2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin) | 978-1-7281-5885-3/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICCE-Berlin50680.2020.9352201

Camera-LIDAR Object Detection and Distance Estimation with Application in Collision Avoidance System

Nikola Sakic, Momcilo Krunic, Stevan Stevic, Marko Dragojevic
Department for Computer Engineering and Communications
Faculty of Technical Sciences, University of Novi Sad
Novi Sad, Serbia
@rt-rk.com

***Abstract*—Nowadays we are aware of accelerated development of automotive software. Numerous of ADAS (Advanced Driver Assistance Systems) systems are being developed these days. One such system is the forward CAS (Collision Avoidance System). In order to implement such a system, this paper presents one solution for detecting an object located directly in front of the vehicle and estimating its distance. The solution is based on the use of camera and LIDAR (Light Detection and Ranging) sensor fusion. The camera was used for object detection and classification, while 3D data obtained from LIDAR sensor were used for distance estimation. In order to map the 3D data from the LIDAR to the 2D image space, a spatial calibration was used. The solution was developed as a prototype using the ROS (Robot Operating System) based Autoware open source platform. This platform is essentially a framework intended for the development and testing of automotive software. ROS as the framework on which the Autoware platform is based, provides a library for the Python and C++ programming languages, intended for creating new applications. For the reason that this is a prototype project, and it is popular for application in machine learning, we decided to use the Python programming language. The solution was tested inside the CARLA simulator, where the estimation of the obstacle distance obtained at the output of our algorithm was compared with the ground truth values obtained from the simulator itself. Measurements were performed under different weather conditions, where this algorithm showed satisfactory results, with real-time processing.**

Keywords—automotive, software, computer vision, sensor fusion, object detection, distance estimation

I. Introduction

CAS (Collision Avoidance System) is one of the primary parts in autonomous driving systems, because it directly protects the vehicle and the driver from a frontal collision. Most modern vehicles have implemented some systems that make up level 2 [1] autonomous driving, including CAS. Most new vehicles have CAS fitted as standard. In fact, since 2019, it is no longer possible to get a five-star Euro NCAP safety rating [2] without it. The algorithm on which this system is based, processes information from different types of sensors that collect various types of environmental information. The main types of sensors used for this purpose are camera, LIDAR (Light Detection and Ranging) and radar. CAS uses some combination of these sensors and uses different types of sensor fusion. All autonomous driving systems are based on a similar data processing flow and it is shown in Fig. 1.

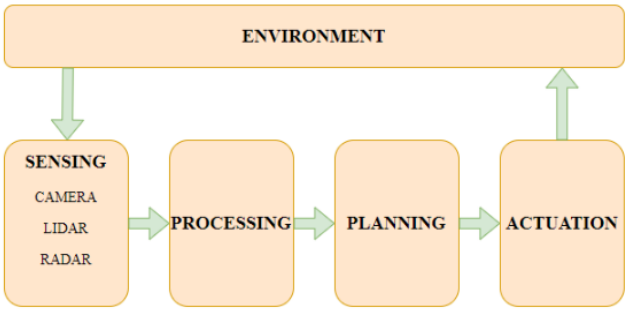

Fig. 1. Data flow in automotive software.

A major development challenge is finding the right solution for this problem. Any safety-critical software must comply with the ISO 26262 standard [3]. For that reason, it is important to find the best combination of available sensors and choose appropriate algorithm for processing that information. The main challenges that arise in the implementation of such a system is the accuracy in terms of object detection and even more important factor is the speed of processing. Early CAS functioned at lower speed, avoiding collision with other vehicles only. However more recent systems operate up to motorway speeds and in collision with pedestrians and cyclists.

Main part in CAS implementation is object detection and tracking. Due to the rapid development of deep learning and convolutional neural networks (CNNs) in recent years, 2D camera object detection and recognition have been improved. However, with 2D object detection on the input frame from a monocular camera, it is not possible to determine the 3D position of the detected object. Simple space calibration between image space and the ground plane itself is not possible due to various reasons, like sudden braking and acceleration of the vehicle. In order to determine the 3D position of the detected object, it is convenient to use a LIDAR sensor. The LIDAR itself is not sufficient to determine exact 3D boundaries, as well as to perform the classification of the detected object. The reason for this is the reduction in the density of the sampled data, as well as the impossibility of colour perception. However, by a combination of these two sensors where the camera is used for detection and classification, while data from LIDAR

sensor are used to determine the distance of the detected object, it is possible to localize obstacle ahead.

In this article, we present one approach which combines the advantages of both, 2D object detection from chromatic images using deep learning techniques and accurate 3D measurements from LIDARs, in order to obtain an accurate 3D object detection. We will see how this module is coupled to the lane keeping sub-system, where together they form a system that implements level 2 of autonomous driving. The entire system is based on Autoware platform [4] and Carla simulator [5] is used for validation of our solution.

II. Related Work

Over the last decade, and to this day, different solutions have been proposed for successful and efficient obstacle detection, based on the use of different types of data from different types of sensors. Some of these solutions base their implementation just on camera data. Such solutions use a stereo camera [6][7]. Many recent stereo-vision based obstacle detection systems require a dense disparity map and then locate the obstacles according to the depth information. This kind of solution requires the usage of significant processor's resources, because two neural networks for detection must be run in parallel for two different frames. However, calculating the correspondence for each pixel is very time consuming. In automotive applications object detection must be performed in real time. On the other hand, such solutions use all advantages of the camera in terms of detection and classification, but also largely depend on the disadvantages of the camera sensor in the form of external visibility conditions.

On the other hand, some other solutions use the camera and radar as the primary source of information [8]. Such solution demonstrates how 3D object detection can be achieved using deep learning on radar point cloud and camera images. Currently, the main limitations of the performance of the object detection with radar data and camera images is the dataset, which is until now rather small. However, the results show the deep learning is generally a suitable method for object detection on radar data. This combination of sensors is commonly used within the industry today. Information from radar sensor makes such solution robust in conditions of reduced visibility.

There are also certain solutions that use a combination of camera and LIDAR [9], which we also decided on in our work. Three-dimensional LIDAR sensor can directly obtain the position and geometric structure of an object within its detection range, whereas the use of vision cameras is most suitable for object recognition. Such solution utilises 3D LIDAR data to generate accurate object-recognition proposals. Then, these candidates are mapped onto the image space from which regions of interest (ROI) of the proposals are selected and input to a convolutional neural network (CNN) for further object recognition. This solution meets the real-time demand of autonomous vehicles and as an output gives 3D bounding boxes of the detected objects. We use some different approach. First of all, we perform 2D object detection on the camera frame, and then we translate LIDAR point cloud to image space where we mapping certain 3D points to corresponding bounding box. Each of these point clouds represents a candidate for belonging to a particular object, what is determined in the clustering phase.

III. Implementation

Unlike mentioned solutions, we detect objects just on the image what we get from the monocular camera, while LIDAR data is mapping to 2D space based on the transform matrix obtained by the calibration process without any up sampling. The idea is that we get two separate representations of the same scene. One view with detected objects from the camera and another one with displayed point clouds obtained from LIDAR. After that, we would determine which of points clusters belongs to which bounding box. The data processing procedure itself is shown in the following sections.

A. System arhitecture

The whole system (Fig. 2) implements two functionalities, CAS and Lane Keeping (LK). These two modules process data coming directly from the sensors. Input for LK is just camera data, while like we mentioned CAS processes data from camera and LIDAR. What represents the output of the LK module is information about the position of our vehicle in the lane. On the other hand, CAS module will generate information about distance to the obstacle ahead. Based on this information, the motion planning module would be able to generate a trajectory in the form of a waypoints that would be sent to the last module in charge of actuation of the vehicle.

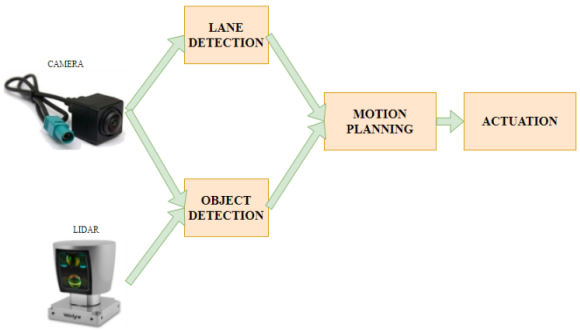

Fig. 2. Block diagram of the whole system that implements CAS and LK functionalities.

Further in this paper, the implementation of the object detection module is explained in terms of distance estimation to the obstacle ahead. The internal structure of this module is shown in Fig. 3.

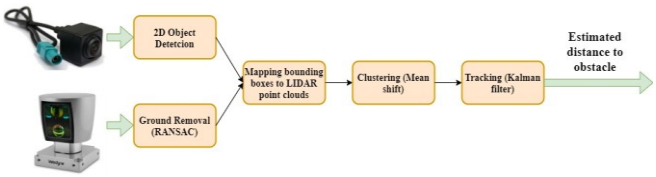

Fig. 3. Object detection sub-system overview.

B. Camera-LIDAR calibration

When fusing data from several sensors, it is necessary to perform their spatial as well as temporal calibration. Their spatial calibration is necessary in order to be able to translate 3D data obtained from LIDAR sensor into 2D camera space. In order to do that, it is necessary to determine in advance geometric relationships from observable feature detected by two sensors separately. Sometimes it is not easy to do calibration due big sparsity of LIDAR data. The process of calibration is based on determination of extrinsic LIDAR-

camera parameters R and T, where R represents rotation matrix and T represents transformation matrix between LIDAR and camera coordinate systems. Commonly used solution uses chessboard pattern and auxiliary calibration objects [10]. In ROS environment used during the development of this solution, there is a built-in package that is used for sensor calibration. LIDAR-Camera extrinsic calibration is performed by clicking on corresponding points in the image and the point cloud. More details on the use of this package in the ROS environment can be found on the official web page [11]. As for the temporal calibration, it is not covered within this implementation,but two consecutive samples were taken, which we get from each of these sensors. In some of the extensions of this implementation this must certainly be added, because based on it we know that the samples we get from different sensors come from the same time moment with a certain threshold.

C. 2D object detection – YOLO

The image we get from the camera is forwarded to the input of the algorithm for 2D object detection. In our solution, we decided to use YOLO (You Only Look Once) algorithm [12] used to detect the objects in the image. YOLO is one of the fastest neural networks for image processing and this is the main reason for choosing this solution, on some GPUs processing speeds of up to 45 fps can be achieved. The reason why this network is so fast lies in the fact that it performs object detection and classification with one pass, while many other neural networks use two passes. Version 3 of the YOLO introduces some additional improvements [13]. What is another important feature of YOLO and what gives it an advantage over other CNN (Convolutional Neural Network), we can easily tradeoff beetwen speed and accuracy simply by changing the size of the model, no retraining required.

We based the solution so that the camera is the basic source of information. If the camera does not detect the obstacle in time or does not detect it at all, the algorithm will not be able to react and safely stop the vehicle. From the point of view of such implementation, timely detection of objects in the image obtained from the camera is of crucial importance. For this reason, it is very important to choose the appropriate model size as a link between the success of the detection itself and the processing speed. Output of this module are detected bounding boxes as it is shown on the Fig. 4.

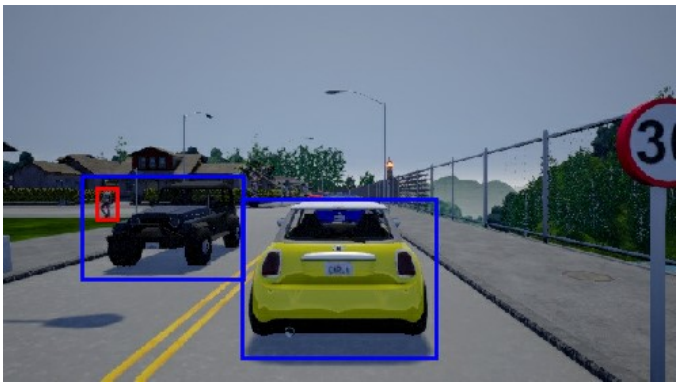

Fig. 4. Objects detected by YOLOv3.

D. Ground removal – RANSAC

RANSAC (Random Sample Consensus) is used for detection of contours and what is more important for us it is used for line detection. It finds application in image processing and lane segmentation where this algorithm is used to detect the line on the road. This is iterative method for assessment of the parameters of mathematical model based on data where the protruding values exist (outliers). E.g. method “Least squares” is sensitive to the existence of outliers. Informal, outliers’ values are data that are not in the line with the model, while data that are in the line with the model call inliers.

In our case outlier values represent all data points that does not belong to the ground plane. Before we run this algorithm, it is necessary to present 3D data from the LIDAR in 2D space, where only the depth and height component are taken. After taking only these two coordinates into account, the ground plane can be viewed as a single line (Fig. 5). The RANSAC algorithm itself is based on an iterative procedure, where two arbitrary points are taken within each iteration, whose merging gives the assumed direction. With a defined threshold, the number of inliers is calculated. After a given number of iterations, the solution with the most inliers is selected. The number of iterations is determined with the ratio of outliers-inliers, as well as needed probability of ground plane detection.

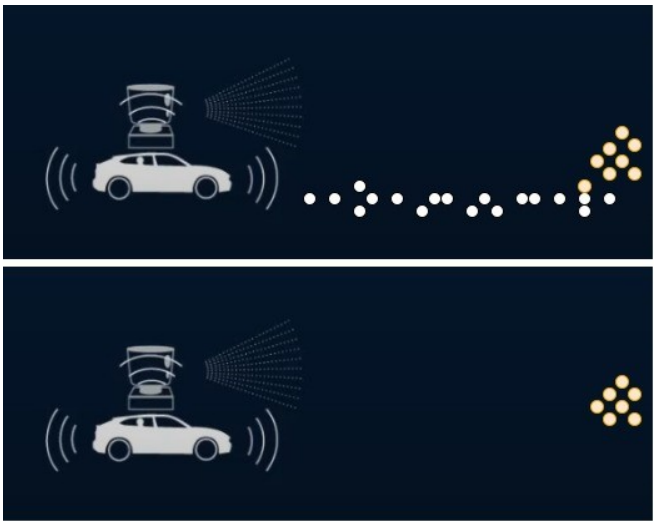

Fig. 5. RANSAC iterative procedure.

The importance of removing redundant data in a timely manner is reflected in the avoidance of later processing during determining cluster of points that belong to the target object. During the 2D object detection shown in the previous chapter, the detected bounding box except object itself, contains pixels that mostly belong to the ground plane. If we do not do this filtering on time, then in the mapping process explained in the next chapter, we would have a situation where 3D point cloud from the LIDAR representing points on the road surface would be associated with detected bounding boxes. This would require post-processing in the clustering phase, which is a significantly more demanding process. The algorithm itself may not be able to remove all points belonging to the ground plane, but the sparsity of the remaining points will be so small that these points will be neglected within the clustering process.

E. Mapping bounding boxes to LIDAR point clouds

After the calibration we have mapped set of point clouds into camera 2D space. We are now able to observe the view of the same scene from the same perspective with data obtained from two different sensors. After passing the data from the camera through the YOLO network, we now have detected objects whose locations we want to determine based on point cloud obtained from LIDAR. We want to separate all point cloud by the bounding boxes to which they belong. Fig. 6 shows the way for doing that.

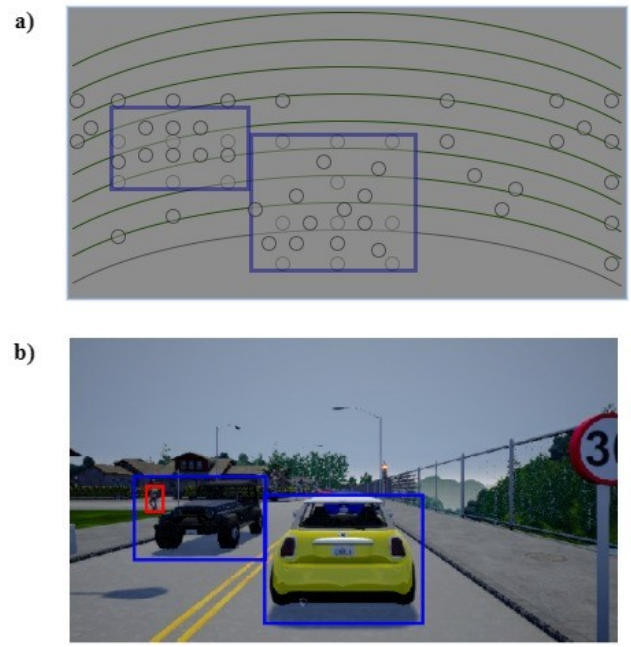

Fig. 6. a) Point cloud assigned to specific bounding box b) Bounding boxes detected on camera frame.

The next step is to divide all point cloud that belong to one of the detected bounding boxes into clusters. For each bounding box we can have a maximum of three different clusters that we divided by spatial distribution and by a coordinate that represents the depth in the image. These three possible clusters represent the background, the foreground and the object itself. However, it often happens that we do not have a foreground and therefore we cannot know in advance the number of clusters we will have. For this reason, we will use Meanshift clustering instead of K-means. Meanshift clustering will be explained in the next section. We repeat this clustering process for each of the bounding boxes. What needs to be addressed is that some of these clusters may recur among different bounding boxes. A cluster that represents the object itself within one bounding box can represent the foreground within another bounding box. After this step and after clustering that is explained in the next chapter, we have obtained clusters that represent objects of interest as well as their centroids. The only step we have left in detecting exactly the object that is closest to our vehicle. This is easily determined based on the spatial coordinates of each centroid of the found objects.

F. Clustering (Mean shift)

After we have separated point clouds of each bounding box, we want to divide this data into clusters. For that purpose, we have decided to use Meanshift algorithm, from simple reason, because it is one of the most used algorithm for clustering, where we do not need to know the number of the clusters in advance, while from instance by K-means algorithm we need to specify the number of the clusters in advance [14].

With mean shift clustering the system figures out how many clusters there ought to be and where those clusters are. Meanshift clustering is a sliding-window based algorithm that attempts to find dense areas of data points. It is centroid-based algorithm meaning that the goal is to locate the centre points of each group, which works by updating candidates for centre points to be the mean of the points within the sliding-window. These candidate windows are then filtered in a post-processing stage to eliminate near duplicates, forming the final set of centre points and their corresponding groups.

For each bounding box we take the largest cluster as an object of interest. Although for some bounding boxes this will not be the case, the only thing that matters to us is that this will certainly apply to the cluster that is closest to our vehicle. We accept the risk that we will mistakenly declare some clusters to be the certain object, but these are clusters that are not directly in front of us.

G. Tracking (Kalman filter)

After detecting the centroid for each of the clusters we classified as an object of interest, we need to determine which of these centroids belongs to cluster which represents obstacle in our lane. For simplicity in this solution it is assumed that the cluster whose centroid has the same x axis plus minus half of the lane bandwidth (observing the coordinate system of the vehicle) represents the object whose distance we want to track. As part of the solution, we decided that the centroid should be a reference point that we will monitor, because we are always sure that it belongs to the vehicle. If we decide for the nearest point within that cluster, there is always the possibility that it will be an outlier that does not belong to the vehicle itself. The centroid always deviates minimally from the very end of the vehicle, because the LIDAR detects only points on the back of the vehicle.

The centroid of the reference object represents coordinate that is passed to the input of the tracking algorithm. We use a Kalman filter [15] for this purpose. Kalman filter is used to predict what the future state of the system will be, and it is suitable for the situations where we have uncertain information about some dynamic system. The system we are looking at here is the obstacle in front of our vehicle. In the case when the model is not linear (constant acceleration), which is generally not the case in practice, an extended version of Kalman filter is used. Kalman filtering algorithm uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by using Bayesian inference and estimating a joint probability distribution over the variables for each timeframe. Let's say that "Bayesian inference" has to do with statistics. Its goal is to make predictions using all the information currently available until new information is generated. With this statement we can already get the main idea from Kalman Filters.

A significant thing with the Kalman filter is the Kalman Gain parameter. Based on the value of this parameter, it is

determined whether we give more importance to the measured or predicted values. This is significant in situations where two consecutively measured values deviate drastically. This can occur in situations where two consecutive passes through the algorithm we declare different clusters for the reference object. This can happen for a variety of reasons, but mostly in situations where our vehicle turns. These are situations where we give more priority to predicted values. The reason why this solution suits us lies in the fact that there is no need to save the complete data history, but the prediction is based only on information about the previous state, it does not require a lot of computational power, which makes it suitable for real-time problems.

IV. Evaluation

Tests were conducted in simulated environment (CARLA simulator) instead of real-world test drives because of less risk and for simplicity sake. The testing was based on driving the ego vehicle through a simulator, where different weather conditions were simulated. On the Fig. 7 we can see the visualized output of our algorithm.

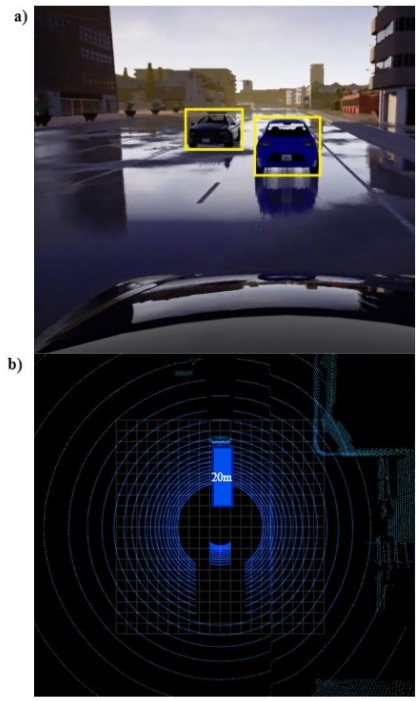

Fig. 7. a) Detected objects in camera frame. b) Estimated distance to target object obtained from LIDAR data.

Within tests, simulator ground truth values were used to validate the results obtained at the output of algorithm. (Table 1.)

Table 1. Comparison of the obtained sensors measurements and ground truth values

Measurement No.	Algorithm estimation	Ground truth values
1	40m	30m
2	28.6m	28.2m
3	25.3m	25m
4	23.9m	23.6m
5	21.5m	21.3m
6	30.5m	19.7m
7	18.7m	18.5m
8	18m	17.8m
9	17m	16.9m
10	16m	15.9m

Within the table shown, the results are given that represent the measured values based on sensory readings, without taking into account the predictions given to us by the Kalman filter. The reason for this is to point out the importance of the Kalman filter in the first and sixth measurements. At these moments, a drastic deviation from the ground truth values can be noticed. This occurs in situations where the algorithm for various reasons declares the wrong point cluster as a reference object. Thanks to the Kalman filter and its Kalman Gain parameter, the algorithm in such situations where there is a drastic deviation between successive measurements, gives a greater chance to the predicted values and takes them as reference. These may not be as accurate predictions, but they are certainly more accurate than the values measured in this case. In these situations, the Kalman filter gives us the following values (Table 2.).

Table 2. Kalman predictions for first and sixth measurements

Measurement No.	Kalman prediction	Ground truth values
1	29	30m
6	21	19.7

Mean error during these ten measurements is 0.41m, where the absolute error is 1.3m, in case of the sixth measurement where we took Kalman filter predicted value as reference distance to the obstacle in front of us. If the algorithm finds the appropriate cluster that really belongs to the required object, which was achieved in 95% of measurements during the one-hour drive, the average error is only 0.2m. which we accepted by selecting the centroid as a referent point in the clustering phase. Situations where the predicted values are taken as reference are rare and individual, so it can therefore be considered as a negligible noise.

We have tested our solution in different weather conditions and the percentage success rate of distance estimation in such situations is shown in Table 3.

Table 3. Testing of algorithm in different weather conditions.

Weather	Number of tests	Successful detections [%]
clear	20	95%
rain	20	70%
snow	20	60%
fog	20	75%

The success of the algorithm itself is largely dependent on 2D detections obtained from the input image from camera. This algorithm is designed so that if it fails to detect the object on the input frame from the camera sensor, the data from the LIDAR sensor will not even be taken into account. For this reason, this algorithm is so dependent on visibility in external conditions. In some of the extensions of this project, it is envisaged to give more autonomy to the LIDAR sensor and that in a situation where there are no

detections on the input image from the camera, LIDAR data will still be taken into account and check for any obstacles in front of the vehicle.

Hardware used in this purpose is GPU type Nvidia geforce gtx 1070 ti. This gpu allows us to process this algorithm with satisfactory speed. 2D object detection by YOLO algorithm is the module with the highest complexity and the measurement of processing speed is performed in case of execution of YOLO algorithm. The network is set to recognize 3 different classes of objects (cars, pedestrians and trucks) and the processing speed is 0.035s per frame, it is 28 fps.

V. Conclusion

The paper presents a one solution for determining the distance of an obstacle, based on fusion of camera and LIDAR data. The algorithm is based on two types of information, the frame we get from the camera and the point cloud that come from the LIDAR. We have used camera data for object detection, and LIDAR's information for determination of objects position. Based on the information about position of the nearest object in front of us, motion planning module can send commands for control the vehicle.

In the validation phase this solution showed good results with a high percentage of success in estimating the distance to the obstacle. In addition, the solution was able to meet real-time processing requirements. LIDAR is expensive sensor, but it gives accurate information about the position of an objects. It works well in almost all weather conditions, but the performance starts to degrade in the snow, fog, rain and dusty weather conditions.

There are some limitations in this implementation. The algorithm observes the space directly in front of the vehicle, but not the true trajectory of the vehicle, when determining the obstacle. It is also necessary to add time synchronization between data obtained from different sensors. Based on this, the algorithm will be able to process samples from different sensors, which originate from the same time moment with a certain threshold. Another important drawback is the high dependence on camera detection. If algorithm fails to detect the object on the frame obtained from the camera, LIDAR data will be ignored. The idea is to extend the autonomy of LIDAR data in the extension of this solution.

References

[1] "Automated Vehicels for Safety," [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety. [Accessed 15. 03. 2020.].

[2] "EURO NCAP," [Online]. Available: https://www.euroncap.com/en. [Accessed 20. 03. 2020.].

[3] ISO, "ISO 26262-10:2018 Road vehicles - Functional safety - Part 10: Guidlines on ISO 26262," ISO, 2018.

[4] "The Autoware Foundation," [Online]. Available: https://www.autoware.org/. [Accessed 21. 03. 2020.].

[5] "CARLA," [Online]. Available: https://carla.org/. [Accessed 23. 03. 2020.].

[6] Z. Zhang, Y. Wang, J. Brand, and N. Dahnoun, "Real-time obstacle detection based on stereo vision for automotive applications", 5th European DSP Education and Research Conference, Amsterdam, 2012.

[7] B. Strbac, Marko Gostovic, Zeljko Lukac, and Dragan Samardzija, "YOLO Multi-Camera Object Detection and Distance Estimation", *ZINC*, Novi Sad, 2020.

[8] M. Meyer,and G.Kuschk, "Deeplearning Based 3D Object Detection for Automitive Radar and Camera", 16th European Radar Conference, Paris, 2019.

[9] X. Zhao, P. Sun, Z. Xu, H. Min, and H. Yu, "Fusion of 3D LIDAR and Camera Data for Object Detection in Autonomous Vehicle Applications", IEEE Sensors Journal, 2020.

[10] "Geometric calibration for LIDAR-camera system fusing 3D-2D and 3D-3D point correpondences," Beijing, 2020.

[11] "Autoware Camera-LIDAR Calibration Package," [Online]. Available: https://autoware.readthedocs.io/en/feature-documentation_rtd/DevelopersGuide/PackagesAPI/sensing/autoware_camera_lidar_calibrator.html. [Accessed 15. 04. 2020.]

[12] "YOLO: Real-Time Object Detection," [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed 22. 04. 2020.]

[13] F. A. Redmon J., "YOLOv3: An Incremental Improvement," Washington, 2018.

[14] G. Seif, "The 5Clustering Algorithms Data Scientists Need to Know," [Online]. Available: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68. [Accessed 30. 04. 2020.]

[15] A. Singh, "An intro to Kalman Filters for Auonomous Vehicles," 02. 05. 2018.. [Online]. Available: https://towardsdatascience.com/an-intro-to-kalman-filters-for-autonomous-vehicles-f43dd2e2004b. [Accessed 10. 05. 2020.]