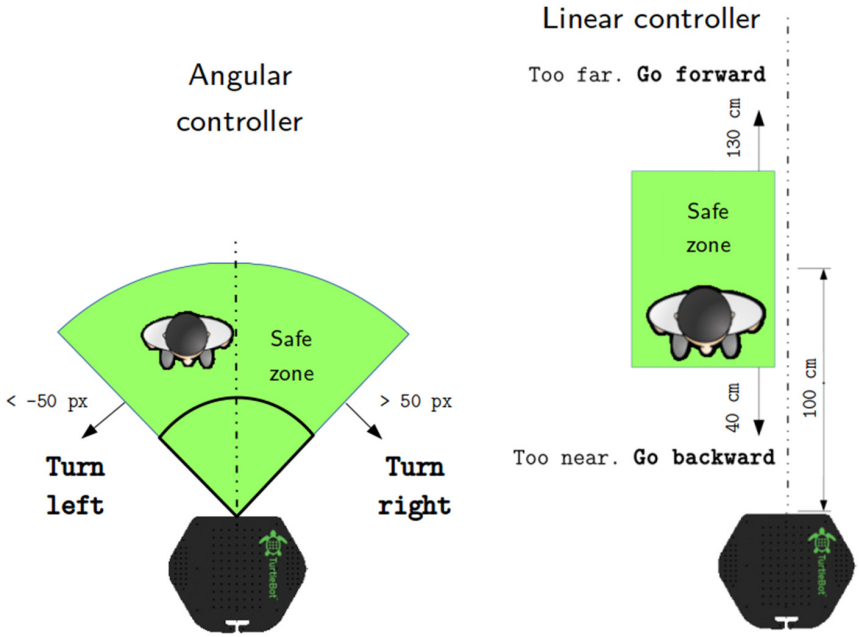


Figure 4. Safe zones for each controller.

The robot must move in specific directions to place the human inside these secure zones. Using the tracked person's coordinates, an error vector (e_x, e_w) is constructed to determine a movement (see Figure 5):

- e_x: The linear error is computed using the depth image, estimating the distance from the robot to the person. Since the camera sensor registers the depth image into the RGB one, the person coordinates can be used in the depth image to find the distance of each pixel inside the bounding box of the reference person: the person depth map. As it is feasible that the box contains an important region of the background (especially if the person opens their arms, as the neural detection will encompass the entire body), the edges of the depth map are trimmed. Later, a 10×10 grid is computed to have 100 uniformly distributed samples of the person's depth. In order to ensure that the background does not affect the range measurement, the median value is computed, as even if some outlier points belonged to the background, they would have to make up 50% of the sampled set to deviate the measurement from the true range;
- e_w: The angular error can be computed, taking into account that, if the robot and the person are aligned, its bounding box will be horizontally placed near the center of the image. Therefore, an error metric can be extracted by computing the difference in the horizontal coordinate between the image center and the center of the bounding box of the reference person.

The last step of the controller takes care of computing two proper responses (linear and angular) for the robot, dependent on the relative distance between the person and the safe zones. If these responses relied only on the error readouts, the robot might receive unsteady commands, which might cause a total loss of the person from the field of view. This can be solved by introducing a PID controller [42] and establishing a closed-loop control system that outputs a response taking into account the previously sent responses, alleviating overshooting and inertial behavior.

After extensive testing under different scenarios, the three PID parameters (k_p, k_i, k_d) were tuned to deliver a soft response to the robot's movements. Visual assessments of the robot's stability under different combinations lead to the values presented in Table 1, which

yielded a steady robot behavior when following a wandering person under typical indoor conditions. These parameters are configurable in the final system.

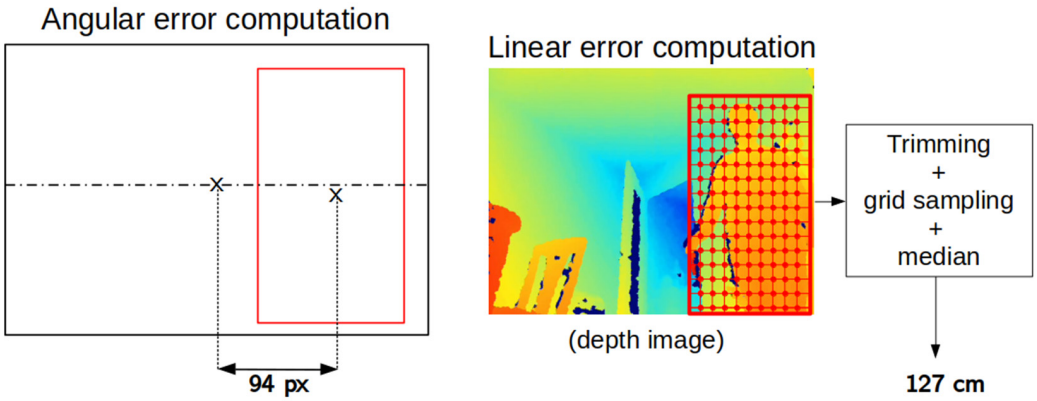


Figure 5. Error computation on each controller. Reference person's bounding box is marked in red.

Table 1. Estimated best values for the parameters in the PID controller.

	Linear	Angular
k_p	0.4	0.005
k_i	0.05	0.006
k_d	0.04	0.0003

3.3. Software Architecture

The software developed feeds the tracker and the neural pipeline with images from the camera and sends the velocity commands to the robot. The software has been implemented on the Jetson board using the Python programming language. As the tracking module has to run asynchronously, the Python threading library is used, deploying the following threads:

1. Main: The purpose of this thread is to continuously draw the output image, compute the errors and suitable responses, and send them to the robot. One thing to notice about this thread is that it does not process all the frames in the sequence, as its rate depends on the drawing time and the computation time of the response. It works asynchronously, fetching the latest frame from the tracker thread;
2. Network controller: this thread handles the three neural networks of the pipeline, running sequential inferences on them. These neural networks are deployed in the GPU of the Jetson board. Therefore, this thread can be seen as the one that interacts with the GPU to pass, retrieve, and transform the tensors from the networks;
3. Tracker: This thread must inherently iterate faster than the neural infrastructure. However, including it in the main thread would be bad for its performance, as the speed would be limited by the image drawing and responses published in the speed topics. Therefore, it is extracted to a specific thread. The simplicity of the Lucas–Kanade tracker makes it fast to execute. However, it would be pointless to track a person several times before a new image arrives from the camera. To avoid this, the thread has a rate limitation of 30 Hz, equal to the frame rate of the camera sensor. As this is the fastest thread to execute, and the tracker must have access to every image from the camera, this is the first component to receive the images from the source in a 30 Hz synchronous manner (the rest of the components can fetch the images asynchronously from the tracker whenever they need them);
4. ROSCam: This component, responsible for fetching the images from the source (an ROSBag or the Xtion camera), is not explicitly deployed as a thread. However, as it works through subscribers when a synchronous mode is required, the ROS API for Python (rospy) automatically deploys these subscribers on independent threads.

This software architecture is depicted in Figure 6, where the interactions among the different threads can be visualized.

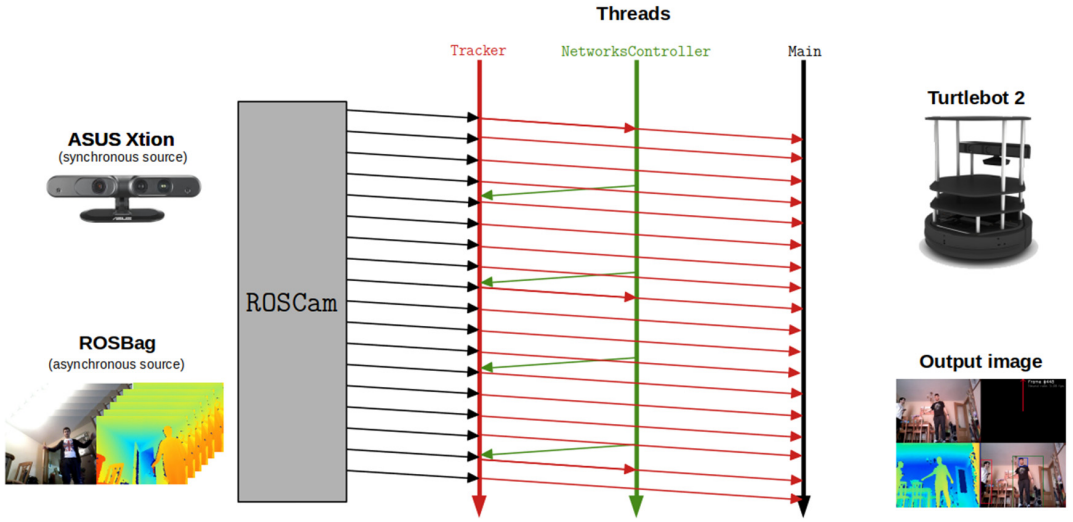


Figure 6. Software architecture of the system.

The image shown in Figure 7 is the visible output of the system. This image is drawn by the main thread when the position errors are computed and the responses have been sent to the robot. It serves to monitor the execution, showing the images, the tracked persons, and the sent commands. A system testing mode exists where these images can be appended to an output video, serving for posterior visualization or assessment of performance.

Figure 7. Output image as drawn by the program. Upper left: input RGB image. Bottom left: input depth image. Upper right: velocity commands sent to the robot and information about the neural rate and current frame number. Bottom right: tracked persons (green if it is the reference, red otherwise) and their faces (blue if it is the reference).

4. Results and Discussion

This section describes the different experiments and benchmarks applied to the proposed system. These tests aim to make design or implementation decisions and select the best choices for improving the performance, accuracy, and robustness of the final system and subsystems. For this purpose, several video sequences were recorded with the camera inside ROSBag files to use the same video input to assess the performance of different configurations, ensuring that the results will not be affected by external variability due to other environmental conditions on the test data.

Most of the tests described below for the neural pipeline measure the IoU score, determining the overlapping quality between two bounding boxes. Thus, it is required to label the video sequences, specifying on each frame the location of the ground truth labels for every video. For this purpose, the tool LabelMe [43] was used to provide the labels to the video, creating a JSON file for each frame of the video sequence.

4.1. Person Detection Experiment

This experiment compares the two detection architectures, YOLO and SSD.

In the case of YOLO, the implemented architecture is YOLOv3 (tiny version). This is due to the memory constraints of the Jetson board where the models are loaded. The available memory (8 GB) has to be shared among TensorFlow and the rest of the processes, causing more memory-intensive models to fail on loading. The YOLOv3 full model demands too much memory, making using it properly on the Jetson TX2 board impossible.

On the other hand, in a real-time application, the most convenient variant of the SSD-based detectors is the one that uses a MobileNet as a feature extraction network. The TensorFlow Model Zoo offers several pre-trained models implementing this network, along which a selection has been carried out (as it will be described in other tests). The chosen model integrates a MobileNetv1, the weights of which have been quantized [44] to reduce the computational cost without reducing the accuracy.

A specific test has been developed to measure these designs' various accuracy vs. inference time trade-offs. A person may be seen moving across the camera's field of vision in a specific film sequence that is 721 frames long. As shown in Figure 8, the humans are identified for each frame of the series using YOLO and SSD, respectively, and the IoU with the ground-truth labels, as well as the inference time, has been measured. On the detections, some gaps correlate to frames where the subject was hidden from the camera's view.

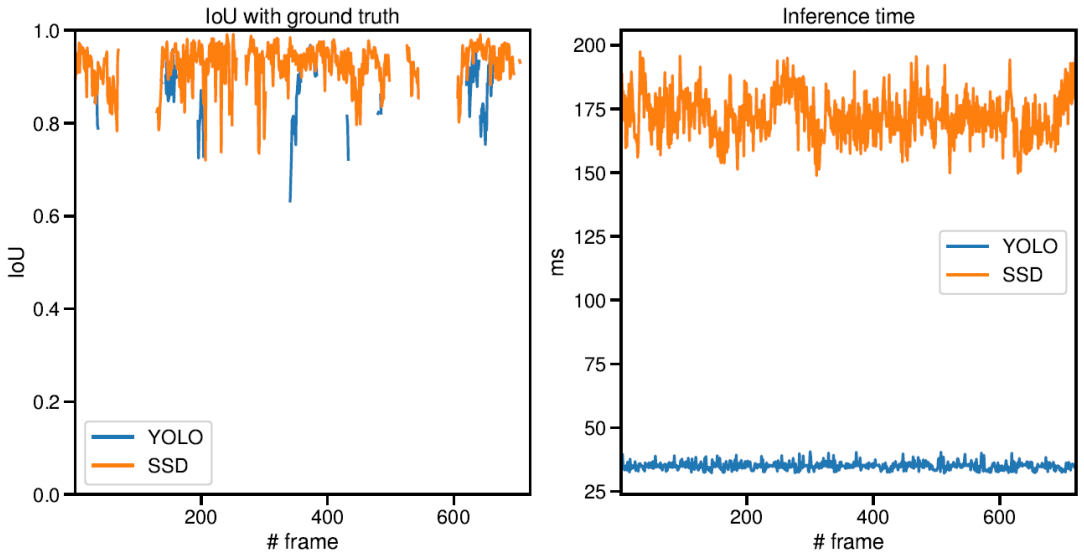


Figure 8. Results of the person detection test: IoU score with ground truth (**left**) and inference time per frame (**right**). A discontinuity represents an absence of detections.

The YOLO-based detector offers a slightly lower average IoU than the SSD-based one (0.858 and 0.926, respectively) while taking five times less average time to make inferences (35 ms vs. 172 ms). In these terms, the YOLO-based detector seems much more efficient.

However, as shown in Table 2, there exists a volatile detection in the YOLO case, being able to detect the person only in 17% of the frames. In contrast, SSD detects the person successfully in 74% of the cases (these percentages are a relative measurement for comparing both models since not all the frames in the sequence contained a person). As there are several frames where the person is not seen, SSD is successful in almost all cases.

Table 2. Numeric summary (average ± standard deviation) for the person detection experiment.

	YOLO	SSD
IoU	0.858 ± 0.068	0.926 ± 0.044
Inference time (ms.)	35.003 ± 1.503	172.237 ± 8.791
Frames with detection	123 (17.06%)	533 (73.93%)

These numeric results show that the YOLO detector is too dependent on pose and lighting conditions for the detections to be successful. On the other hand, the SSD detector yields steady predictions, only cutting on the periods where the person was indeed out of the field of view.

Although SSD presents higher inference times than YOLO, as the system includes the described optical tracker, the YOLO detector can be discarded in favor of the SSD-based one, given that the YOLO version has a much lower detection rate, and the motion tracker cannot palliate this.

4.2. Face Detection Experiment

This experiment is devoted to comparing the performance of the Haar cascade classifier and the neural face detection (faced) using the same video sequence as the previous experiment. The faces are extracted from each sequence frame using one of the methods outlined, and the IoU score is calculated using the ground truth face bounding box. Figure 9 provides a visualization of the outcome.

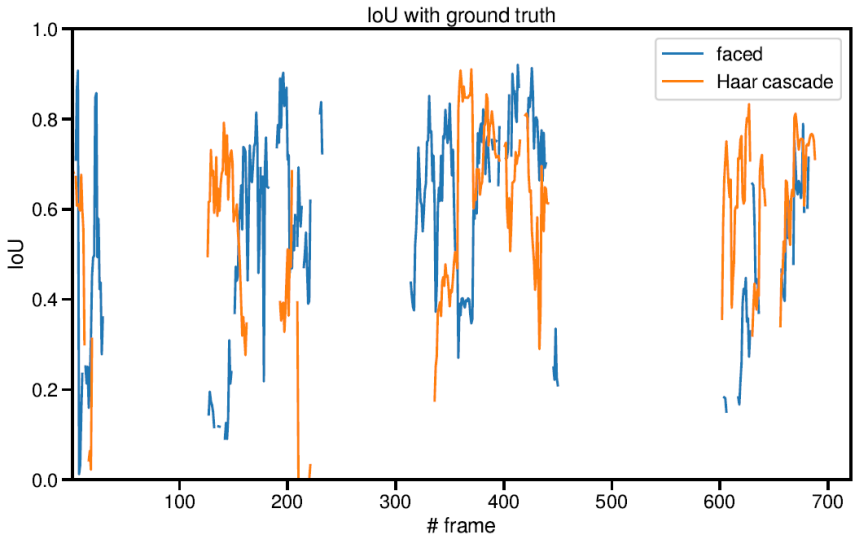


Figure 9. IoU score with the ground truth for each one of the face detection systems.

The detection scores for the two approaches on the same video sequence are shown in Table 3. Both methods provide comparable IoU values, which simultaneously decrease as the subject moves away from the camera. However, the faced implementation, which predicts face positions using deep learning, can maintain a positive IoU at various points when the Haar performance is zero.

Table 3. Numeric summary (average ± standard deviation) for the face detection experiment.

	Haar	**Faced**
IoU	0.579 ± 0.202	0.559 ± 0.221
Frames with detection	248 (34.4%)	266 (36.89%)

The fundamental limitation of the Haar cascade classifier is that it can only detect frontal faces, resulting in performance degradation anytime a person turns their face to the side. Position variations of the individual cause this. The average IoU for both approaches is similar, but the deep learning strategy (faced) recognizes a face in 36.89% of the frames, compared to the Haar cascade's lower rate of 34.40%. As a result, this test verifies the improvement in face detection performance when utilizing a particular neural network specifically trained for that task.

4.3. Face Recognition Experiment

A facial recognition neural network, the final part of the neural pipeline, is used to certify the identity of the reference individual. Given that its location is tracked, this is important for determining whether that individual needs to be followed even if they turn around later. The FaceNet network, which projects a face into a 128-dimensional space, is the foundation of this subsystem. The proposed system uses these projections to identify whether the input face belongs to the reference person by comparing the input face's Euclidean distance to the projection of a reference face.

This experiment is designed to assess the quality of the projection system, which should yield far points for a different face and near points for a matching face. For this proposal, a video sequence was recorded containing two persons wandering in front of the robot. The faces of each frame are labeled, separating the faces of the two persons in two different classes. For computing the distance, a reference face was set using an image, and the distance to the reference face of each one of the faces in the video was stored. The result can be observed in Figure 10.

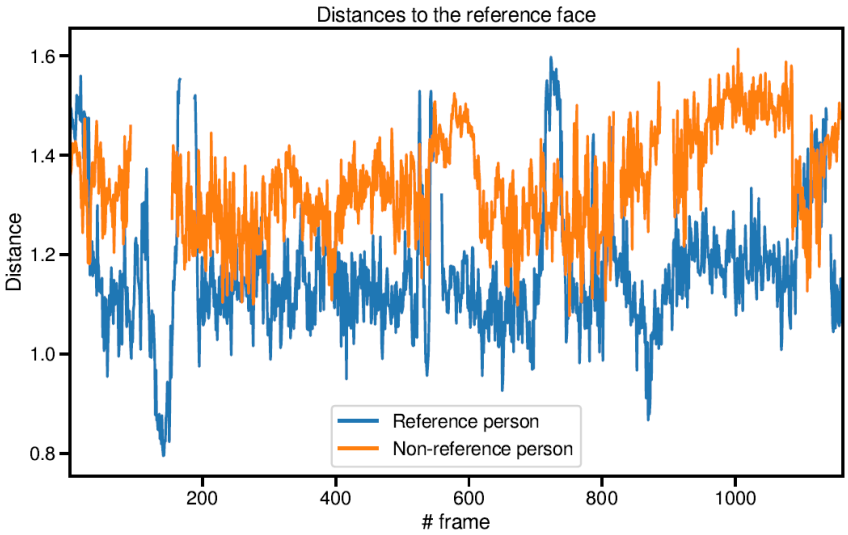


Figure 10. Distance of each face to the projection of a reference face in the face recognition experiment.

Two inferences about the caliber of face projections can be drawn from the results. First, the reference person's encodings exhibit remarkable stability overall. The obtained projections for each frame are typically positioned at a distance of around 1:16. (threshold chosen for accepting a person as the reference one). Occlusions and variations in the face's position cause exceptional elevations in distance by degrading the projection's quality. Second, there is a higher overall gap (1:34) between the encodings of a person different

from the reference person and the reference face. This makes identifying a face as the reference easy and prevents false positives.

This allows for the conclusion of the correct performance of FaceNet training, yielding an efficient separation between the encodings of different persons, as well as close encodings for faces belonging to the same person, making this system a robust approach to performing person recognition tasks since the distance of a projection to the reference face has to be below the threshold for being labeled as the reference face.

4.4. TensorRT Optimization Experiment

The TensorRT engine is used to optimize the implementation of a neural network on an NVIDIA-compatible GPU, using a binding component between TensorFlow network graphs and TensorRT itself.

There are several tunable parameters for customizing the implementation, being MSS, MCE, and precision mode the most relevant ones. As varying these parameters changes the model size and the inference time, an experiment has been conducted to test the inference time of each model. The optimization script performs a grid search between a set of values for each parameter (MSS, MCE, and precision mode) and tests the performance on a specific ROSBag sequence.

Table 4 shows the values of each parameter yielding the fastest inference times for the SSD-based model and the Tiny YOLOv3.

Table 4. Values of tunable parameters yielding the fastest inference times for YOLOv3 and SSD-based models.

	YOLOv3	**SSD**
Precision	FP16	FP16
MCE	50	3
MSS	3	3
Inference time (ms)	39.768	15.922

This experiment aims to quantify the loss of precision when the SSD model is optimized by TensorRT using the FP16 precision model, which is the fastest mode to infer, as shown in Table 4. To do so, the test video sequence is used again, passing each frame forward to the standard neural network and storing the detected persons. Later, the same video sequence is passed through the TensorRT version of the same graph, storing the detections of each person as well. When both passes are performed, the IoU score is computed on each frame between the standard inferences (considered as ground truth labels) and the TensorRT inferences. This IoU score on each frame, along with the inference times for each network model, can be seen in Figure 11.

Figure 11 shows the contrast between an optimized graph and a standard one. On the one hand, this accuracy loss is minimal, yet there are certain apparent cases when the performance is lost by more than 5%. The inference time difference, however, is also discernible. The improvement is more evident since, in addition to improved stability on the inference time, the TensorRT optimized model makes inferences three times quicker than the original graph (56.769 ms vs. 184.477 ms).

Considering these findings, TensorRT optimizations are a valuable tool with which to significantly improve system performance, enabling the slower component (the neural pipeline) to experiment a significant decrease in inference time. As trustworthy neural updates are made more often, the total performance is considerably enhanced.

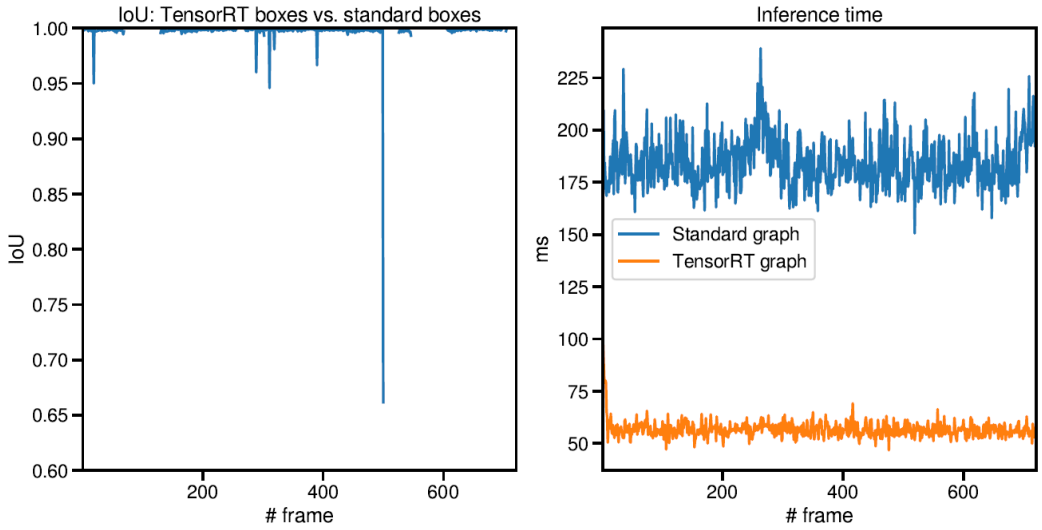


Figure 11. IoU between the standard and TensorRT graph inferences (**left**) and inference times for both networks (**right**). The IoU graph has been re-scaled between 0.6 and 1 to better visualize the IoU variability.

4.5. Motion Tracker Experiment

The Lucas–Kanade tracker aims to follow the person's movements between two consecutive inferences from the neural pipeline. In embedded systems, these inferences might take a long time, so interpolation of the detections using optical flow can be crucial for avoiding a loss of the person's location, especially if a partial occlusion of the person causes the network not to detect them for a while.

This experiment aims to identify the conditions under which a tracker can palliate these drawbacks of the neural detection pipeline, depending on the parameter k. This parameter modulates the number of elapsed frames between two consecutive neural detections, taking a higher value if the inferences take longer to be computed by the neural pipeline.

On the test, a specific test sequence was recorded and labeled. A hanging blanket was introduced to partially occlude the person, making the network lose the detections. A correctly tuned tracker keeps the detection active and updates the bounding box for several frames (determined by the P parameter). The video sequence was evaluated using k = 10 and k = 20, checking the influence of the tracker in the IoU with the ground truth labels of the sequence. The result for both values of k can be observed in Figure 12, where the lapse corresponding to the person occlusion has been emphasized.

The results show the IoU score between the persons and the ground truth labels on the test sequence. Regardless of the value of k (the number of frames elapsed between neural detections), a similar performance can be expected under standard conditions (the faded portion of the graph). However, the emphasized region corresponds to an occlusion behind a hanging blanket and is zoomed in on the bottom plots. On this lapse, better performance is perceptible, especially when the inference time of the neural pipeline is higher (k = 20). The hanging blanket occludes the person, causing the neural network to stop detecting them. However, as the tracker retains the detection for several updates because of the P parameter, the person is not lost until several frames later.

Additionally, the Lucas–Kanade algorithm allows for the determination of the displacement of the person even when the neural pipeline is not detecting it. This explains the higher IoU when the tracker is active due to the bounding box shifting when computed, confirming the improvement in the performance when using the tracker. Outside this region (top plots), some regions can be detected, such as the ending lapse of the sequence for k = 20. This is probably due to non-optimal values for the parameters of the tracker, which do not correctly shift the bounding box towards the actual direction of movement

of the person. Proper in-depth tuning of the parameters can potentially fix this lower performance for situations like that.

While the neural pipeline runs on the GPU of the board, the Lucas–Kanade tracker, whose calculations are much lighter, runs on the CPU. This separation allows the combination of both systems asynchronously without affecting the overall load of the system.

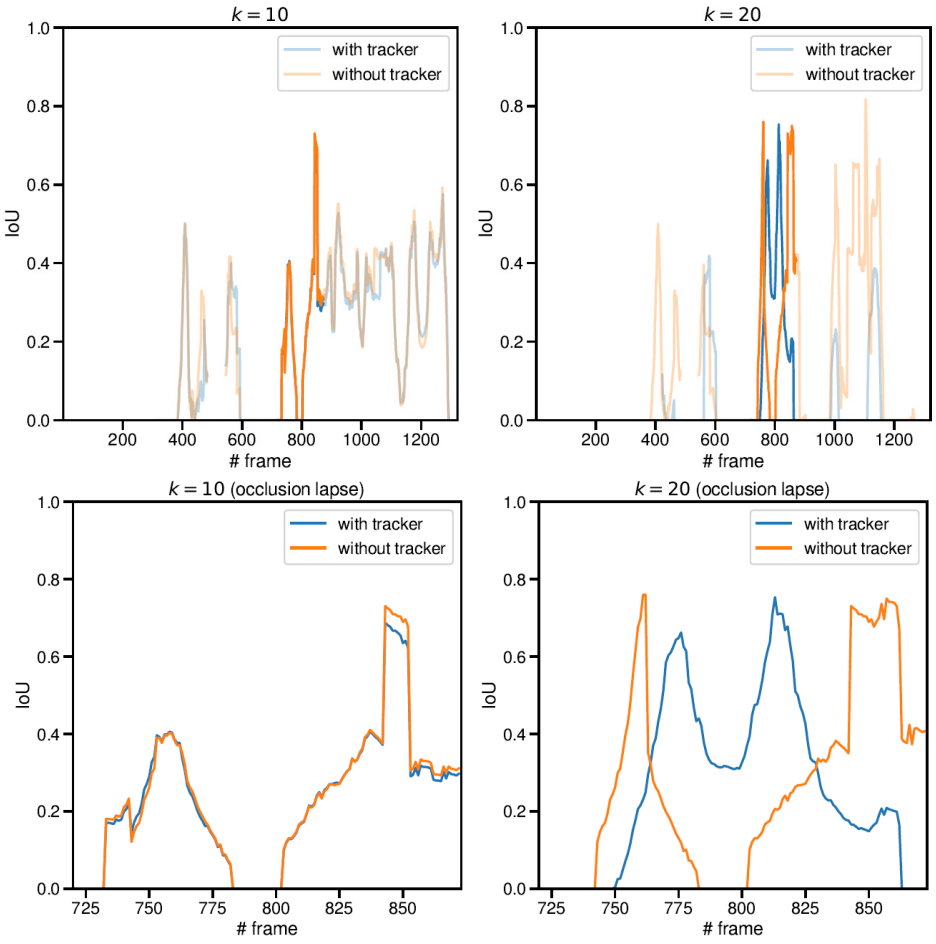


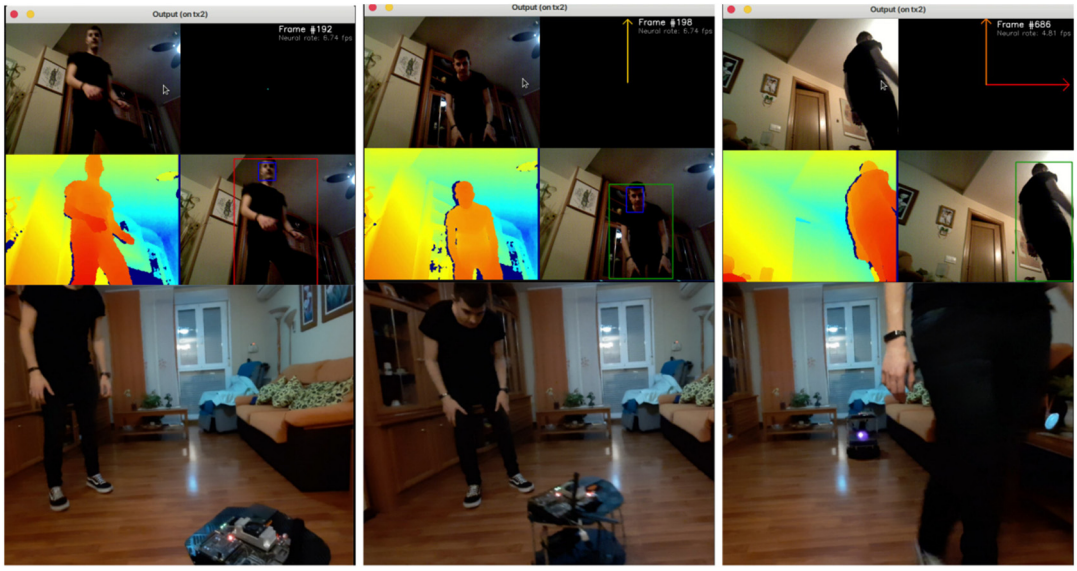
Figure 12. Results of the motion tracker test, for k = 10 (**left**) and k = 20 (**right**). The lapse corresponding to the person occlusion has been emphasized and zoomed in on the bottom graphs.

4.6. Complete System Experiment

A video showing the complete system in action can be seen at the following link: https://www.youtube.com/watch?v=WZ0riKMwJWA (accessed on 25 October 2023). This video illustrates a scene in which the reference person walks into the robot's range of vision and faces the camera. The individual is recognized using the observed face after a few consecutive frames of the person being spotted. The robot begins to follow the human once the projected face is near enough to the reference one.

Even if the individual's face is no longer visible, as the algorithm has already verified that it is the correct person, the linear and angular errors are still calculated for each frame. A velocity instruction is computed and issued to the robot if the errors exceed acceptable ranges. This procedure is repeated until the individual becomes lost and the robot stops.

Figure 13 depicts the robot's intended behavior when following a person correctly. The top portion of the photographs displays some screen captures of the program output, including the RGB-D images, the movement orders, and the monitored individuals. The scene from a cell phone is displayed in the bottom portion of the screen, allowing viewers to see the performance from outside.



(**a**) (**b**) (**c**)

Figure 13. Screen captures of the complete system test. (**a**) Person detection. (**b**) Person recognition and following. (**c**) Following without facial feedback.

4.7. Discussion of Results

As demonstrated in the complete system experiment, the system proposed in this work can successfully identify and track a single individual in a robust manner. As opposed to other works, this system is based on a hardware/software architecture that combines multithreaded software with commercial off-the-shelf components, combining the performance of an optical tracker with a CNN pipeline in an embedded SoM in order to minimize the system's cost and power consumption.

Regarding system limitations, we have performed several tests in crowded environments, and the results yield a notable performance degradation due to the limited computational resources of the platform. On the other hand, CNN-based person identification has been proven to be sufficiently discriminant in all the tests performed.

The use of the SoM has imposed additional constraints on the system. The pipeline of the three CNNs has been space-optimized to fit in the board. In addition, due to the high computational load needed, the iteration loop of the system has been established at 10 Hz. The responsiveness of the system could be improved if additional hardware resources were incorporated into the system.

5. Conclusions

The application of deep learning techniques in robotics is rapidly increasing the complexity of behaviors that a robot may implement, due to the vast range of perceptive abilities that a deep learning system can achieve. The combination of artificial intelligence and robotics is pushing the boundaries of the expectations for future robot accomplishments.

This work presents an embedded mobile system that robustly identifies and follows a specific reference person, relying on the reliability of deep learning for being capable of working in real environments. The system is based on commercial-off-the-shelf (COTS) hardware elements: an affordable educational robot incorporating an NVIDIA Jetson TX2 SoM board and an RGB-D sensor. A multi-threaded software application allows for the achievement of the requirements needed in this kind of system, keeping the economic cost of the system as low as possible. It will undoubtedly be quite advantageous for social robots to incorporate this technology.

The detection and recognition pipeline has been exclusively designed using deep neural networks, ensuring a robust performance in non-controlled environments. The neural pipeline has been complemented by an optical tracking component, improving the

performance under certain circumstances, such as partial occlusions or high inference times (for example, due to power restrictions).

This robustness is crucial, mainly because the camera is located at a very low position: the lens has some vertical inclination to see the whole body of the persons in front of the robot. However, this causes an excessive amount of light from ceiling lamps to reach the camera, dimming the persons in the image. Systems that do not incorporate the combination of deep learning and optical tracking tend to fail under these conditions.

Finally, we mention some further improvements that can be addressed as future lines of work:

1. Implement multimodal tracking using sensor fusion. The depth data of the person also provides positional information, and bringing this information into the tracker can potentially lead to better performance;
2. Implement a probabilistic tracker, such as an EKF (Extended Kalman Filter), relying on the person's trajectory. This approach may avoid confusion between two persons if they cross each other or help the system follow a person's trajectory even if it is temporarily lost. In addition, this can solve problems coming from using optical flow, such as a person moving a part of their body;
3. Add a navigation component to the robot. If the robot is equipped with a laser scanner, it can detect possible obstacles between the robot and the person. Thus, a simple planning algorithm, such as VFF (Virtual Force Field), can be combined with this system to avoid collisions while the robot moves.
4. Incorporate more complex identification networks into the system for better adaptation to changes in a person's appearance/behavior.

Author Contributions: Conceptualization, I.C., E.P. and J.M.C.; methodology, I.C., E.P. and J.M.C.; software, I.C. and E.P.; validation, I.C., E.P., J.F.-C. and J.M.C.; formal analysis, J.F.-C. and J.M.C.; investigation, I.C., J.F.-C., E.P. and J.M.C.; resources, J.M.C.; writing—original draft preparation, I.C and J.F.-C.; writing—review and editing, J.F.-C. and J.M.C.; visualization, I.C. and E.P.; supervision, J.F.-C. and J.M.C.; project administration, J.M.C.; funding acquisition, J.M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Spanish Ministry of Science and Innovation, in the framework of the research project "Plataforma web educativa abierta para la programación de robots en ingeniería (UNIBOTICS-GAM)", Proyectos de Transición Ecológica y Transición Digital 2021, Reference TED2021-132632B-I00 (2022–2024).

Data Availability Statement: The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
2. Deng, L.; Hinton, G.; Kingsbury, B. New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 8599–8603.
3. Krizhevsky, A.; Sutskever, I.; Hinton, G. Imagenet classification with deep convolutional neural networks. *Neural Inf. Process. Syst.* **2012**, *25*, 84–90. [CrossRef]
4. Martínez-Olmos, P. *Deep Learning Course: Convolutional Neural Networks*; University Lecture; Springer: Berlin, Germany, 2020.
5. Potel, J. Trial by Fire: Teleoperated Robot Targets Chernobyl. In *Proceedings of the IEEE Computer Graphics and Applications*; IEEE: Piscataway, NJ, USA, 1998; Volume 18, pp. 10–14. [CrossRef]
6. Berkelman, P.; Ma, J. The University of Hawaii Teleoperated Robotic Surgery System. In Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, USA, 29 October–2 November 2007; pp. 2565–2566. [CrossRef]
7. Okamura, A.M. Methods for haptic feedback in teleoperated robot-assisted surgery. *Ind. Robot. Int. J.* **2004**, *31*, 499–508. [CrossRef]
8. Girimonte, D.; Izzo, D. Artificial Intelligence for Space Applications. In *Proceedings of the Intelligent Computing Everywhere*; Schuster, A.J., Ed.; Springer: London, UK, 2007; pp. 235–253. [CrossRef]

9. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; Berg, A.C. SSD: Single Shot Multibox Detector. In *Lecture Notes in Computer Science*; Springer: Berlin, Germany, 2016; pp. 21–37. [CrossRef]
10. Itzcovich, I. Faced: CPU Real Time Face Detection Using Deep Learning. Available online: https://towardsdatascience.com/faced-cpu-real-time-face-detection-using-deep-learning-1488681c1602 (accessed on 25 October 2023).
11. Schroff, F.; Kalenichenko, D.; Philbin, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015. [CrossRef]
12. Lucas, B.; Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI). In Proceedings of the JCAI'81: 7th International Joint Conference on Artificial Intelligence, Vancouver, BC, Canada, 24–28 August 1981; Volume 81.
13. Shi, J. Good Features to Track. In Proceedings of the 1994 IEEE Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 21–23 June 1994; pp. 593–600.
14. Gockley, R.; Forlizzi, J.; Simmons, R. Natural Person-Following Behavior for Social Robots. In Proceedings of the HRI '07: Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction, Arlington, VA, USA, 10–12 March 2007; pp. 17–24. [CrossRef]
15. 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, Kauai, HI, USA, 8–14 December 2001; Volume 1, pp. 1–511. [CrossRef]
16. Molina-Moreno, I.M.; González-Díaz, I.; Díaz-de-María, F. Efficient Scale-Adaptive License Plate Detection System. *IEEE Trans. Intell. Transp. Syst.* **2019**, *20*, 2109–2121. [CrossRef]
17. Dalal, N.; Triggs, B. Histograms of Oriented Gradients for Human Detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–25 June 2005; Volume 1, pp. 886–893.
18. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv* **2013**, arXiv:1311.2524.
19. Girshick, R. Fast R-CNN. *arXiv* **2015**, arXiv:1504.08083.
20. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *ECCV 2014: Computer Vision*; Springer: Cham, Switzerland, 2014; Volume 8691. [CrossRef]
21. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
22. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
23. Hosang, J.; Benenson, R.; Schiele, B. Learning non-maximum suppression. *arXiv* **2017**, arXiv:1705.02950.
24. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. *arXiv* **2015**, arXiv:1506.02640.
25. Redmon, J.; Farhadi, A. Yolo9000: Better, faster, stronger. *arXiv* **2016**, arXiv:1612.08242.
26. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
27. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. *arXiv* **2015**, arXiv:1512.03385.
28. Li, P.; Wu, H.; Chen, Q. Color distinctiveness feature for person identification without face information. *Procedia Comput. Sci.* **2015**, *60*, 1809–1816. [CrossRef]
29. Bhattacharyya, A. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.* **1943**, *35*, 99–109.
30. Johnston, B.; Chazal, P. A review of image-based automatic facial landmark identification techniques. *EURASIP J. Image Video Process.* **2018**, *2018*, 86. [CrossRef]
31. Gottumukkal, R.; Asari, V. An improved face recognition technique based on modular PCA approach. *Pattern Recognit. Lett.* **2004**, *25*, 429–436. [CrossRef]
32. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. *arXiv* **2014**, arXiv:1409.4842.
33. Weinberger, K.Q.; Blitzer, J.; Saul, L.K. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.* **2006**, *10*, 207–244.
34. Islam, M.J.; Hong, J.; Sattar, J. Person-following by autonomous robots: A categorical overview. *Int. J. Robot. Res.* **2019**, *38*, 1581–1618. [CrossRef]
35. Islam, M.J.; Fulton, M.; Sattar, J. Towards a generic diver-following algorithm: Balancing robustness and efficiency in deep visual detection. *arXiv* **2018**, arXiv:1809.06849. [CrossRef]
36. Eirale, A.; Martini, M.; Chiaberge, M. Human-Centered Navigation and Person-Following with Omnidirectional Robot for Indoor Assistance and Monitoring. *Robotics* **2022**, *11*, 108. [CrossRef]
37. Ghimire, A.; Zhang, X.; Javed, S.; Dias, J.; Werghi, N. Robot Person Following in Uniform Crowd Environment. *arXiv* **2022**, arXiv:2205.10553.
38. Condés, I.; Cañas, J.M. Person following Robot Behaviour using Deep Learning. In Proceedings of the 19th International Workshop of Physical Agents (WAF 2018), Madrid, Spain, 22–23 November 2018; pp. 147–161.
39. Condés, I.; Cañas, J.M.; Perdices, E. Embedded Deep Learning Solution for Person Identification and following with a Robot. In *WAF 2020: Advances in Physical Agents II*; Springer: Cham, Switzerland, 2020; Volume 1285. [CrossRef]

40. TensorFlow. TensorFlow Object Detection: Model Zoo. Available online: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md (accessed on 25 October 2023).
41. González-Díaz, I.; Díaz-de-María, F. Adaptive multipattern fast block-matching algorithm based on motion classification techniques. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, *18*, 1369–1382. [CrossRef]
42. Åström, K.J.; Murray, R.M. *Feedback Systems: An Introduction for Scientists and Engineers*; Tech. Rep.; Princeton University Press: Princeton, NJ, USA, 2004.
43. Wada, K. labelme: Image Polygonal Annotation with Python. 2016. Available online: https://github.com/wkentaro/labelme (accessed on 25 October 2023).
44. Blog, G.A. Accelerating Training and Inference with the Tensorflow Object Detection API. Available online: https://blog.research.google/2018/07/accelerated-training-and-inference-with.html (accessed on 25 October 2023).