Multiple object tracking system for autonomous vehicles

Camilo A. Mosquera Victoria
camilo.mosquera@uao.edu.co

German A. Dussan Narvaez
german.dussan@uao.edu.co

Juan C. Perafan Villota
jcperafan@uao.edu.co

Victor Romero Cano
varomero@uao.edu.co

Walter Mayor Toro
wmmayor@uao.edu.co

Electronics and Automation Department
Universidad Autónoma de Occidente
Cali, Colombia

Abstract— Tracking is the process of giving a moving detection, a unique tag or object ID and then being able to identify that same object throughout future frames. This paper presents the implementation of a Multi Object Tracking (MOT) system on a Jetson TK1 development board in order to give an autonomous vehicle the capability to detect, track and measure its distance to passing cars in real time. The experimental dataset was collected from a moving car using a cellphone as a video recording system. To detect the desired objects, various pre-trained convolutional neural-network-based detectors were re-trained using the left color images for object detection evaluation in 2012the KITTI Dataset), then, these nets were tested for accuracy using an Intersection Over Union (IOU) approach. For speed evaluation the FPS of the nets were adjusted for a real time detection until the result video was fluid enough to work on a laptop with a Nvidia GeForce 920 graphic video card. After that, the Kalman filter in combination with the Hungarian algorithm and the IOU evaluation were used to perform the tracking, and the distance was calculated calibrating the parameters of the focal length of the camera and the general height of a car. Finally, tests were done in real time in a car, using the Jetson TK1 at 5 FPS.

Keywords— Tracking, detection, Jetson TK1, Kalman Filter, Hungarian algorithm

I. INTRODUCTION

As reported by the World Economic Forum in 2016, the number of vehicles in urban centers will double by 2040 reaching the 2 billion mark. The necessity of improve future urban mobility have allowed the advance of new autonomous technologies. Therefore, autonomous vehicles are called to be an important tool for improving transportation systems. The SAE International defines the levels of autonomy of a self-driving car as follows: level 0 is a car with no automation; level 1 is when the car can perform some small steering or acceleration tasks without human intervention, but everything else is still under human control; level 2 is advance cruise control or autopilot system, where the car can take some safety actions, but the driver has to stay alert at the wheel; level 3 stills requires a human driver, but under certain traffic or environmental conditions the driver could derivate some "safety-critical functions" to the vehicle; level 4 is a car that can drive itself almost all the time, but still would require a human driver for specific conditions; lastly level 5 is a fully automated car that doesn't require a human driver at any time.

Enterprises such as Waymo, General Motors and Argo IO have achieved a level 4 of autonomy, but the highest level in the market as of now is only 2 with the Tesla cars. In Colombia the level of autonomy that there is in the streets is 1, with Chevrolet and Volvo cars, also currently there are a handful of Tesla cars with level 2 of autonomy, but their autonomous functions are restricted to European countries.

Looking to achieve a higher autonomy level for cars in Colombia we propose the design of a system that allows to detect, track and measure the distance between the autonomous car and other vehicles around it so it can take quick actions when other cars approach it. To achieve that a system consisting of 3 subsystems was implemented: the detection subsystem, in which the convolutional neural network Tiny-YOLOv3 of the Darknet interface is used to detect the cars; the tracking subsystem, in which a bank of Kalman filters is used to perform the tracking of cars, and the Hungarian algorithm in combination with the Intersection over Union evaluation method (IOU) are used to assign the detections to the filters; and the decision making subsystem, in which the similarity triangle and a trigonometric function are used to measure the distance and the angle of the detections with respect to the ego-car, respectively. All the system was selected with the thought of been able to implement it in an embedded mobile board, the NVIDIA Jetson TK1, and been able to test the system in real time.

II. RELATED WORK

The artificial vision aims to generate intelligent and useful descriptions of scenes and visual sequences, as well as the objects that appear in them, by performing operations on images and videos. These operations are based on the detection and tracking of objects, improving the efficiency and accuracy of these operations over time, with the objective of studying the detection and tracking. There are works whose objective is to improve each of the procedures they require. Also, recent works

of tracking multiple objects has shown significant progress, which will be presented below:

In 2017, Agarwal and suryavanshi [3] performed a work in which they built a MOT using deep neural networks, in which they combine the faster R-CNN detector [4] with the GOTURN architecture [5]. They divided the MOT in two categories: detection of multiple objects and the correspondence of those objects.

For the first part they used the CNN (convolutional network), which frames the region of interest, and combined it with a CNN based on regions, which detects the presence of the object in them, because the network does not require knowledge of the object class to perform the detection, the architecture is flexible and can be adjusted to multiple designs. To track objects, they used GOTURN, which uses "t" and "t-1" images of a real-time video and individually enters them in different CNNs. The output of these CNNs is enters another neural network that tries to make the correspondent relation of the object, looking for similar characteristics in the original object. A drawback that is evident in the process is that the GOTURN architecture only can track a single object, that's why, the CNN its necessary to perform it to multiple objects.

On the other hand, in 2018, Beaupre, Bilodeau and Saunier [6] they presented a work to improve the tracking of multiple objects. They focused on how to create better foreground images, which then become the entry of MKCF (Multiple kernelized Correlation Filter Tracker) [7] and the UT (Urban tracker) [8]; these two helped to study the effects of these improvements on the performance or effectiveness of trackers. In order to improve the foreground images, they applied the following processes: Background subtraction [9], Optical Flow [10] and Canny Edge Detector [11]; with the first one they located the regions of interest, with the second they separated object and with the third they obtained the edges of the objects in the foreground, with which the sizes of the objects were adjusted. Once this information was gathered, they generated a new binary image.

Likewise, in 2019, Li, Dobler, Song, Feng and Wang [12], presented an innovative work. Detection and tracking are not used separately, but rather create a unified structure network to simultaneously perform the MOT task. This network takes a consecutive GoP (group of Pictures) like a 3D representation and detects the objects that are moving inside the GoP through "tubes" that used a 3D network [13] and a VGG (Visual Geometry Group)[14], with convolution in 2D as base networks to perform the extraction of the spatial and temporal characteristics in the input videos, also an spatial transformer module [15] is used to deal with the variation of orientation of the objects. Later, a TPN (tube proposal network) [16] generate "tubes" for the detected objects. Finally, a post-TPN stage refines the classification and the parameters of localization of the "tubes" located to obtain a better accurate detection.

Our proposal implements a real time MOT system in an low-cost embedded mobile board, using a deep learning approach, the CNN, to detect the desire objects and using an algorithm that relies heavily on the state equations of the object, as well as the previous detections and predictions, the Kalman filter, to track them. This reduces the time and computer resources it takes to deploy the system.

III. System Description

Our proposal is composed by a re-trained convolutional neural net to make the detection, a Kalman filter coupled with the Hungarian algorithm to track the detection and a triangle similarity to get the distance and the angle of the detection (see Figure 1).

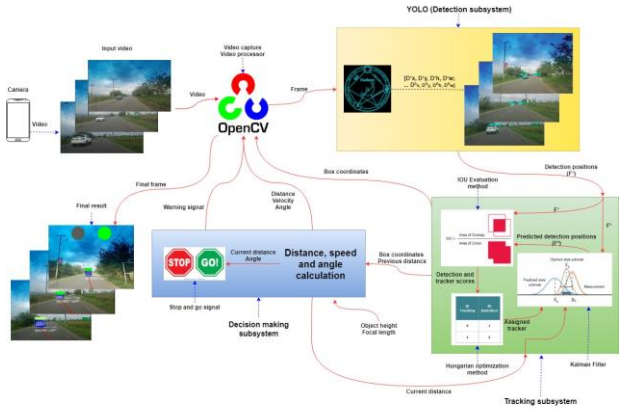

Figure 1. Project diagram.

The result is a video processed from live feed in which we see what an autonomous car would "see". The system is divided into 3 major modules: The detection subsystem, the tracking subsystem and the decision-making subsystem.

A. *Detection subsystem*

Before choosing a CNN to work with, we re-trained them to see if we could improve their performances against cars as our desire object to be detected.

First, we selected a dataset that had a good amount of detections and a good point of view (because we were going to point the camera to the front of the car, we needed a dataset consistent of this kind of images). Amount the ones considered, we choose the KITTI dataset, which consist of more than 80.000 labeled objects between cars and pedestrians (we only focused on cars).

Before starting the re-training process, an evaluation was performed to check how good the detectors were out of the box, doing an IOU evaluation over 20 random images taken from the testing folder of the KITTI dataset (this images didn't have labels, so it was necessary to label them manually, using labeling). In Table I, the results for the 5 detectors used is condensed, showing the overall performance as well as the quality of the detections.

Table I. IOU evaluation of the original detectors.

Net	IOU (Average)	Detections (%)	Excellent (%)	Good (%)	Bad (%)
YOLOv3	0,647	81,55	9,52	73,81	16,67
YOLOv3-tiny	0,244	29,13	0	6,67	93,33
SSD MobileNet V2	0,649	36,89	18,42	73,68	7,89
Faster RCNN Inception V2	0,744	86,41	16,85	76,4	6,74
Faster RCNN ResNet	0,832	83,5	29,07	65,12	5,81

As seen in Table I, the detectors with the worst performance were the tiny version of YOLOv3 and the SSD Mobilenet v2, as they were able to detect 29% and 36% of the total number of truth detections, plus the Mobilenet v2 was able to get a good quality on the detections, while the quality of the detections of the YOLOv3 tiny was rather poor.

Once the first assessment was done the re-training was performed, having in mind that the amount and size of the images (7481 and 1224x370px respectively) were big. An speed test was done to measure the amount of time that it would take to re-trained each detector locally (on a laptop with a 2GB Nvidia GeForce 920M video card) and on the cloud (On Google Collaboratory, with an Nvidia Tesla K80), that's why the re-training process was allowed to run for a few steps, then the final time was calculated based on the final number of steps that the re-training would finish with (See Table II).

Table II. Summarized retraining times.

Net	Local re-train (Nvidia GeForce 920M)		Re-train on the cloud (Nvidia Tesla K80)	
	Time (1 step) (s)	Total time (s)	Time (1 step) (s)	Total time (s)
YOLO V3	7.4-16	22200-48000	1.7-8	7300
YOLO V3 Tiny	4.3-6	21500-30000	1.7-1.9	10000
	Time (100 steps) (s)	Total time (s)	Time (100 steps) (s)	Total time (s)
SSD MobileNet V2	135	13500	60	6000
Faster RCNN Inception V2	40	4000	10	1000
Faster RCNN ResNet	78	7800	31	3100

Based on this, the re-training on the cloud was chosen, as it was faster and the data was easier to handle (to add space a Google Drive account was linked, as it allow us to move the dataset from the drive to the Collaboratory workspace, knowing that every time the workspace was close for more than a few hours the data in it was deleted), 2 scripts were created to re-train both Tensorflow and YOLO detectors, and at the end of each session the results were saved in the drive, and the re-trained nets were tested to see if there were improvements and to extract the coordinates to perform a new IOU evaluation. In Table III the results of the re-training of all the detectors is shown, both the overall performance and the quality of the detections.

Table III. IOU evaluation of the retrained detectors.

Net	IOU (Average)	Detections (%)	Excellent (%)	Good (%)	Poor (%)
YOLOv3	0,785	77,67	10	88,75	1,25
YOLOv3-tiny	0,628	84,47	3,45	70,11	26,44
SSD MobileNet V2	0,698	60,19	6,45	85,48	8,06
Faster RCNN Inception V2	0,827	87,38	15,56	83,33	1,11
Faster RCNN ResNet	0,837	80,58	22,89	71,08	6,02

Next, to compare the results Table IV is presented:

Table IV. Comparison between retrained detectors and original detectors.

Net	IOU (Diference)	Detections (%)
YOLOv3	0,138	-3,88
YOLOv3-tiny	0,384	55,34
SSD MobileNet V2	0,049	23,3
Faster RCNN Inception V2	0,083	0,97
Faster RCNN ResNet	0,005	-2,92

YOLOv3 detected 4% less objects, but the quality of the detections improved 14%, and YOLOv3 tiny got a 55% increment on the detections and a 39% increment on the quality of such detections. Mobilenet v2 was able to detect 23% more object, while its quality shows a little improvement (5%), meanwhile the detectors Inception v2 and Faster Resnet remained almost the same, Resnet detector showed signs of deterioration (its detection percentage decreased 3%). The detector Faster Resnet was originally trained with the KITTI dataset, so re-training it was redundant, while the other 4 detectors were all originally trained with the COCO dataset.

Finally, to choose the detector to work with, a speed test was performed, obtaining the FPS for which each detector worked fluid (depending on the machine if the FPS are too high the result video would look choppy), noting that the more FPS the more natural the video will look. In Table V it's presented the FPS for which each detector worked fluidly.

Table V. Optimal FPS of the detectors for a laptop with an Nvidia GeForce 920M video card.

Net	FPS
SSD MobileNet V2	5,1
Faster RCNN Inception V2	1,7
Faster RCNN ResNet	0,6
YOLOv3	6
YOLOv3-tiny	30

At the end the detector YOLOv3 Tiny was chosen based on the speed as well as the performance.

B. Tracking subsystem

To track one object the Kalman Filter was implemented, the idea is to initialize the filter by feeding it with an initial detection, then the filter, according with the model (for our case we assume constant velocity), would calculate the position for the detection in the next frame, then when the next frame's detection is received, the filter use this new position and the position predicted to predict the next position, thus, the more information given to the filter the more accurate the predictions would be, so if the detection get lost for a few instances the filter could predict in accordance with the previous behavior of the object it was tracking.

The challenge was to do this for multiple objects, in order to separate some objects from the others, so the predictions of each one of them would be as accurate as possible. To do this, it was necessary to create a filters bank, in this way every time a new detection appears, a new filter would be created to track that new

object. To improve the precision of the predictions of the filters and based on the work done in [21], the Hungarian algorithm, and pair with IOU evaluation were implemented.

Now, in each frame the detections are captured, then the detection is evaluated with each existing filter using the IOU evaluation, to check if there is a possibility that any of those detections corresponded to an existing instance of the actual filters. The IOU evaluation gave us a value from 0 to 1 of how similar the detections were in proximity and in size, and because the change in time between frames is not big. if the detection corresponds to a previous object already detected both the coordinates and the size of the detection shouldn't change a lot, and after calculating the IOU of each detection the Hungarian algorithm is performed to optimize the pairing of the detection and the existing filters.

Finally, the predicted position is updated with the real coordinates. In case there were detections that didn't match any filters, a new one would be created and initialized, and if there were filters that weren't pair with any detection the idea is to look at the number of frames that filter carries without any new detection, and if the number is greater than a set threshold the filter would be discarded and the object would be assumed as lost.

C. Decision-making subsystem

Having the detection and the tracking part tackled, the focus change to measure the distance of the detection from the ego-car (knowing that an autonomous vehicle needs to know how close are the other vehicles) to be able to make a decision or to stop, when it's safe to advance and to change lanes.

The implemented method was the triangle of similarity, and it's based on calculations over the focal length of the camera, and to make it work first the camera must be calibrated. The process of calibration consists of taking a picture of an object for which we know its measure (either height or width) and the distance from the camera to this object; then over the picture a square is drawn to enclose the object. Finally, we calculate the focal length of the camera with (1):

$$\textit{Focal length} = (\textit{object pixel height} * \textit{Known distance}) / \textit{Known height} \quad (1)$$

Basically, the distance is calculated relating the real height or width with the pixel height or width, and stating that any change is linear and that the shifting factor is the focal length. An average height of 1.5 meters is assumed for cars (It is important to mention that if the size of the video being taken changes, the focal Length must be recalculated).

Furthermore, an attempt to calculate the relative velocity of the ego-car was done by drawing the difference between the actual distance and the previous distance, and dividing that distance by 1/FPS, however sometimes the changes could be very big.

Finally, the angle of the detection was calculated getting the tangent arc of the distance from the bottom of the image to the bottom of the detection over the distance from the middle of the image to the middle of the detection. If the result angle is between 0 and 90 degrees the car is to the right of the ego car, if it's over 90 degrees, it's to the left of the ego-car.

For instance, a traffic light was implemented, to indicate when a car is too close to the ego-car, this happens when a car is less than 3 meters, and in an angle between 80 and 100 degrees.

D. Implementation of the system in a development board

To implement the system on the board it was necessary to use a different Darknet repository [17], because the original repositories work on an advance version of CUDA and unfortunately the last version that the Jetson TK1 supports was 6.5.

To get the video to the Jetson (because of the lack of an USB camera) the application Ip Webcam [18] was used, this application turned out really useful as it allow us to change the size of the image and the FPS captured (because of the computational capacity of the Jetson, as well as the computational demand of the Darknet detector, it was necessary to find the right combination of this 2 parameters, knowing that a low image size would result in the image been indistinguishable and a low FPS would result in the image too intermittent).

To access the Jetson from the car (without a monitor) the SSH protocol was used, that way from a laptop we were able to execute the commands to start the system and to visualize the results. To power up the board (it required 12V and at least 1A) a converter was used, plugging it at the output of the car. Finally, an image size of 288x288 pixels and an FPS of 5 allow us to get decent results.

IV. Results

This section presents the experimental results. Video frames were captured with the app Ip Webcam and both, the Jetson board, in charge of data processing, and the cellphone, were connected to an internet-less hotspot created on another mobile phone. The first module is the retraining and evaluation process, where it first shows an example of the original image and its detection both before and after the re-training process (see Fig. 2, Fig. 3 and Fig. 4):

Figure 2. Original image.
Source: [1]

Figure 3. Detection with the original detector.
Source: Adapted from [1]

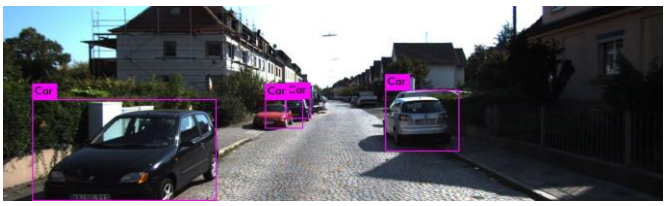

Figure 4. Detection with the re-trained detector.
Source: Adapted from [1]

Then an example of the IOU evaluation is shown, where the green rectangle shows the detection of the CNN, the blue rectangle denotes the ground truth and the red rectangle denotes a misdetection (see Fig. 5 and Fig. 6):

Figure 5. Example of the IOU evaluation.
Source: Adapted from [1]

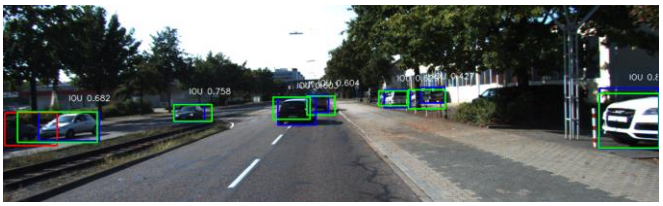

Figure 6. Example of the IOU evaluation.
Source: Adapted from [1]

In Fig. 7, the evaluation of the distance algorithm is shown, where we use a measuring tape to get the real distance (1.8 meters), we can see that the error is of 0.14 meters:

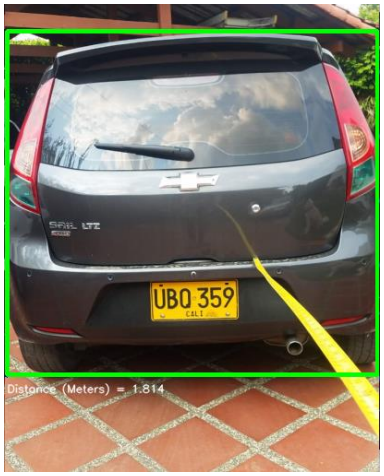

Figure 7. Verification of the distance calculator algorithm.

For Fig. 8, we use a ruler to measure the real distance of a student card (0.27 meters). The error is of 0.06 meters.

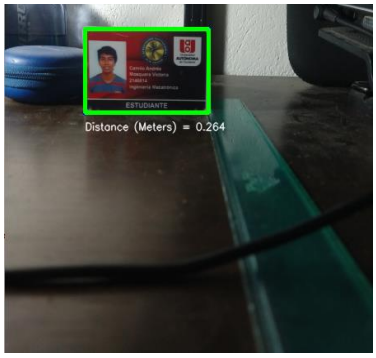

Figure 8. Verification of the distance calculator algorithm.

The precision and accuracy of the tracker was tested with pymot, a repository that implemented a way to measure this MOT metrics. Bernardin et al. define the MOTP as "the total position error for matched object-hypothesis pairs over all frames, averaged by the total number of matches made. It shows the ability of the tracker to estimate precise object positions, independent of its skill at recognizing object and keeping consistent trajectories", and the MOTA as "the account of all object configuration errors made by the tracker, false positives, misses and mismatches, (…) it gives a measure of the tracker's performance at keeping accurate trajectories" [25, p. 4]. In table VI is presented the MOTP and MOTA of the proposal's implementation.

Table VI. MOTP and MOTA of the proposal implementation.

Results	
Ground truths	1431
False positives	117
Misses	0
Mismatches	0
Recoverable mismatches	0
Non recoverable mismatches	0
Correspondences	1314
MOTP	0,94
MOTA	0,92

Finally, results of the whole system are shown in Fig 9, Fig. 10, Fig. 11, Fig. 12 and Fig. 13, in the form of a car being tracked (notice that the label doesn't change), and the distance being calculated:

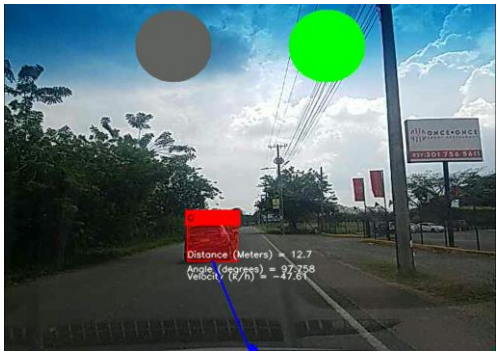

Figure 9. Tracking of a car (t=0).

Figure 10. Tracking of a car (t=1).

Figure 11. Tracking of a car (t=2).

Figure 12. Tracking of a car (t=3).

Figure 13. Tracking of a car (t=4).

V. CONCLUSIONS:

The development of this work allowed the successful detection and tracking of vehicles on the road in real time where the algorithm can generate a detection and tracking, distinguishing the number of cars that travel on the road, showing different frames of different color to distinguish cars in motion. This work aims to provide an approach to the problem of vehicle driving autonomy so that in future jobs it can be installed in vehicles and achieve a successful autonomy.

By modifying certain parameters, we achieved a better accuracy in the results that were wanted, such as the height and width. When performing tests, the variables had to be modified to achieve a better effectiveness in the detection. A balance between the input size and the SPF had to be found, as the network input decreased, the detections became less accurate.

The results that were obtained, evaluating the network training using, MobileNet, Faster RCNN inception V2, Faster RCNN ResNet and YOLO, showed that is better to use YOLO, because it has a greater speed and precision detecting cars.

ACKNOWLEDGMENT

The authors would like to thank the Universidad Autónoma de Occidente as well its Robotics research seedbed (SIR) for providing us with the knowledge and resources necessary to develop this project.

REFERENCES

[1] Geiger, Andreas & Lenz, P & Stiller, Christoph & Urtasun, Raquel. (2013). Vision meets robotics: the KITTI dataset. The International Journal of Robotics Research. 32. 1231-1237. 10.1177/0278364913491297.

[2] Reyes Fajardo, Juan Manuel. "Vehículos autónomos, ¿a la vuelta de la esquina?". [En línea]. [Consultado: 19 de septiembre de 2018]. Disponible en: https://www.publimetro.co/co/tacometro/2018/01/18/vehiculos-autonomos-la-vuelta-la-esquina.html.

[3] A. Agarwal y S. Suryavanshi, «Real-Time* Multiple Object Tracking (MOT) for Autonomous Navigation», p. 5.

[4] Girshick, Ross. (2015). Fast r-cnn. 10.1109/ICCV.2015.169.

[5] Held, David & Thrun, Sebastian & Savarese, Silvio. (2016). Learning to Track at 100 FPS with Deep Regression Networks. LNCS. 9905. 749-765. 10.1007/978-3-319-46448-0_45.

[6] D.-A. Beaupré, G.-A. Bilodeau, y N. Saunier, «Improving Multiple Object Tracking with Optical Flow and Edge Preprocessing», arXiv:1801.09646 [cs], ene. 2018.

[7] Yang, Yuebin & Bilodeau, Guillaume-Alexandre. (2017). Multiple Object Tracking with Kernelized Correlation Filters in Urban Mixed Traffic. 209-216. 10.1109/CRV.2017.18.

[8] J. Jodoin, G. Bilodeau and N. Saunier, «Urban Tracker: Multiple object tracking in urban mixed traffic», IEEE Winter Conference on Applications of Computer Vision, Steamboat Springs, CO, 2014, pp. 885-892.

[9] R. S. Rakibe y B. D. Patil, «Background Subtraction Algorithm Based Human Motion Detection», vol. 3, No. 5, p. 4, 2013.

[10] A. K. Chauhan y P. Krishan, «Moving Object Tracking using Gaussian Mixture Model and Optical Flow», International Journal of Advanced Research in Computer Science and Software Engineering, p. 4, 2013.

[11] L. Ding y A. Goshtasby, «On the Canny edge detector», Pattern Recognition, vol. 34, No. 3, pp. 721-725, mar. 2001.

[12] Li, Chenge Lexi & Dobler, Gregory & Feng, Xin & Wang, Yao. (2019). TrackNet: Simultaneous Object Detection and Tracking and Its Application in Traffic Video Analysis.

[13] Hou, Rui & Chen, Chen & Shah, Mubarak. (2017). An End-to-end 3D Convolutional Neural Network for Action Detection and Segmentation in Videos. 14.

[14] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.

[15] Jaderberg, Max & Simonyan, Karen & Zisserman, Andrew & Kavukcuoglu, Koray. (2015). Spatial Transformer Networks. Advances in Neural Information Processing Systems 28 (NIPS 2015).

[16] R. Hou, C. Chen, y M. Shah, «Tube Convolutional Neural Network (T-CNN) for Action Detection in Videos», en 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 5823-5832.

[17] dtmoodie. 2016, darknet, [Github respository]. [Consultado: 3 de septiembre de 2019]. Disponible en: https://github.com/dtmoodie/darknet/tree/tk1

[18] Pavel Khleovich. (2010). Ip Webcam (1.14.31.737) [Aplicación Móvil]. Disponible en: https://play.google.com/store/apps/details?id=com.pas.webcam&hl=es_419

[19] Rosebrock, Adrian. “Find distance from camera to object/marker using Python and OpenCV”. [En línea]. [Consultado: 10 de octubre de 2019]. Disponible en: https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/

[20] Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement.

[21] Toro, Walter & Perafan, Juan & Mondragon, Oscar & Obando-Ceron, Johan. (2019). Divide and Conquer: an Accurate Machine Learning Algorithm to Process Split Videos on a Parallel Processing Infrastructure.

[22] David Dao. 2016, Tensorflow models, [GitHub repository]. [Consultado: 10 de mayo de 2019]. Disponible en: https://github.com/tensorflow/models

[23] Sang, Jun & Wu, Zhongyuan & Guo, Pei & Hu, Haibo & Xiang, Hong & Zhang, Qian & Cai, Bin. (2018). An improved YOLOv2 for vehicle detection. Sensors. 18. 4272. 10.3390/s18124272.

[24] Abadi, Martín & Agarwal, Ashish & Barham, Paul & Brevdo, Eugene & Chen, Zhifeng & Citro, Craig & Corrado, G.s & Davis, Andy & Dean, Jeffrey & Devin, Matthieu & Ghemawat, Sanjay & Goodfellow, Ian & Harp, Andrew & Irving, Geoffrey & Isard, Michael & Jia, Yangqing & Jozefowicz, Rafal & Kaiser, Lukasz & Kudlur, Manjunath & Zheng, Xiaoqiang. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

[25] Bernardin, Keni & Elbs, Alexander & Stiefelhagen, Rainer. (2006). Multiple object tracking performance metrics and evaluation in a smart Room environment. Proceedings of IEEE International Workshop on Visual Surveillance.