CNN Based Perception System for Collision Avoidance in Mobile Robots using Stereo Vision

Edgar Macias-Garcia, Deysy Galeana-Perez, and Eduardo Bayro-Corrochano, *Senior Member, IEEE*
Department of Electrical Engineering and Computer Science
CINVESTAV, Campus Guadalajara
Guadalajara, Mexico
email: {emacias, dgaleana, edb}@gdl.cinvestav.mx

***Abstract*—In this paper, a CNN based perception system for collision avoidance in mobile robots is presented. In the considered scenario, a mobile robot is ordered to reach a target on its workspace, where several types of objects influence a collision risk for a suitable movement to the desired goal. To ensure collision-free planning through the environment, a set of convolutional neural networks in parallel are employed to detect a set of static or dynamic objects of interest in the environment, as well as objects on the floor that could imply a collision risk during a movement execution. Afterward, stereo vision and filtering algorithms are employed to recover and track the spatial position of detections, in order to generate enough information to plan a collision-free trajectory. All the above steps are evaluated in real-time and real environments, proving to be enough robust and fast for a wide range of mobile robot applications.**

***Index Terms*—Deep learning, Cognitive robotics, Computer vision, Mobile robots**

I. Introduction

Motion planning has become a crucial stage in robotics, where the aim is to enable robots to automatically compute their movements from high-level task descriptions and models acquired through sensing [1]. In this area, obstacle detection is probably the first challenge to face in order to ensure that the robot can navigate safely through its environment. Through artificial vision techniques, stereo cameras could serve as a reliable sensor for mobile navigation; in comparison with other navigation sensors such as radars or lasers, they generate a richer set of features at a lower cost.

In recent years, computer vision techniques have been employed modern camera sensors to acquire a set of features from the environment that allows generating enough spatial information for a collision-free navigation [2], [3]. Despite the variability of applications, depending on the desired activity, these techniques may require intensive engineering efforts, as achieving a good performance could be difficult or even unreachable. For example, recognizing certain types of patterns in uncontrolled environments where variations in illumination, position, occlusion, object morphology, or other noise types complicate the detection task [4]. Thanks to a notorious increase in available storage and processing capacity of the modern computing devices, deep learning or deep neural networks have gained focus attention in many areas from pattern recognition and machine learning [5].

Historically, deep learning has been gaining ground since the Deep Belief Network (DBN) [6] with three hidden layers applied over the MNIST dataset [7] was presented, since then it has served as the basis for other networks such as the Convolutional Neural Networks (CNN). In this topic, there are many efforts to avoid some related problems like overfitting, with algorithms such as Dropout [8], batch normalization [9], and dataset augmentation techniques. The main advantage of the neural networks remains in that after training, a model is capable of real-time inference with a model size no larger than a few hundred megabytes [10], which in most of the cases is faster and more accurate than traditional vision techniques, thanks to its generalization capacity.

Recently, most deep learning works have been focused on the solution of perception related problems, such as road segmentation in autonomous driving [4], obstacle detection [11], and mobile robot navigation [12]. In [13] a convolutional neural network for office door detection was presented by employing real images as dataset, showing good generalization capacity against variations in terms of door's morphology. On the other hand in [14] a hybrid perception system for pedestrian detection and evasion in corridors was proposed, which was tested in real scenarios with moving and stationary pedestrians, showing good performance in real-time operations.

By taking advantage of the generalization capacity of the neural networks, this work proposes a CNN based perception system for collision avoidance in mobile robots. In this scheme, two neural networks in parallel are employed to detect objects on the floor that could imply a collision risk during a movement execution through the environment, as well as a set of potential targets that the robot may want to reach or interact with. Then, stereo vision, filtering, and tracking algorithms are employed to recover the position of each detected object, in order to generate a collision-free trajectory. Compared with other works, the main contributions of this work can be listed as follows:

- The whole detections are developed directly from RGB images, instead to process the information from depth maps, which in turn would result in a better speed performance by employing only those pixels with relevant information.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

- For each object detection a Kalman filter [15] is used to support filtering and tracking algorithms, which allows to generate enough information to calculate a collision-free trajectory for each detection frame.
- The system runs on a Jetson TX1 developer board, with an average processing speed of 15 frames per second (fps).

The rest of the paper is organized as follows: section II describes the problem formulation addressed in this work, as well as a brief introduction of the proposed perception system. Sections III and IV describe the neural network models employed for obstacle and target detection, describing its architecture, training process, and processing scheme. Section V describes the approaches considered to recover the spatial position of each detection, as well as the filtering and tracking algorithms employed before generating a collision-free trajectory. Section VI describes several performance tests applied over each stage of the system, and finally, conclusions and future work are drawn in Section VII.

II. Problem Formulation

In this work, the following situation is considered: a mobile robot is standing in a workspace and there is a target somewhere in the environment that it must reach, but there are several objects along the way that may imply a collision risk if a direct movement is employed. Using its vision, the robot must extract enough information from the environment to plan a trajectory that allows reaching the target while avoiding collisions with the objects in its path (Fig. 1).

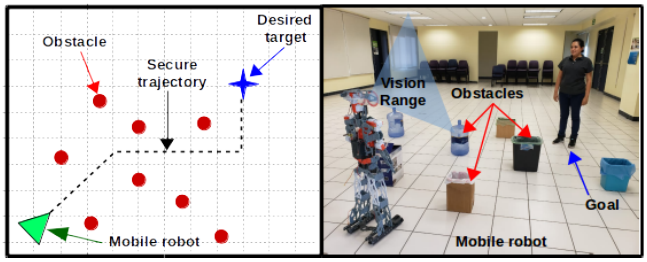

Fig. 1. Representation of the perception problem. A mobile robot must plan a trajectory to reach a target while avoiding collisions with static or moving objects on its path.

The proposed perception system scheme is summarized in Fig. 2:

- First, an RGB image and a depth map are acquired from the environment using a stereo camera, each one of size 480×640 pixels.
- According to the input specifications of later modules, the RGB input image is processed in two ways: a set of stripes or "Stixels" of size $100 \times 10 \times 3$, and a $448 \times 448 \times 3$ resized version of the original image.
- Two parallel neural networks are employed to detect objects of interest in the workspace; a target to reach and obstacles to avoid during a movement execution.
- Using the previous information, stereo vision is employed to recover the position of each detected object.
- A set of tracking algorithms and Kalman filters [15] are employed to classify and filtering the position of each detected object, in order to deal with the noise related to position estimation (due to a wrong depth calculation by hardware, or multiple detections of the same object).
- Once the previous information is available, the RRT algorithm [16] is employed to generate a collision-free trajectory to a desired target.
- Between frames, if the goal changes its position or the previously planned trajectory is crossed by an object, a new trajectory to reach the goal is calculated.

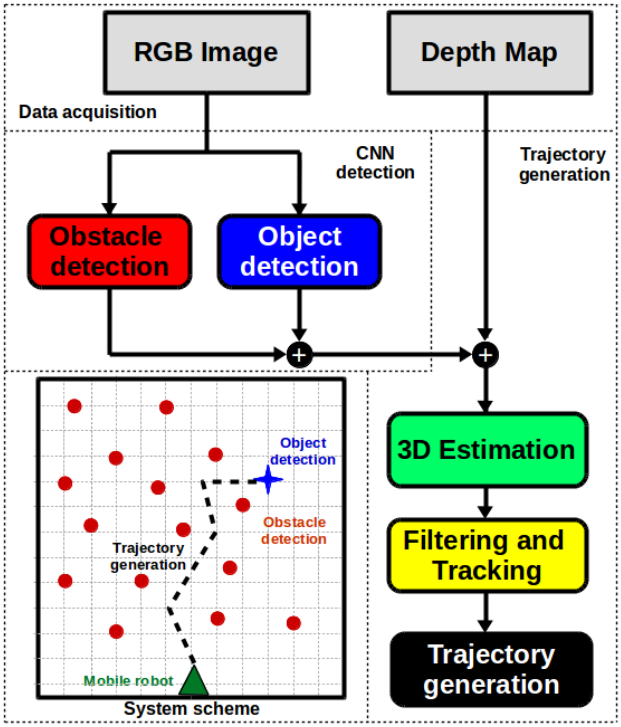

Fig. 2. Perception system scheme. Several modules are employed to solve different perception tasks: obstacle detection, target detection, 3D position estimation, position filtering, tracking scheme and collision-free trajectory generation.

In the scope of this work, the following restrictions are considered:

- It is only considered the situation to reach a target from the robot initial position (camera frame), whereby all the distance measurements are related to this reference frame.
- To obtain the depth map at all times, a ZED stereo camera is employed, which generates real-time depth information of objects at a distance between 0.70 and 20 meters.
- It is assumed that all obstacles are at floor level, so if any are above this, they are taken as if they were on the floor, which is a common situation for indoor environments.
- When multiple targets are available, it is assumed that a single desired goal is always known. Whereby, once it is detected tracking algorithms are employed to follow the changes in its position.

III. Obstacle detection

In the proposed scheme, the objective of this module is detecting a set of potential obstacles that could appear between the robot and its goal. To address this problem, a modified version of the network "StixelNet" proposed in [11] for autonomous driving is employed. Based on the literature, this network offers advantages against other standard architectures as follows:

- It employs a column regression problem scheme, which allows saving computational cost by employing smaller input images, as well as allowing to process multiple columns simultaneously.
- A new semi-discrete loss function termed "PL-Loss" is proposed, which in turn allows having better generalization capacity on the neurons in the last layer, thanks to a redistribution of the output information between neighboring neurons.
- From its original application for autonomous driving, it had the objective of segmenting the free road space in highways. Similarly, it can be employed to segment the free space in a workspace for a mobile robot, by retraining the model with a new specialized dataset.

This model is composed of five layers distributed in two convolutional at the beginning followed by a set of three fully connected layers. The ReLU non-linearity is applied at the output of all layers except the last one, which has a Softmax function, and for the convolutional layers, a Max-pooling layer follows the ReLU operation as well (Fig. 3).

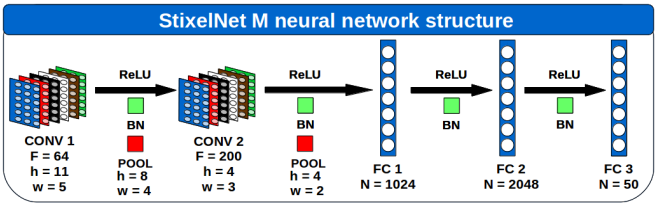

Fig. 3. StixelNet, modified neural network structure. Where F indicates the number of filters, h and w the size of the kernel on convolutional and pooling layers, N the number of neurons in fully-connected layers, and BN the position of batch normalization layers.

In order to adapt the network for indoor applications, the following modifications were considered:

- For obstacles at floor level just the region bounded by the pixel limits $h \in [h_{min}, h_{max}]$ was considered, avoiding to process pixels with no relevant information.
- To adjust the neural network parameters to the work resolution, the size and stride of all the filters were modified, as well as the number of neurons in each layer.
- To improve the training process and to avoid overfitting, batch normalization layers [9] were added after each activation function ReLU on the network, as well as Dropout layers [8] after the first two fully connected layers.
- The final structure is presented in Fig. 3, which is able to detect up to 32 obstacles for each input image.

A. Forward processing

A $480 \times 640 \times 3$ input RGB image is trimmed horizontally over the pixels $[h_{min}, h_{max}]$ (floor level), then it is resized at a half of its dimensions, producing an array of $(\frac{h_{max}-h_{min}}{2}) \times 320 \times 3$ size, which is divided in 32 vertical stripes $V(i)$ of 10 pixels of width, producing a final input array of $32 \times (\frac{h_{max}-h_{min}}{2}) \times 10 \times 3$ stripes or stixels. For each stripe, the neural network must determine the average vertical pixel location $v(i)$ of the nearest obstacle presented (Fig. 4).

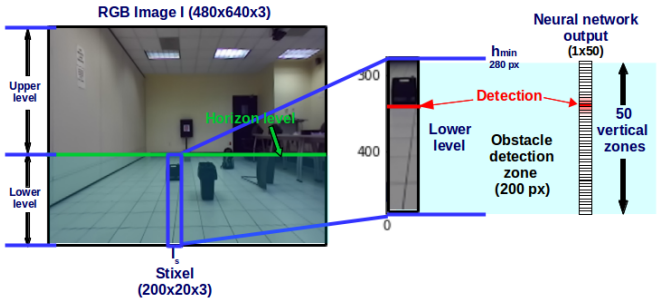

Fig. 4. StixelNet, output layer behavior. The network is trained to predict the nearest obstacle position for every 20 pixels of the image.

Thus, each neuron k at the output layer of size N assigns a probability $P(v)$ that the nearest obstacle on the stripe is presented in the row v, which by applying a simple linear regression is given by the expression:

$$v = \frac{h - h_{min}}{N} k + h_{min} \tag{1}$$

while the pixel column u is determined by the middle column w of the stripe s, according to the relation:

$$u = ws - \frac{w}{2}, \tag{2}$$

In this paper, an obstacle detection zone with limits $h \in [280, 480]$ pixels, $N = 50$ sections and stixel dimensions of $w = 20$ pixels were considered. Furthermore, by taking the maximum probability row according a threshold measurement $max(v(i)) > thresh$, the position v of the nearest obstacle presented in $V(i)$ is determined.

B. Training set building process

To build the training dataset, a ZED stereo camera was employed to take RGB images and depth maps from different scenes through indoors, where people, walls and a wide variety of objects were presented. For each image, manual segmentations were carried out to detect the nearest obstacle for every 20 pixels of width, in order to generate 100 labeled images distributed in 3,200 RGB stripes. Then, by taking advantage of the proposed detection zone $h \in [h_{min}, h_{max}]$ a data augmentation process was carried out; by defining new detection zones $h \in [h'_{min}, h'_{max}]$ on each labeled image, a single stripe could generate multiple labeled samples (at different pixels of height). At the end of the process, 115,500 stixels distributed in 104,000 for training and 11,500 for testing were generated.

The neural network was implemented and trained using the Caffe framework [17], during 20,000 iterations with a batch size of 64 using Adam [18] as learning law with a learning rate of 5.25×10^{-5}, using the PL-Loss [11] cost function. Once trained, the neural network learns to detect obstacles for each 20 horizontal pixels in the image, showing good generalization capacity with new samples (Fig. 5).

Fig. 5. Generalization capacity of StixelNet using two validation images, detected obstacles are drawn with red lines for every 20 horizontal pixels.

IV. Object detection

The objective of this stage is the detection of potential targets that the robot may want to reach or interact with during a movement execution. Particularly for this work, the network is intended to detect pedestrians that can be presented in the workspace, which is one of the most common situation for dynamic obstacles through indoor environments. Based on recent works, You Only Look Once (YOLO) [19] is a state of the art neural network that employs bounding boxes to localize and classify different object classes. Recently, the last version (YOLOV3) was released to recognize effectively 100 different classes with reasonable processing speed in most of the modern hardware systems. In practice, however, it was difficult to run the model alongside the other modules of the system due to memory limitations. Whereby, a fewer set of 9 classes (including pedestrians and a small set of other object types) were considered to develop and train a new smaller model.

This architecture is based in a simplified version of Tiny YOLO, with a reduced number of neurons on the last layer to recognize a fewer set of classes; this architecture has 13 convolutional layers followed by a set of YOLO detection layers, where Maxpool layers are placed at the output of each one in order to reduce the processing volume gradually to a final detection layer at the output, which generates bounding box coordinates for the objects corresponding to a selected set of classes: person, bottle, cup, chair, dining table, monitor, laptop, mouse and keyboard.

A. Forward processing

In this scheme, the neural network gets a $416 \times 416 \times 3$ RGB image and produces a 3D tensor of size $N \times N \times [3(4 + 1 + classes)]$ with codified information about the position (bounding box coordinates), objectivity and class scores about the objects detected in the image.

B. Training set building process

The training and testing datasets for this network were taken from a subset of images from the MS-COCO dataset [20], where for all the considered classes a total set of 137,645 images distributed in 118,287 for training and 19,358 for testing were taken. This neural network was implemented and trained using the Darknet framework [19], trough 500,200 iterations with a batch size of 64 and a learning rate of 2×10^{-4}.

V. Trajectory generation

A. 3D Position recovering

By employing the camera intrinsic matrix K, the 3D position of each detected object $p(u, v)$ can be calculated using the transformation [21]:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = z_c K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{3}$$

where z_c is obtained directly from the depth map, according to the median pixel neighborhood of each detection.

B. Filtering and tracking scheme

Once the position of all detected objects is calculated, it is necessary considering some problems related to possible changes in the environment; once a target is selected to reach, a scheme to follow the changes in its position is required, in the case that other potential targets are available, or when an occlusion occurs. Also, there could be noising problems related to a wrong object detection (by false positives) or a wrong depth calculation (due hardware). Whereby, the following scheme is proposed:

- In both cases, the available information (on the image plane $p = \{u, v\}$, and the spatial position $P = \{x, y\}$ of bounding boxes and stixels) are employed between frames to establish a relation between the objects detected in old and new frames [22].
- For incoming bounding boxes, a Kalman filter with states $\tilde{P}_t = \{\tilde{x}, \tilde{y}, \tilde{\dot{x}}, \tilde{\dot{y}}\}$ is assigned to estimate and track the changes in their position.
- In the case of obstacles, just the position $\tilde{P}_o = \{\tilde{x}, \tilde{y}\}$ is considered to deal with the noise related to a wrong depth estimation.
- If any of the bounding boxes presents occlusion, the Kalman filter is updated with no measurements using a constant velocity [23].
- For incoming objects where a relation can not be established. a new filter is assigned.
- Any bounding box detection is preserved during T frames, after this period it is deleted.

By considering a general processing speed of the modules (15 fps), a period of $T = 60$ is considered to keep the information of previously detected bounding boxes.

C. Trajectory generation

Per each frame, once the position of all the incoming objects is filtered, the standard RRT algorithm is employed to generate a collision-free trajectory. This algorithm considers two branches P_{origin} and P_{target} with initial points at the origin and goal respectively, both branches grows iteratively in the direction of a random point x_{rand} once a connection between both can be established, where each branch could establish a new connection only if no collisions are presented with other objects in the workspace. Between frames, a previously trajectory is preserved until the position of an object compromises a new collision, or when the target moves away by a radius R from the previously trajectory end point.

VI. Experimental results

A. Obstacle detection

The neural network was evaluated through a controlled digital validation dataset, which is composed of 5,000 images of 10 digital obstacles distributed in 10 scenarios according to the 50 positions in the detection zone $h \in [h_{min}, h_{max}]$, according to the following scheme:

1) For each true label $v_r(i)$, an estimation $v(i)$ is generated by the neural network.
2) For each sample, only detections that meet the condition $v_r(i) - \epsilon \leq v(i) \leq v_r(i) + \epsilon$ are taken as correct, where $\epsilon = 4$ pixels is a tolerance measure in accordance with contiguous detection regions (the minimum error measurement).
3) Finally, the average mean squared error $e(v_r, v)$ between real and detected positions is calculated.

After evaluations, the neural network presented an average mean squared error of 41.51 squared pixels on the entire dataset, with an average performance of 85% according to the aforementioned evaluation scheme. As can be seen in Fig. 6, all the zones have a good performance with exception of the first one (near the starting point of the detection zone, on the horizon level), this may be a consequence of the data augmentation process, as this is the zone that gets a fewer number of examples. Nevertheless, this may not a problem during a movement execution, because as the robot approaches to far objects, closer detection regions where the performance is good enough will be employed, allowing to detect them prior representing a collision risk.

B. Object detection

For object detection according to YOLO evaluations, the MS-COCO dataset was employed to validate the network according to the mean Average Precision (mAP) metric. In this scheme, the following metrics are considered:

$$Precision = \frac{TP}{TP + FP}, \; Recall = \frac{TP}{TP + FN}, \quad (4)$$

where TP, FP and FN indicate the number of true positive, false positive, and false negatives samples respectively.

Using the above measurements, the Mean Average Precision was calculated under an Intersection over Union (IoU) index of 0.5. These results are presented in Fig. 6 for the selected set of classes, where a general performance of $mAP = 35.9$ is achieved, which is consistent with the results of the original publication [19].

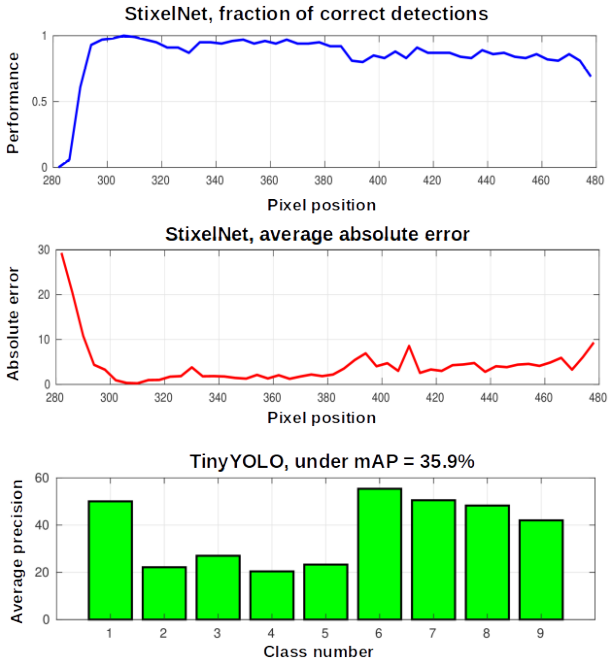

Fig. 6. General performance of the detection system under the generated validation dataset. StixelNet at the first two rows for performance and absolute error measurement (in pixels), and TinyYOLO at the last row for its performance on the selected set of classes: 1: Person, 2: Bottle, 3: Cup, 4: Chair, 5: Dining table, 6: TvMonitor, 7: Laptop, 8: Mouse, 9: Keyboard.

C. Spatial reconstruction

In order to test the performance of the position estimation module, the position of 54 manually measured obstacles was tested using the obstacle detection module according the following scheme:

- As mentioned in previous sections, the camera position $q_0(x_0, z_0)$ was considered as the reference frame, whereby all the measurements are referred to this point.
- For each manually measured object, the system was employed to detect and estimate its 3D position.
- For detections, in order to avoid false positives, only those with a probability $P(k) > 0.30$ were considered.
- For obstacles detected in more than one stripe, the average position was considered.
- Once the position was calculated, the mean squared error $e = e(P_r, P_d)$ between real and estimated positions was calculated.
- As an additional measurement rule, those obstacles with a mean squared error below the threshold measurement $e < 0.1m^2$ were considered as correct detections, while the rest as wrong.

As can bee seen in Fig. 7, the system achieves a mean squared error of $e = 0.084\ m^2$ across both dimensions, with a performance of 87 % in spatial reconstruction after obtaining 47/54 detections inside the aforementioned tolerance range $e < 0.1m^2$.

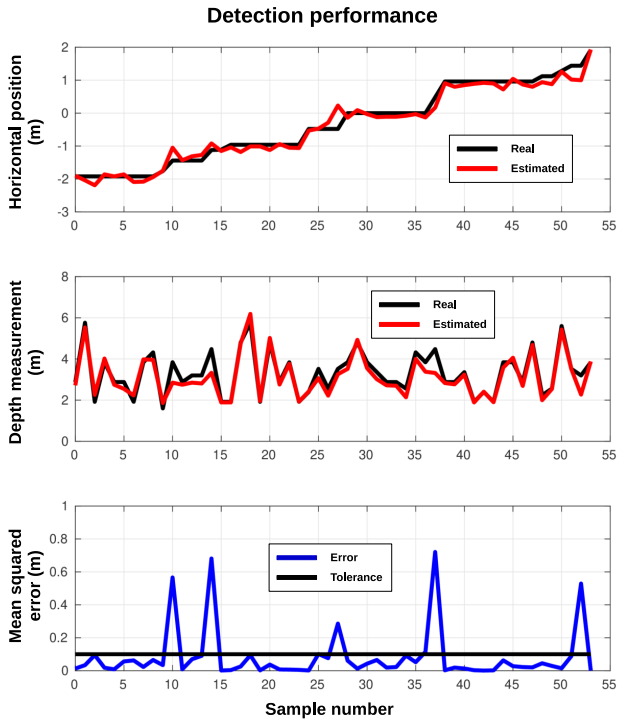

Fig. 7. Position estimation of several obstacles samples employing the modules of the perception system. Top: Horizontal distance estimation, Middle: Depth estimation, Bottom: Mean squared error.

D. Trajectory generation and tracking

All modules were implemented on an Nvidia Jetson-TX1 developer board under C^{++} and C, using libraries from OpenCV, Caffe, Darknet, ROS, and ZED. All the modules were programmed from scratch, except for the YOLO module which is a modified version from the original implementation presented in [24], where the 3D position estimation and object tracking modules were added. Once implemented, the whole system achieves an average processing speed of 15 fps, an implementation scheme of the system is presented in Fig. 8.

In order to evaluate the whole implementation, the system was employed to detect and calculate the position of different object types across an indoor environment, as well as generate a collision-free trajectory to a selected target according to the following scheme:

- Different types of obstacles were placed through an indoor scene, with pedestrians walking around.
- Two proofs were carried out; with a single and two pedestrians walking around the obstacles randomly.
- A single pedestrian was considered as the target to reach, while the second as a mobile obstacle to avoid during trajectory calculations.

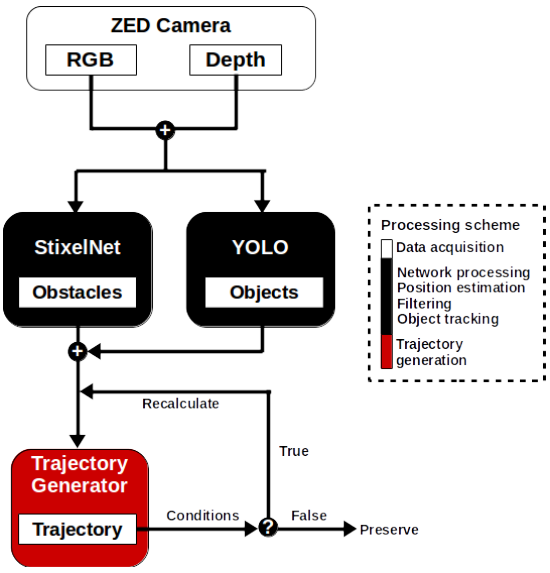

Fig. 8. Perception system ROS implementation in several nodes. A trajectory is preserved until a condition for recalculation meets.

- For the pedestrian selected as the target, a zone of radius $R = 0.6$ meters with center in its position (end zone) was considered. This zone is intended to consider small variations on the target position before calculating a new trajectory.
- Between frames, an evasion trajectory is generated and preserved until a collision compromises the path, or when the target moves out from a previous end zone.
- In order to avoid collisions, a security zone of radius $r = 0.2$ meters was considered around each obstacle and pedestrian during the trajectory generation step.
- For the general detections a bounded area of size 8×8 meters was considered, where points outside the range were not considered during the trajectory generation process (avoiding far objects).

These experiments are presented in Fig. 9 by employing several interfaces to represent each one of the obstruction problems, for 2D segmentation (OpenCV) and 3D problem representation (RViz). As can be seen, the different system stages were able to work together in order to detect all the relevant information from the obstruction problems (potential targets to reach, and obstacles to avoid), where in all the cases a collision-free trajectory can be generated from the robot initial position to the selected target. The system is also capable to generate a new trajectory each time that one of the previously mentioned recalculation conditions meet, as well as distinguish and track the position of the desired target when multiple of them are available, even when an occlusion (with other object) covers the goal for a few frames (row 4 on Fig. 9), proving to be robust against the problems presented in section V. For additional multimedia resources about the experiments, please consult: https://drive.google.com/drive/folders/1pOFP1poUn_zYCbZpQsmxyoOIcNDdANDi.

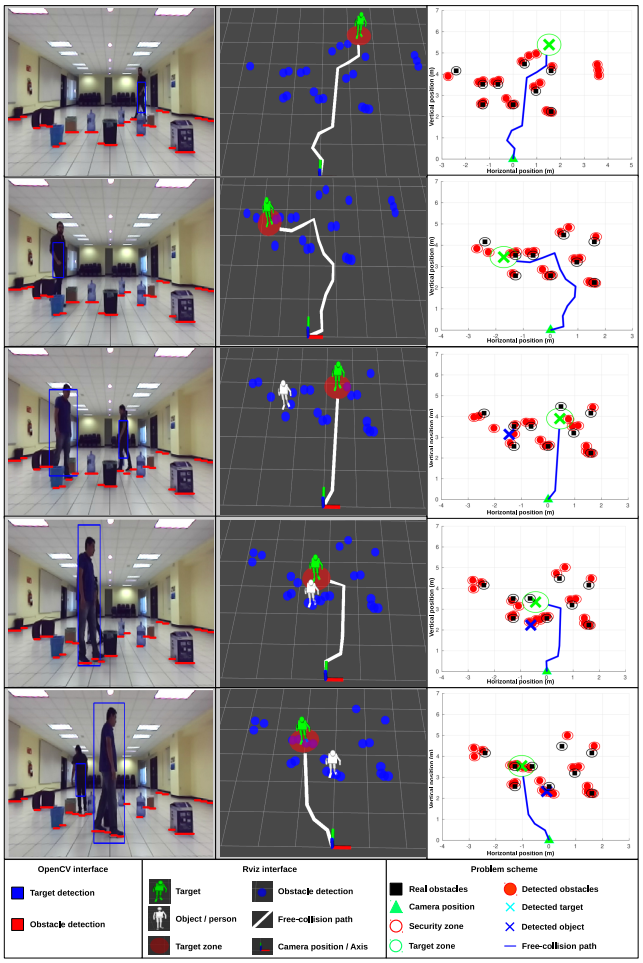

Fig. 9. Experiments evaluated in real-time using the full perception system, for a single (2 top images), and two pedestrians (3 bottom images). Left: segmented image, center: ROS Rviz visualization, right: Problem scheme.

VII. Conclusions

In this work, a set of CNN based modules in parallel were developed to detect a set of objects of interest through the environment of a mobile robot, where stereo vision and tracking and filtering algorithms were employed to recover the 3D position of each detection, in order to generate a collision-free trajectory to a desired target by employing the RRT algorithm. A replanning scheme was also proposed to deal with the position of dynamic objects, in order to calculate a new trajectory each time that is required. Each stage was evaluated through several performance test, proving to be enough robust and fast for a wide-range of practical real-time applications. As future work, we will consider interactions with other types of targets and obstacles, scenarios with a high complexity like outdoor environments, and the situation of replanning when the robot is following the generated path.

Acknowledgment

The authors would like to thank CONACYT and CINVESTAV-IPN for the scholarship and the economic and technologic support for the realization of this work.

References

[1] H. Choset, *Principles of robot motion: theory, algorithms, and implementation*. Cambridge, MA: MIT press, 1 ed., 2005.

[2] A. Burlacu, S. Bostaca, I. Hector, Herghelegiu, *et al.*, "Obstacle detection in stereo sequences using multiple representations of the disparity map," in *20th International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 854–859, IEEE, 2016.

[3] S. Mane and S. Vhanale, "Real time obstacle detection for mobile robot navigation using stereo vision," in *International Conference on Computing, Analytics and Security Trends (CAST)*, pp. 637–642, IEEE, 2016.

[4] B. Huval, T. Wang, S. Tandon, J. Kiske, *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv:1504.01716*, 2015.

[5] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, no. 1, pp. 85–117, 2015.

[6] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[7] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv:1502.03167*, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[11] D. Levi, N. Garnett, E. Fetaya, and I. Herzlyia, "Stixelnet: A deep convolutional network for obstacle detection and road segmentation," in *British Machine Vision Conference*, pp. 109.1–109.12, 2015.

[12] E. Macias, A. Cruz, J. Zamora, and E. Bayro, "Indoor navigation based on model switching in overlapped known regions," in *Robot Motion and Control (RoMoCo), 2019 12th International Workshop on*, pp. 38–43, IEEE, 2019.

[13] W. Chen, T. Qu, Y. Zhou, K. Weng, *et al.*, "Door recognition and deep learning algorithm for visual based robot navigation," in *International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1793–1798, IEEE, 2014.

[14] D. Ribeiro, A. Mateus, P. Miraldo, and J. C. Nascimento, "A real-time deep learning pedestrian detector for robot navigation," in *International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 165–171, IEEE, 2017.

[15] R. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[16] T. Ho, "Random decision forests," in *International Conference on Document Analysis and Recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, ACM, 2014.

[18] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv:1804.02767*, 2018.

[20] T. Lin and M. Maire, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[21] R. Davies, *Computer and machine vision: theory, algorithms, practicalities*. Egham, Surrey: Academic Press, 4 ed., 2012.

[22] M. Buehrem, "Functions for the rectangular assignment problem." MATLAB Central File Exchange, 2019. Accessed on: Nov. 5, 2019. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem.

[23] K. Saho, "Kalman filter for moving object tracking: Performance analysis and filter design," *Kalman Filters-Theory for Advanced Applications*, pp. 233–251, 2017.

[24] M. Bjelonic, "YOLO ROS: Real-Time Object Detection for ROS." leggedrobotics, 2018. Accessed on: Mar. 15, 2019. [Online]. Available: https://github.com/leggedrobotics/darknet_ros.