HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Object Detection and Depth Estimation by MEMS LiDAR and Camera Fusion

Scientific project in Master Mechatronics

Mrunal Sai Bandari(35075)
Aditya Deshmukh(35205)
26 July 2022

A scientific project submitted in partial fulfillment of the requirements
for the degree of Master of Science Mechatronics (MM).

Supervisor: Prof. Dr. Stefan Elser
Ravensburg-Weingarten University of Applied Sciences
Co-supervisor: Felix Berens, M.SC
Ravensburg-Weingarten University of Applied Sciences

Abstract

In the recent literature, there are many works that have focused on ways to fuse the sensor modalities of LiDAR and Camera. Inspite of various levels of fusions like Low and High levels, this project comes up with a simple yet accurate approach whose focus is more on accuracy than speed. Relying on the characteristics of Camera and LiDAR, an effective depth estimation algorithm is developed. This project involves detection of the object using object detector (YoloV3-spp) and associating it's depth information from the LiDAR. Among various types of LiDAR technologies, MEMS LiDAR whose consumption of power is lower compared to 360 degree LiDAR is used. Considering the unavailability of the real world dataset of MEMS LiDAR, dataset is generated using CARLA Autonomous Driving Simulator. The generated dataset is run through YoloV3-SPP for object detection which provides us with the bounding box information.This detection is then run through the presented algorithm which collects the depth information of all the points lying inside the corresponding bounding boxes and gives out an accurate depth of the object that is detected. Thus, enhancing the perception of the EGO vehicle by fusing the sensor modalities which compensates each other's shortcomings. This method has achieved an absolute mean error of 0.6 meter .

Declaration of Originality

Hereby we declare that this thesis is our own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and all used resources are indicated in the list of references.

Mrunal Sai Bandari & Aditya Deshmukh, Weingarten, Jul. 26th 2022

Contents

1 Introduction

Perception is important in fields like Autonomous vehicles or mobile robots, however this cannot be achieved alone with a single type of sensor because of their own shortcomings. Sensors like Camera and LiDAR are important sensor modalities for self-driving cars in particular. Each one of them has it's own advantages and disadvantages. On one hand, the image from a camera has fine-grained texture and colour information which aids in object classification but also has an inherent depth ambiguity. The LiDAR point cloud, on the other hand, provides a very accurate range view, but with low resolution and texture information. For instance in the Figure 1, the pedestrian and signpost are clearly visible in the image where as they look more or less unclassifiable in the LiDAR modality. Clearly, dependence on LiDAR alone is not a reliable approach.Hence, in this work we consider the problem of fusing the sensor modalities of MEMS LiDAR point cloud and the RGB image. By fusing information from both sensors, the idea is we could leverage the advantages of both sensors, overcoming individual limitations. Having multiple sensors on-board also allows for redundancy, which is a crucial element in safe autonomous driving in the event of sensor failure.

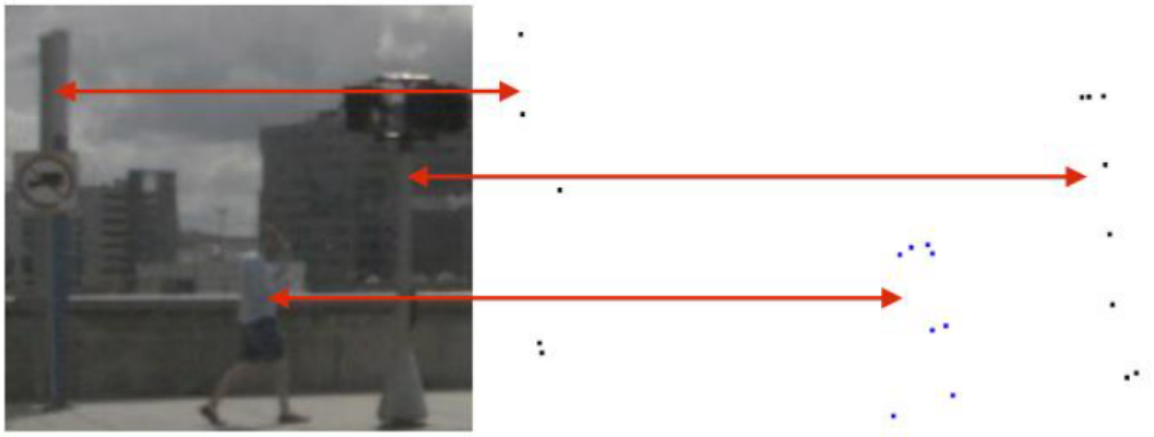

Figure 1: Example scene from the nuScenes [1] dataset.The pedestrian and pole are 25 meters away from the ego vehicle.

2 Methodology

2.1 Dataset generation

The Autonomous Driving simulator, CARLA 0.9.10 (or higher version)[UE, d] which is a python based API is used to generate dataset.CARLA exposes a powerful API that allows users to control all aspects related to the simulation, including traffic generation, pedestrian behaviors, weathers, sensors, and much more. Hence a lot of python scripting is done in order to generate the dataset. Also, the knowledge on simulating and generating dataset is gained through the python examples[UE, c] and documentation[UE, b] provided by CARLA. The steps involved here are

1. Establishment of connection between local host and CARLA
2. Spawning of actors,sensors and simulation of the world.
3. Synchronization of information from sensors with respect to world frame.

Here as the coordinates of actors in the CARLA world that are to be spawned can be predefined by the user, the actual depth of the object/actor infront of the spawned EGO vehicle is known, which is helpful later to draw a comparison between the actual and the calculated depth.When it comes to spawning of actors, each actor is searched from the carla world's blueprint library and spawned at an appropriate location. Similarly, the EGO vehicle is also spawned, the RGB Camera sensor and LiDAR sensor are attached to the EGO vehicle at position relative to the EGO vehicle's origin. However, CARLA's blueprint library doesn't contain MEMS LiDAR whereas it provides only 360 degree LiDAR sensor. Here, the extension of MEMS LiDAR for CARLA is used which is provided by "Felix Berens M.SC and Dr.Stefan Elser and Dr.Markus Reischl"[M.SC et al., 2022].

In order to achieve synchronization the mems data and camera data are saved into a queue since the start of the simulation, then for each world tick the image and mems data are retrieved and saved to disk where MEMS LiDAR point cloud data is saved as a numpy file. Thus, the world frame, camera frame and mems lidar frame are synchronized. The sample code for the dataset generation is made available at [Bandari and Deshmukh, a].
A lot sample sets under different scenarios in different orientation of car are generated (as shown in the Figure3), so that a high variance can be achieved in the data.

2.2 Mapping of MEMS LiDAR point cloud onto image

Mapping of LiDAR point cloud onto 2D image brings us back to the concept of capturing 3D world data into a 2D image in a camera.Both have the similarity of projecting 3D data onto 2D plane. To execute this, one must know the intrinsic and extrinsic matrices/parameters of the camera where extrinsic matrix is the transformation matrix which helps in translating the world's origin to camera's origin. Here, instead of world's origin it is the MEMS LiDAR origin.

Firstly, to find the extrinsic matrix we need the relative locations of MEMS LiDAR and Camera. The critical point to note is that MEMS LiDAR and camera are spawned relative

Figure 2: Sample output of RGB camera

Figure 3: Generated Dataset

to the ego vehicle's origin but not relative to the CARLA world's origin. Coming to the data obtained from camera, the data is relative to the position of the camera but not relative to the origin of the EGO vehicle whereas MEMS LiDAR point cloud data is relative to the EGO vehicle's origin but not relative to the MEMS LiDAR position.

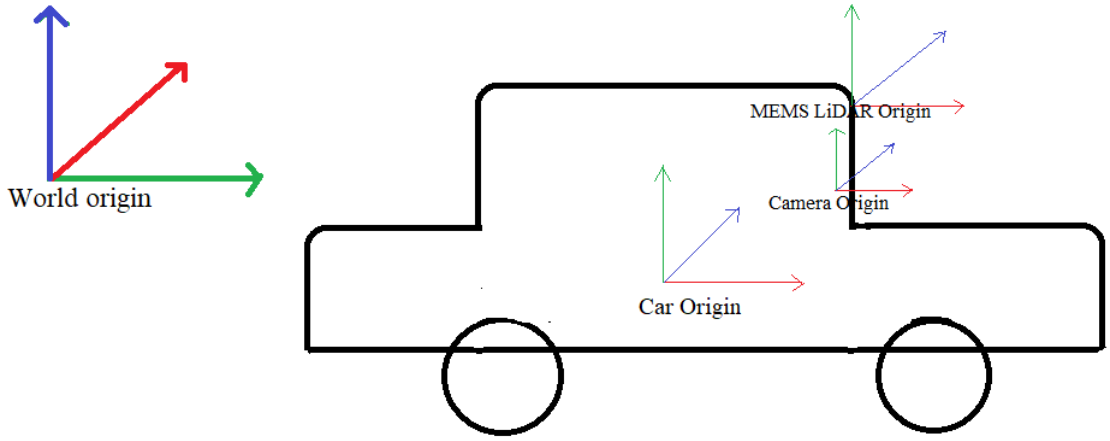

Figure 4: Figure describing origins

2.2.1 Extrinsic Matrix, transformation of coordinate systems

Hence, before finding out the extrinsic matrix the LiDAR data points must be transformed from car's origin to MEMS LiDAR position as shown in the Figure 5. The transformation matrix turns out in the form $\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ whose size is (4 x 4), whereas the size of point

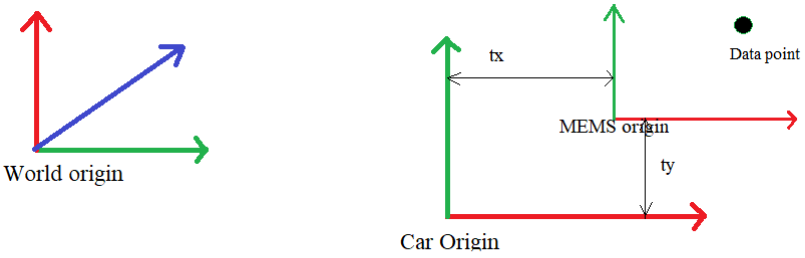

Figure 5: Translation from car origin to MEMS origin

cloud is (3 x (no.of points)). For homogeneity an extra row of ones is added to the point cloud which makes it to (4 x (no.of points)) size , so that matrix multiplication can be performed.

Now an extrinsic matrix is to be calculated which transforms the LiDAR point cloud from

MEMS origin to Camera's origin. As the relative positions of MEMS and camera are predefined by the user, a transformation matrix which transforms the data points from LiDAR coordinate system to camera coordinate system as shown in the Figure 6 can be found, which could be in the form $\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

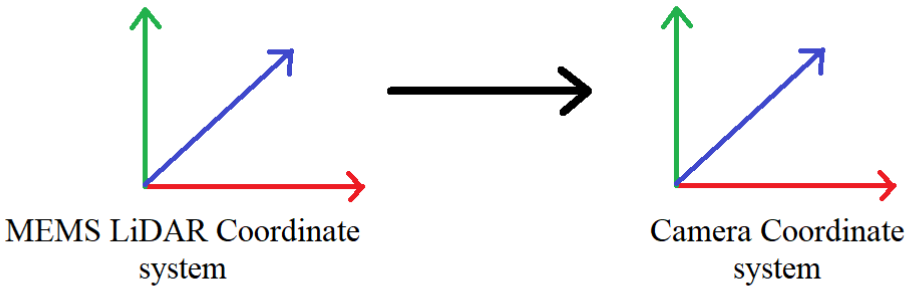

Figure 6: Transformation of LiDAR coordinate system to Camera coordinate system

To avoid calculations involved in finding an extrinsic matrix there are methods / getter functions like get_transform(), get_matrix(),get_inverse_matrix() available in CARLA'S python library which provide us with transformation matrix of size (4x 4) from the actor's position to world's origin (i.e, actor coordinate system to world coordinate system).In order to use these methods we have to follow the conversion of coordinate systems as shown in the Figure 7.

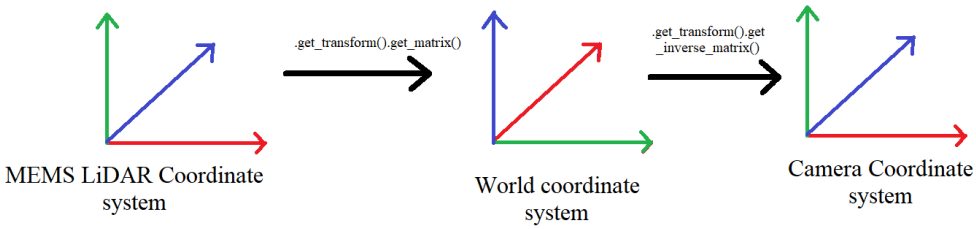

Figure 7: Transformation of coordinate systems

Firstly, we get a transformation matrix from LiDAR to world coordinate system using mems_sensor.get_transfrom().get_matrix() which on dot product with the LiDAR point cloud matrix gives the data points with respect to world origin (say lidar_2_world points). Then a inverse transformation matrix using camera_sensor.get_transfrom().get_inverse_matrix() from camera to world coordinate system is obtained, which on dot product with the lidar_2_world points gives the data points with respect to camera's origin. Thus a complete transformation of lidar points into camera coordinate system is done, however a small axes swapping is still left.

When it comes to image coordinate system, it is different from the LiDAR / camera coordinate system. For instance in the Figure 8, the corresponding axes do not point in the same direction, hence transformation of the data points must be done to achieve the LiDAR points with respect to Image coordinate system. When closely observed the transformation matrix could be just a matrix which swaps the rows (i.e, x-coordinates in the image system are the respective y-coordinates of the LiDAR points, y-coordinates in the image system are the negative of respective z-coordinates of the LiDAR points and z-coordinates in the image system are the respective x-coordinates of the LiDAR points). Hence, the transformation matrix is $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$ of size 3 x 3 which cannot be multiplied with (4 x no.of points) size point cloud, so the last row of ones that was added earlier is removed and brought into the size of (3 x no.of points).

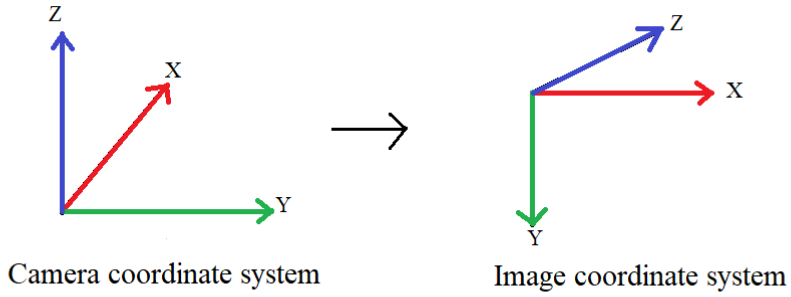

Figure 8: LiDAR to Image coordinate system

Finally the MEMS LiDAR 3D points are transformed into Image coordinate system, however the offset between the camera origin and the image origin are not yet dealt and will be solved during Intrinsic matrix formation..

2.2.2 Intrinsic matrix, conversion of 3D points to 2D

Projection of 3D points onto 2D plane takes us to pinhole camera model[Kitani,], where the mathematics of projection is seen. An intrinsic matrix looks like $\begin{bmatrix} f & 0 & t_x \\ 0 & f & t_y \\ 0 & 0 & 1 \end{bmatrix}$ where t_x and t_y are the offset values between the camera and image coordinate system as shown in the Figure 9 and 'f' is the focal length of the camera . As the offset between the image and camera coordinate systems are half of the width and height of the image, t_x and t_y are considered as (Image width/2) and (Image height/2) respectively. Focal length is calculated from the field of view of the camera from the following formula

$$focalength = \frac{Imagewidth}{2\tan(fov\frac{\pi}{360})} \tag{1}$$

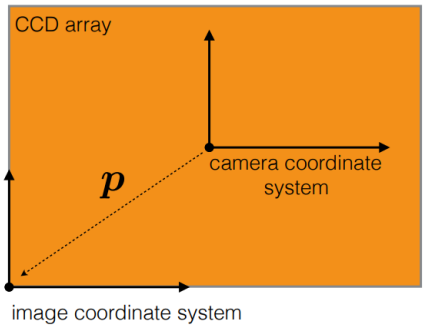

Figure 9: Camera and Image coordinate system
[Kitani,]

Now the result of the dot product of the intrinsic matrix with the already transformed data points must be normalized which exactly provides us with a completely projected 3D MEMS LiDAR points onto 2D image as shown in the Figure 10. The sample code for mapping of 3D points on to 2D image is made available at [Bandari and Deshmukh, a].

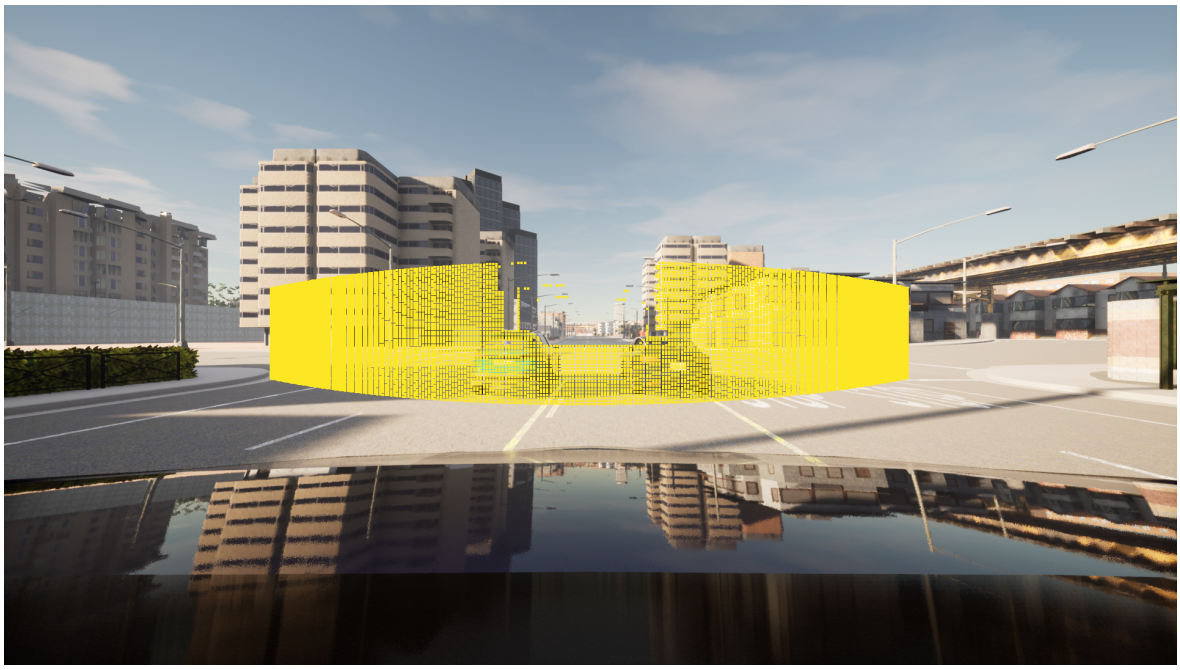

Figure 10: Camera and Image coordinate system

2.3 Object Detection

In general, the methods of object detection based on deep learning are divided into two categories, the two-stage method and the one-stage method. The two-stage object detection method is also called the region-based object detection method and the classic models include regions with CNN features (R-CNN)[Ross Girshick,], spatial pyramid pooling network (SSP-Net), fast R-CNN[Girshick,], faster R-CNN, multi-scale CNN (MS-CNN) and subcategory-aware CNN (SubCNN). Though the above methods have high detection precision, the detection speed is slow and cannot meet the real-time requirements.Hence, the

one-stage object detection method emerged to improve the detection speed. It obtains the prediction results directly from the image without the need to generate a region proposal. Although the detection precision is reduced, the entire process requires only one step, which dramatically shortens the detection time and realizes real-time detection.

Hence a one stage method which is You only look once (YOLO), a state-of-the-art real-time object detection system, is used. The version YOLOv3[Joseph Redmon, a] is fast and accurate compared to the previous versions. Based on weights and configurations there are several models in YOLOv3[Joseph Redmon, b], out of which YOLOv3-spp model is chosen whose mAP(mean average precision) is the highest, which is 60.6. However, it comes with a down side of lower FPS which is 20.

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections, whereas YOLOv3 uses a different approach which applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.YOLO has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN[Joseph Redmon, b].

We implement the pre-trained YOLOv3 network in opencv using darknet module. Corresponding spp model weights and configuration files are downloaded from [Joseph Redmon, b]. The obtained sample output from RGB camera, as shown in figure 2 is passed onto the implemented YOLOv3 network in opencv and hence the bounding box information, object detection as shown in the Figure 11 are extracted and performed

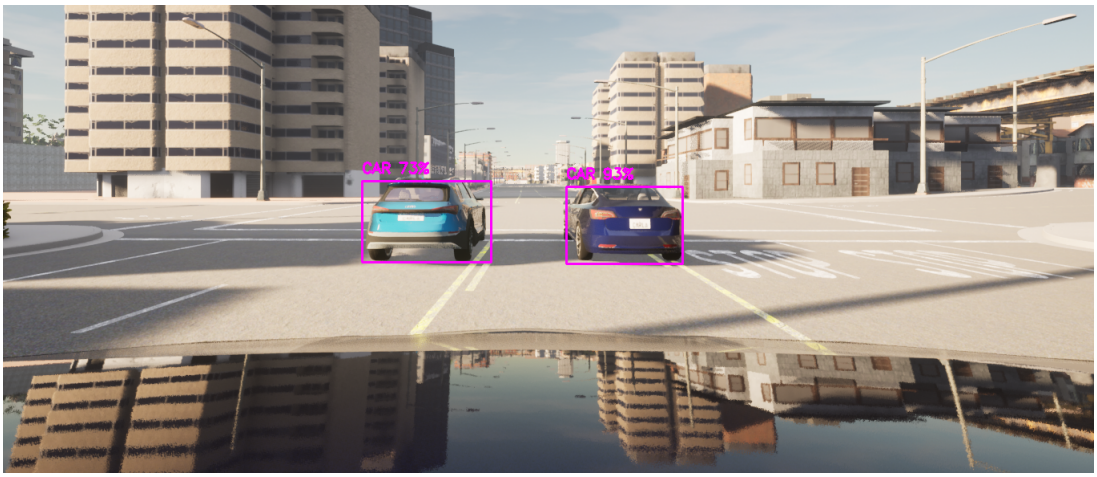

Figure 11: Sample image of object detection

2.4 Depth Estimation

The task here is to gather the depth information associated with each point which lie inside the bounding box and calculate an accurate depth. For instance in the Figure 12, visualization of points inside a bounding box is shown. The main drawback here is, apart from the points on the car there are also outliers which are inside the bounding box. Consideration of these outliers will drastically deviate the overall calculated depth. Hence, an algorithm called RANSAC is implemented to eliminate the outliers. RANSAC is capable of interpreting smoothing data containing a significant percentage of gross errors, and is thus ideally suited for applications where interpretation is based on the data provided by error-prone data[Fischler and Bolles,].

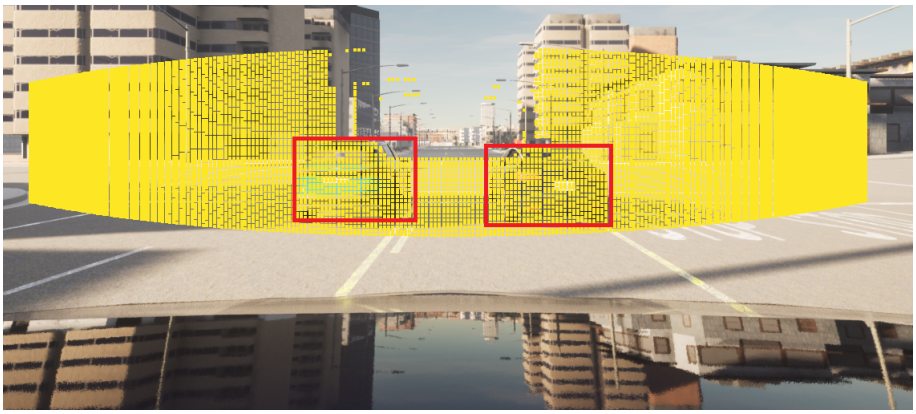

Figure 12: Visualization of points inside the Bounding Box

Now that the bounding box information [x, y, width, height] is obtained as mentioned previously, list out the points which are inside bounding box i.e, points whose $(x \leq x{-}coordinate \leq x + width)\&(y \leq y - coordinate \leq y + height)$. These list of points are passed onto the RANSAC algorithm. In order to eliminate the outliers whose error margin is way higher, we have applied a tight residual threshold of 1 and hence an appropriate interpretation of data is achieved as shown in the example Figure 13.

The inliers data obtained is stored into list. Here a statistical approach is required to draw out the accurate depth of the object out of several depths from the stored list. Among the average, median and minimum methods, considering the least depth point out of all the points has lead to a most accurate result. For this, the sample code is made available at [Bandari and Deshmukh, b].

3 Evaluation and Results

The obtained depth from the above algorithm is compared with the actual depth. In real, we get a spawned depth at which two car are spawned in carla. However, the spawned depth is the distance between the car origins. So, we have to find the actual depth of the spawned vehicle till the back of the car. For this we need the physical dimensions of the car's that are spawned. The process of finding the actual depth is as depicted in the Figure 14.

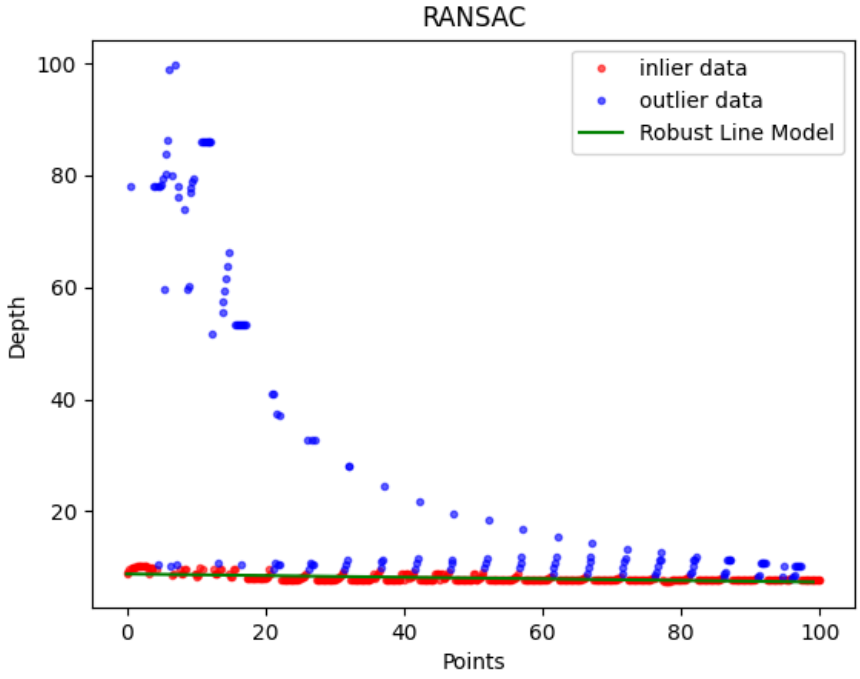

Figure 13: RANSAC output

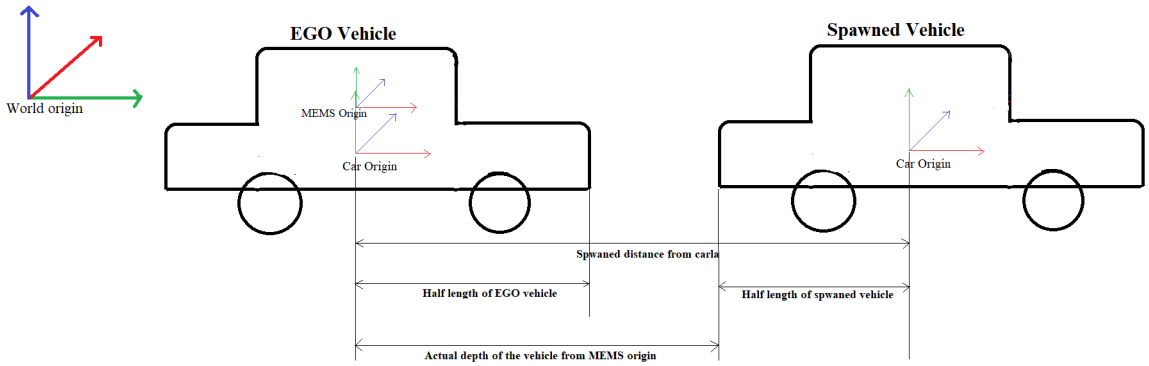

Figure 14: Actual depth calculation

$$Actualdepth = Spawndepth - (halflengthofspawnedvehicle) \tag{2}$$

where spawn depth is the distance between the two car origins. As shown in the Figure 14, placement of MEMS sensor plays crucial role in actual depth calculation. In our case, as the MEMS sensor is co-linear with the car's origin we do not want to eliminate the half length of the EGO vehicle in the actual depth calculation.As spawn depth is predefined by the user, the unknown here is the half length of the spawned vehicle.

Each actor in carla is fenced by a bounding box as shown in the Figure15. The origin's

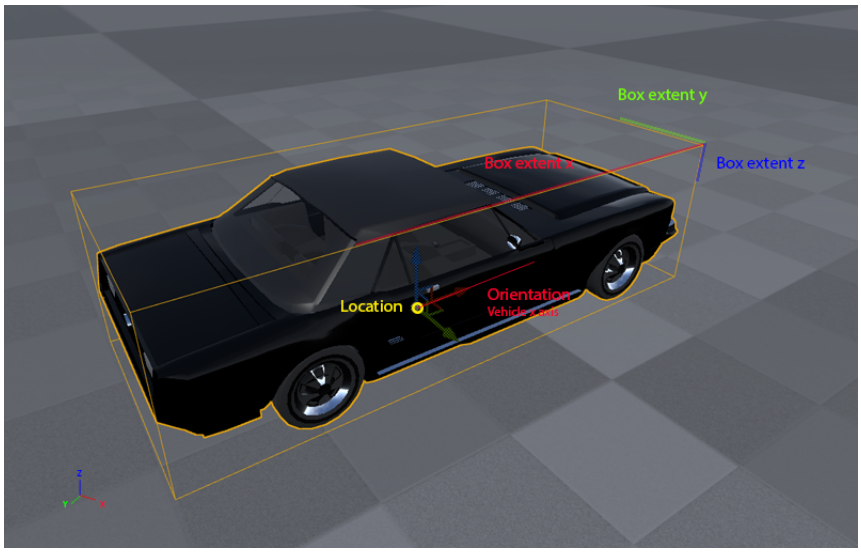

Figure 15: Bounding box of an actor in carla
[UE, a]

location and the extent of the bounding box can be obtained. The extent values give the half lengths of the vehicle. Similar to the sample output shown in Figure 2, several different samples are generated to calculate the error in the depth estimation (i.e, the different between actual depth and the estimated depth).

$$Error = Estimateddepth - Actualdepth \tag{3}$$

A positive error indicates that the estimated depth is greater than the actual depth and a negative error implies the converse. The mean and absolute mean of the errors are calculated using the formulae

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N}(x_i) \qquad \& \qquad \overline{x} = \frac{1}{N}\sum_{i=1}^{N}(|x_i|) \tag{4}$$

where,
N = Size of sample, $\overline{x} = Meanofthesample, x_i$ = Each value from the sample

which are 0.54 and 0.6 respectively. Also, the standard deviation is calculated using the

formula

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{5}$$

where,
s = Standard deviation, N = Size of sample, $\overline{x}$= Mean of the sample, x_i = Each error value from the sample

which is 0.71. The distribution of error range is as shown in the Figure 16. Now, for an error

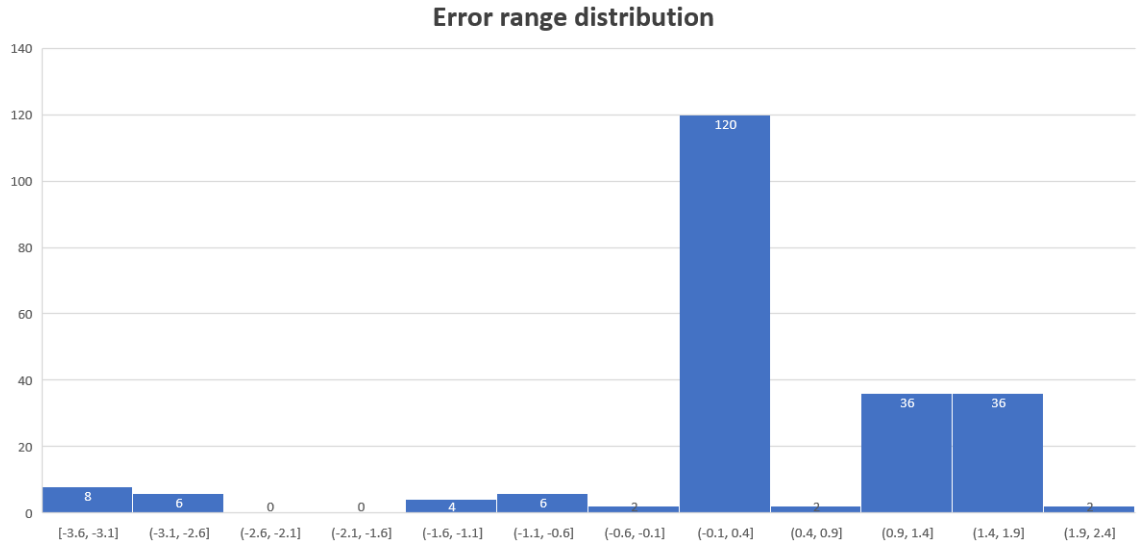

Figure 16: Error graph

slab of -0.5 to + 0.5 meter a probability of 55% is obtained. This implies that the algorithm can provide a depth estimation with in a tolerance +/- 0.5 meter with a probability of 55%. This is calculated by taking a percentage ratio of no.of values in the slab of -0.5 to 0.5 meter to the total no.of values.

4 Conclusion

There are many levels of fusion where complexity, accuracy as well as the speed are high. Here this work provides us with a simple algorithm to estimate the depth of the object by the fusion of MEMS LiDAR and camera. Implementation of RANSAC and the usage of YOLOv3-spp model with corresponding weights and configuration have contributed a lot in achieving lesser error margin. The achieved mean error is 0.54 and the absolute mean error is 0.6 with a standard deviation of 0.71. Also, the probability that the estimated depth lies with in an error margin of +/- 0.5 meter is 55%.The complete code for this work is provided at [Bandari and Deshmukh, c].

References

[Bandari and Deshmukh, a] Bandari, M. S. and Deshmukh, A. Carla dataset generation. https://github.com/MrunalSai/Object-detection-and-depth-estimation-by-MEMS-LiDAR-and-camera-fusion/blob/main/Dataset_generation.py. 2, 7

[Bandari and Deshmukh, b] Bandari, M. S. and Deshmukh, A. Depth estimation. https://github.com/MrunalSai/Object-detection-and-depth-estimation-by-MEMS-LiDAR-and-camera-fusion/blob/main/Depth_estimation.py. 9

[Bandari and Deshmukh, c] Bandari, M. S. and Deshmukh, A. Object detection and depth estimation by mems lidar and camera fusion. https://github.com/MrunalSai/Object-detection-and-depth-estimation-by-MEMS-LiDAR-and-camera-fusion. 13

[Fischler and Bolles,] Fischler, M. A. and Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. https://dl.acm.org/doi/pdf/10.1145/358669.358692. 9

[Girshick,] Girshick, R. Fast-rcc. https://arxiv.org/pdf/1504.08083.pdf. 7

[Joseph Redmon, a] Joseph Redmon, A. F. Yolov3. https://pjreddie.com/media/files/papers/YOLOv3.pdf. 8

[Joseph Redmon, b] Joseph Redmon, A. F. Yolov3code. https://pjreddie.com/darknet/yolo/. 8

[Kitani,] Kitani, K. Camera matrix. https://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf. 6, 7

[M.SC et al., 2022] M.SC, F. B., Elser, D., and Reischl, D. (2022). Mems class. https://github.com/BerensRWU/MEMS-LiDAR-Generator. 2

[Ross Girshick,] Ross Girshick, Jeff Donahue, T. D. J. M. Rcc. https://arxiv.org/pdf/1311.2524.pdf. 7

[UE, a] UE, C. Carla docs. https://carla.readthedocs.io/en/0.9.5/measurements/. 11

[UE, b] UE, C. Carla documentation. https://carla.readthedocs.io/en/latest/. 2

[UE, c] UE, C. Carla python examples. https://github.com/carla-simulator/carla/tree/master/PythonAPI/examples. 2

[UE, d] UE, C. Carla releases. https://github.com/carla-simulator/carla/releases. 2

List of Figures