Vehicle Pose Estimation using Camera and LiDAR fusion in the CARLA Simulator

Parth Kadav[1]

***Abstract*— Accurate pose estimation of objects in the ego-vehicle's environment is necessary for safe vehicle navigation. Measurements coming in from different sensors need to be synchronised and fused in order to estimate position of various objects in the ego-vehicle's surrounding. For our study, we propose three different sensor fusion methods to fuse Camera sensor, and Light Imaging and Ranging (LiDAR) sensor data obtained using the CARLA simulator to estimate the position of a lead vehicle. The three fusion methods developed were Group, Inverse Covariance Form (ICF), and Regression Based Fusion. We modeled methods 1 and 2 using the principles of a linear Kalman Filter and method 3 using Machine Learning techniques. We used a constant velocity motion model. Our process model was observable and linear. The measurements were synchronous at all timesteps. We obtained a RMSE of 2.14m on our group fusion, 2.15m on ICF, 0.3m on Linear Regression Fusion, and 1.76m on Artificial Neural Network (ANN) Based Regressor. We showed successful pose estimation for the lead vehicle using the three different fusion techniques. Additionally, evaluated a new way of fusing measurements using regression based fusion.**

I. INTRODUCTION

This study develops a sensor-fusion technique for vehicle position estimation using the Car Learning to Act (CARLA) simulator. We developed hree sensor fusion methods to estimate the lead vehicle's position using camera and LiDAR fusion. CARLA simulates real-world sensory data. Image and point cloud messages from the camera sensor and LiDAR were obtained through the simulator using ROS. The lead vehicle's position is predicted using Kalman Filtering. Group, Inverse Covariance, and Regression-Based Sensor Fusion methods are developed. The methodology section covers data preparation, Camera detections in the CARLA simulator, and LiDAR Detections in the CARLA simulator, Process and Measurement Models, and Sensor Fusion Methods. The results section will analyze the performance of the various sensor fusion techniques, while the conclusion will provide a summary of this work.

II. METHODOLOGY

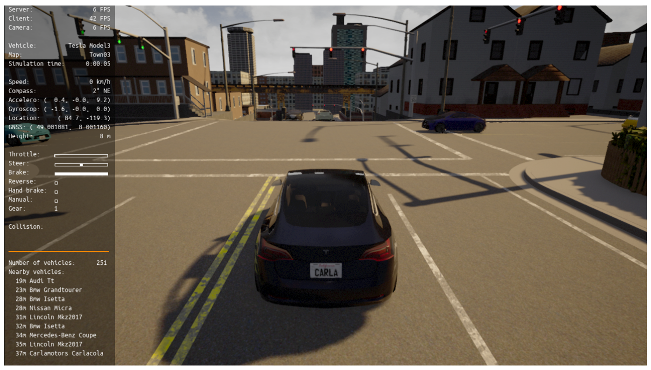
In this section, we will first discuss the methods we used to collect and prepare the data from the simulator. We will then describe the process model, the measurement model and the three different sensor fusion methods in detail.

*This work was not supported by any organization

[1]Parth Kadav is a PhD student with the Mechanical and Aerospace Engieering Department, Western Michigan University, Kalamazoo, MI, USA
parth.kadav@wmich.edu

A. Data Preparation

CARLA is an open-source simulator designed to train, validate, and test algorithms for Autonomous Vehicle (AV) applications. It provides a simulation environment with numerous sensor specifications, environmental conditions, vehicles, etc. Instances of such situations include object detection using computer vision, vehicle localisation, environmental mapping, vehicle control, and path planning. CARLA enables the user to access the information ROS. CARLA has also been utilized to train and test neural network models for camera and LiDAR object detection, making it an excellent option for our application [1]. Figure 1 shows the CARLA simulator. CARLA offers different maps which have different landscapes. The simulator also allows the user to spawn different objects such as vehicles, sensors and objects.


```
Server:                 6 FPS
Client:                42 FPS
Camera:                 6 FPS

Vehicle:          Tesla Model3
Map:                    Town03
Simulation time:       0:00:05

Speed:                  0 km/h
Compass:                 2° NE
Accelero: (  0.4, -0.0,  9.2)
Gyroscop: ( -1.6, -0.0,  0.0)
Location:       ( 84.7, -119.3)
GNSS: ( 49.001081,  8.001160)
Height:                    8 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:        1

Collision:

Number of vehicles:      251
Nearby vehicles:
  19m Audi Tt
  23m Bmw Grandtourer
  28m Bmw Isetta
  28m Nissan Micra
  31m Lincoln Mkz2017
  32m Bmw Isetta
  34m Mercedes-Benz Coupe
  35m Lincoln Mkz2017
  37m Carlamotors Carlacola
```

Fig. 1. CARLA simulator with the vechicle control panel

We were only interested in collecting Camera and LiDAR data from the ego vehicle using the simulator for this study. The experimental setup included a lead vehicle and an ego vehicle that followed it. Through the simulator, the lead vehicle was set to autopilot. We followed the lead vehicle by varying the speed of the ego vehicle manually. In order to estimate the pose of the lead vehicle, we must first obtain its position using the two sensors independently. To obtain any type of data from the simulator, we must establish bidirectional communication between the simulator and ROS. The CARLA ROS Bridge allows us to obtain data from the CARLA simulator. Before moving on to the section on data collection, let us examine briefly how ROS operates. Nodes are the foundation of the ROS architecture, a ROS Node can either function as a Publisher or a Subscriber. A Publisher is the entity responsible for specifying messages of a standard message type, such as integers, strings, and images, to a specific topic. In contrast, the Subscriber subscribes to the topic and receives messages whenever they are published from the topic. The figure below depicts a simple illustration of the publisher and subscriber nodes. Figure 2 shows a simple illustration of the publisher and subscriber nodes.

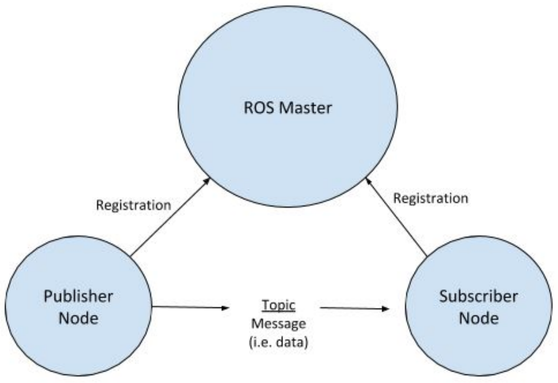


Fig. 2. A simple ROS publisher and subscriber architecture. Image taken from [2]

The raw camera output is one such topic that the simulator publishes. We can subscribe to this image topic and apply object detection to determine the position of the lead vehicle. We implemented real-time object detection in ROS using the You Only Look Once (YOLOv3) algorithm to detect the position of the lead vehicle. YOLO is a real-time, cutting-edge object detection algorithm. The YOLO algorithm is a convolutional neural network that has been pre-trained on 80 distinct classes, including vehicles, people, traffic signs, animals, and others [3].

B. *Camera Detection using the CARLA simulator*

The raw camera output is an image with dimensions $(512$ x $512)$. The image message for the camera topic obtained through ROS is set at $20Hz$. The output of the YOLO object detection algorithm is an image containing the detected bounding box of the lead vehicle. Figure 3 depicts the detection of the leading vehicle within the bounding boxes. To determine the center of the detected object, or in this case, the center of the lead vehicle's position in pixel coordinates, we should compute the center of the bounding box. As the camera detections do not contain depth information, the computed center would be at the rear of the lead vehicle.

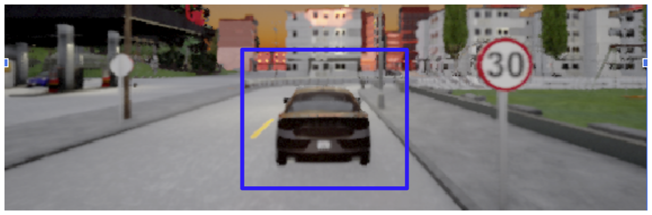


Fig. 3. Lead vehicle detected in the CARLA simulator using YOLOv3 algorithm

As the sensors are positioned in different places on the vehicle, we need to transform these sensors to the same frame or coordinate system in order to apply fusion tehniques. We will transform both the camera and the LiDAR to the world frame. We could also transform everything to the camera frame or the LiDAR frame, there are no set rules to follow when performing transformations as long as the measurements from the differnt sensors are in the same coordinate frame. Let us look at transforming the measurements from the camera to the world frame. Figure 10 in the appendix shows the transformation from world coordinates to image coordinates. Given $o_{image} = (u, v, 1)$ which is the pixel location for the center of the bounding box, we can compute the coordinates for the object in the camera frame C. This can be done by using Equations 1 and 3.

$$o_{image} = \begin{bmatrix} x_{img} \\ y_{img} \\ z_{img} \end{bmatrix} = K \left[R|t\right] \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix} \tag{1}$$

Equation 1 shows the conversion from World Coordinates $\rightarrow$ Image Coordinates, where K is the camera intrinsic matrix which models the internal camera parameters. Equation 2 is the rotational matrix along with the translational vector to model any transformation between a world coordinate frame and another. $a_{11}..a_{33}$ are the elements related to the rotations, $b_1..b_3$ denote the translational vector.

$$[R|t] = \begin{bmatrix} a11 & a12 & a13 & b1 \\ a21 & a22 & a23 & b2 \\ a31 & a32 & a33 & b3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Equation 3 shows the conversion from the Pixel Coordinates $\rightarrow$ Image Coordinates and vice-versa, where $\begin{bmatrix} u & v & 1 \end{bmatrix}$ are the pixel coordinates of the center of the detected vehicle.

$$\begin{bmatrix} x_{img} \\ y_{img} \\ z_{img} \end{bmatrix} \rightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} \begin{bmatrix} x_{img} \\ y_{img} \\ z_{img} \end{bmatrix} \tag{3}$$

Note that the pixel coordinates (u, v) are measured from the top left corner and the image coordinates (X, Y) are measured from the bottom left corner. The "Depth Image" topic publishes the depth Z of the image at each pixel (X, Y). We follow the North-East-Down (NED) convention as shown in Figure 4 to carry out the frame transformation.

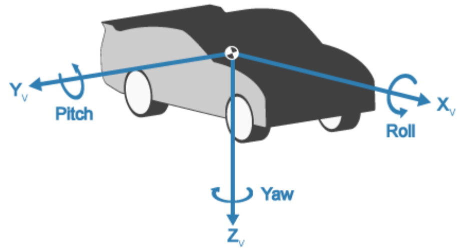


Fig. 4. North East Down Frame in a vehicle. Image taken from [5]

Given the above equations and the pixel location of the center of the detected object, we can carry out the following conversions to go from pixel coordinates to world coordinates.

1) Convert the pixel coordinates to image coordinates using Equation 4

$$\begin{bmatrix} x_{img} \\ y_{img} \\ z_{img} \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} * (z_{img}) \tag{4}$$

2) Convert the image coordinates to camera coordinates

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}^{-1} K^{-1} \begin{bmatrix} x_{img} \\ y_{img} \\ z_{img} \end{bmatrix} \tag{5}$$

where : $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$ is a transformation matrix to convert from image coordinates to the NED coordinate system as shown in Figure 4

3) Get the transformation matrix to convert between camera frame and world frame $[R|t]$

4) Get vehicle coordinates in the world frame with Equation 6

$$\begin{bmatrix} x_{map_{cam}} \\ y_{map_{cam}} \\ z_{map_{cam}} \end{bmatrix} = [R|t] \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{bmatrix} \tag{6}$$

Now that we have the coordinates of the lead vehicle in the world frame $(x_{map_{cam}}, y_{map_{cam}}, z_{map_{cam}})$, we can get the position of the lead vehicle using the LiDAR in the world frame.

C. LiDAR Detection using the CARLA simulator

To detect the position of the lead vehicle using the LiDAR we can use the "Semantic Lidar" topic published from the simulator which outputs the point clouds for the various objects in the ego vehicle's surroundings. We can obtain the cluster for the lead vehicle and then compute the centroid from the points to obtain one point $(x_{lead_{lidar}}, y_{lead_{lidar}}, z_{lead_{lidar}})$ where the centroid can be computed using Equation 7

$$\begin{bmatrix} x_{lead_{lidar}} \\ y_{lead_{lidar}} \\ z_{lead_{lidar}} \end{bmatrix} = \begin{bmatrix} mean(x_{cluster}) \\ mean(y_{cluster}) \\ mean(z_{cluster}) \end{bmatrix} \tag{7}$$

In order to get the position of the lead vehicle in the world frame $(x_{map_{lidar}}, y_{map_{lidar}}, z_{map_{lidar}})$,we need to multiply it with the transformation matrix which includes the rotation and translation of the sensor in the world frame as shown in Equation 8.

$$\begin{bmatrix} x_{map_{lidar}} \\ y_{map_{lidar}} \\ z_{map_{lidar}} \end{bmatrix} = [R|t] \begin{bmatrix} x_{lead_{lidar}} \\ y_{lead_{lidar}} \\ z_{lead_{lidar}} \\ 1 \end{bmatrix} \tag{8}$$

Figure 5 shows the lead vehicle with the red marker. The red marker shown for visual representation depicts the centroid of the cluster $(x_{map_{lidar}}, y_{map_{lidar}}, z_{map_{lidar}})$ for the lead vehicle.

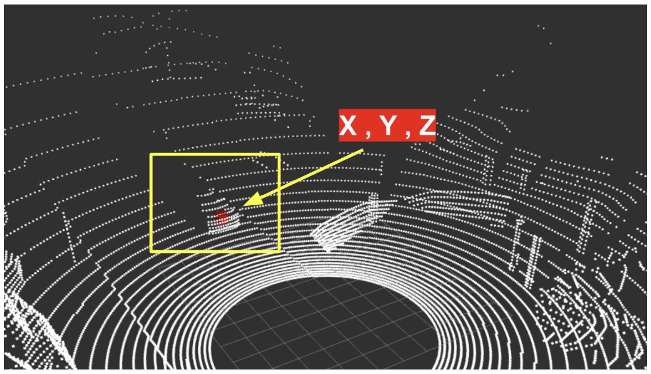


Fig. 5. Lead vehicle pose in the world frame visualized using ROS visualiztion tools. The (X, Y, Z) shown is the centroid computed from the semantic LiDAR clustering in the world frame

Now that we have the detected position of the lead vehicle for both the camera $(x_{map_{cam}}, y_{map_{cam}}, z_{map_{cam}})$ and the LiDAR $(x_{map_{lidar}}, y_{map_{lidar}}, z_{map_{lidar}})$ in the world frame we can apply fusion techniques. For fusion we need to first define the process and the measurement model. These are described in the next sections.

D. Kalman Filter Based Estiamtors

The Kalman filter is an optimal estimator based on a recursive computational methodology which uses a series of measurements observed over time, including noise and other inaccuracies, and produces estimates of the unknown variables. In our study, estimation of the lead vehicle position at each timestep is done using the measurements received from the camera and the LiDAR. The simulator uses ROS to synchronize all the sensors so we will have synchronous measurements at each timestep.

E. Process Model

The process model is a linear time invariant system which is subject to random process noise $w(k)$ as shown in Equation 9. We would be using a constant velocity process model. A constant velocity model is useful to model smooth target motion. A, B, and G denote the system matrix, input matrix and process noise gain matrix for the process model. The process model is observable, linear and synchronous at all ΔT.

$$x(k+1) = Ax(k) + Bu(k) + Gw(k) \tag{9}$$

The linearised process model shown in Equation 10 shows the linearised system matrix and the process noise gain matrix. There is no control input to the system so we do not include the input matrix. The process model noise $w(k)$ with dimensions $\in \begin{bmatrix}3x1\end{bmatrix}$ would be associated with the acceleration of the vehicle which is going to be white gaussian in nature (0 mean and given standard deviation). The state vector is $\begin{bmatrix}x & v_x & y & v_y & z & v_z\end{bmatrix}^T$.

$$\begin{bmatrix}x_{k+1}\\v_{x(k+1)}\\y_{k+1}\\v_{y(k+1)}\\z_{k+1}\\v_{z(k+1)}\end{bmatrix} = \begin{bmatrix}1 & \Delta T & 0 & 0 & 0 & 0\\0 & 1 & 0 & 0 & 0 & 0\\0 & 0 & 1 & \Delta T & 0 & 0\\0 & 0 & 0 & 1 & 0 & 0\\0 & 0 & 0 & 0 & 1 & \Delta T\\0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_k\\v_{x(k)}\\y_k\\v_{y(k)}\\z_k\\v_{z(k)}\end{bmatrix} + \\ + \begin{bmatrix}0.5\Delta T^2 & 0 & 0\\\Delta T & 0 & 0\\0 & 0.5\Delta T^2 & 0\\0 & \Delta T & 0\\0 & 0 & 0.5\Delta T^2\\0 & 0 & \Delta T\end{bmatrix} w(k) \tag{10}$$

F. Measurement Model

A general measurement model can be represented by Equation 11. The H matrix is the sensor measurement model matrix and $v(k)$ is the measurement noise associated with the sensors itself. The measurement noise is gaussian noise with 0 mean and given standard deviation

$$z(k) = Hx(k) + v(k) \tag{11}$$

We have two measurement models each corresponding to the two sensors as shown in Equation 12 and 13

$$z_1(k) = H_1x(k) + v_1(k) \tag{12}$$

$$z_2(k) = H_2x(k) + v_2(k) \tag{13}$$

As shown in Equation 14 and 15, we can linearise the measurement models at each ΔT for the camera $z_1(k)$ and the LiDAR $z_2(k)$to obtain the measurement model matrix. As we only obtain the position of the lead vehicle from the sensors, both the H_1 and H_2 matrix only computes the x, y, and z at each timestep. The measurement vector is $\begin{bmatrix}x & y & z\end{bmatrix}^T$.

$$z_1(k) = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\\0 & 0 & 1 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x_k\\v_{x(k)}\\y_k\\v_{y(k)}\\z_k\\v_{z(k)}\end{bmatrix} + v_1(k) \tag{14}$$

$$z_2(k) = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\\0 & 0 & 1 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x_k\\v_{x(k)}\\y_k\\v_{y(k)}\\z_k\\v_{z(k)}\end{bmatrix} + v_2(k) \tag{15}$$

G. Sensor Fusion Methods

We will now discuss the three different sensor fusion methods which were applied to the camera and LiDAR measurements to get the lead vehicle pose in this section. We use Group Fusion, Inverse Covariance Fusion, and Regression-Based Fusion for this study.

1) Method 1: Group Fusion: The group sensor method is based on synchronous measurements and combines the measurements from all the sensors into a single sensor. Some requirements for this method are that all sensors need to observe the same state and the process model must be common between sensors. Additionally, the group fusion method is practical to implement for a small number of sensors. The measurement model for the linear time invariant system for the group fusion method is shown in Equation 16.

$$\begin{bmatrix}z_1(k)\\z_2(k)\end{bmatrix} = \begin{bmatrix}H_1(k)\\H_2(k)\end{bmatrix}\begin{bmatrix}vector1\\vector2\end{bmatrix} + \begin{bmatrix}v_1(k)\\v_2(k)\end{bmatrix} \tag{16}$$

where $vector1$ and $vector1$ are the measurement vectors for sensors 1 and sensor 2 respectively. The measurements, sensor measurement model matrices, and the measurement noise are stacked as shown in Equations 17, 18, 19.

$$z(k) = \begin{bmatrix}z_1^T(k), z_2^T(k)\end{bmatrix}^T \tag{17}$$

$$H(k) = \begin{bmatrix}H_1^T(k), H_2^T(k)\end{bmatrix}^T \tag{18}$$

$$v(k) = \begin{bmatrix}v_1^T(k), v_2^T(k)\end{bmatrix}^T \tag{19}$$

We receive two measurements from the camera, and the LiDAR, which are $z_1(k)$ and $z_2(k)$ respectively. We stack the two measurements along with the sensor measurement model matrices, (H_1 and H_2) and the measurement noise ($v_1(k)$ and $v_2(k)$) associated with the two sensors using Equations 17-19.We can perform group fusion using the 5 steps of the Kalman Filter for state estimation using Equations 23 - 27 the Appendix section.

2) Method 2: Inverse Covariance Fusion: ICF method uses the standard Kalman Filter expressions for state prediction (Equation 24) and the prediction error covariance matrix (Equation 23) as these steps do not depend on the measurements. After computing the Kalman Gain (Equation 25), we modify the state estimate equation to create linear composite correction. We also modify the estimate error covariance calculation (Equation 25). So to summarize, we compute the prediction error covariance matrix $M(k)$, state predicition $\overline{x}(k|k-1)$, and Kalman Gain $K(k)$ using the standard Kalman filter (Equation 23-26) shown in Appendix. We compute the estimate error covariance matrix $P(k)$ and state estimate $\hat{x}(k|k)$ (steps 4 and 5) using Equation 20 and 21

$$P(k) = \begin{bmatrix}M^{-1}(k) + \sum_{i=1}^{S} H_{i,k}^T V_i^-1(k) H_{i,k}\end{bmatrix}^{-1} \tag{20}$$

$$\hat{x}(k|k) = P(k) \begin{bmatrix} M^{-1}(k)\overline{x}(k|k-1) + .. \\ .. + \sum_{i=1}^{S} H_{i,k}^{T} V_i^{-}1(k) z_i(k) \end{bmatrix} \tag{21}$$

where S = number of sensors

ICF has many advantages over group fusion, such as the largest matrix inversion has the dimensions of the state vector and addition of new sensors only requires addition of new terms in Equation 20 and 21. Computational complexity grows only linearly with the number of sensors used. These advantages are only obvious when there is a significant increase in the number of sensors used.

3) Method 3: Regression Based Fusion: The measurements obtained from the two sensors can be used in regression-based machine learning techniques to predict the lead vehicle's position. Regression is a machine learning method for determining the relationship between independent variables (features) and dependent variables (target). We can use regression to forecast continuous outcomes. In predictive analysis, regression models can be used to represent the trends between different features [6], [7]. Our measurements from the two sensors are synchronous and are independent of each other. We can apply regression techniques between the targets from the two sensors and find out a relation if there exists any. The regression models can be trained on the measurements. Our measurements $z_1(k)$ and $z_2(k)$ can be the features. The ground truth (x_{gt} , y_{gt} , z_{gt}) for the lead vehicle which is obtained through the simulator is the target. We would perform multi-variable regression because we have 6 features. A few assumptions are that the features should be measured at continuous time levels and the observations should be independent of each other which is true for our measurements.

a) Linear Regression: In linear regression, the parameters of the linear predictor functions used to model the relationships are calculated from the features. To fit a predictive model to an observed set of targets given its features, linear regression can be used if the objective is prediction or forecasting [7]. Equation 22 shows a general form for multi-variable linear regression, where all observations are indexed as i and all dependent variables are indexed as j. X_{ij} is the i^{th} observation of the j_{th} independent variable. β are the parameters needed to be estimated. ϵ is the independent identically distributed normal error. We use the scikit-learn library in python to model our linear regressor [8]. We did a 65 - 35 % split on the dataset, which were the measurements and the ground truth. 65 % data was used for training and 35 % was used for testing.

$$Y_ij = \beta_{0j} + \beta ij X_{i1} + \beta_{2j} X_{i2} + ... + \beta_{pj} X_{ip} + \epsilon_i j \tag{22}$$

b) Artificial Neural Network Based Regression: An Artificial Neural Network based Regressor can perform much better at regression tasks as it can model complex non-linear relationships between the features and the target whereas linear regression cannot learn these relationships very well [9]. We use the tensorflow's keras to make our ANN based regressor [10]. Figure 11 in the appendix shows the model setup and summary. We use the sequential model class with 5 layers. The first layer is a dense layer which consists of 128 hidden units and 6 input dimensions, which relate to the 6 input features to model, which are the measurements from the two sensors. The second layer and the third layer consist of 64 and 32 hidden units. The fourth layer is a dropout layer which randomly shuts down a specified number of neurons to reduce the chance of overfitting. Finally we have the last dense layer which has 3 units that relate to the fused measurements as the outputs from the model. We chose loss function as mean squared error and the optimizer as Adaptive moment estimation (Adam). We can also test the model on metrics such as accuracy and mean-squared-error. The model reference code is shown in the appendix.

III. RESULTS

To evaluate the position estimators developed in the methodology section, we collected the ground truth vehicle position data from the simulator. Three different sensor fusion methods were modeled. The sensor update rate for both the sensors was kept at constant 20 Hz for all fusion methods. Measurements from the sensors and ground truth are plotted in Figure 6 in the world frame. We calculated the root mean square error (RMSE) for the measurements with respect to the ground truth which was 0.78m, 0.76m in (X, Y) for the camera, and 0.59m, 1.12m in (X, Y) for LiDAR. Estimates in the Z direction can be excluded as they are not useful in autonomous vehicle applications as the vehicle has restricted motion in the Z direction. We initialise all of the fusion methods with an initial estimate of $x_{\hat{k(0)}} = (0, 0, 0)$, the prediction error covariance matrix $M_{(k=0)}$, and estimate error covariance matrix as $P_{(k=0)}$ as a $I_{(6x6)}$ identity matrix. We initialise the process noise as $w(k) = 0.01$ and measurement noise $v_{1(k)} = 0.01$, and measurement noise $v_{2(k)} = 0.08$. The camera detetction's are converted from pixel coordinates to world frame, similarly the LiDAR detection's are converted from the LiDAR frame to the world frame. In addition to the internal sensor noise, these conversions add error to the measurements.

After applying the Group Fusion method from section G, we obtained an RMSE of 2.14m on the fused estimate. Figure 12 and 13 in the appendix show the fused estimate in the X,Y respectively. We plot the ground truth obtained from the simulator and the Group Fusion estimates in Figure 7

We can now take a look at the ICF method discussed in section G. We obtained an RMSE of 2.15m on the ICF method. Similar to the Group Fusion method, we can obtain the estimates in X,Y respectively and plot them. Figure 14 and 15 in the appendix show the fused estimate in the X and Y respectively. We can plot the ICF fused estimate along

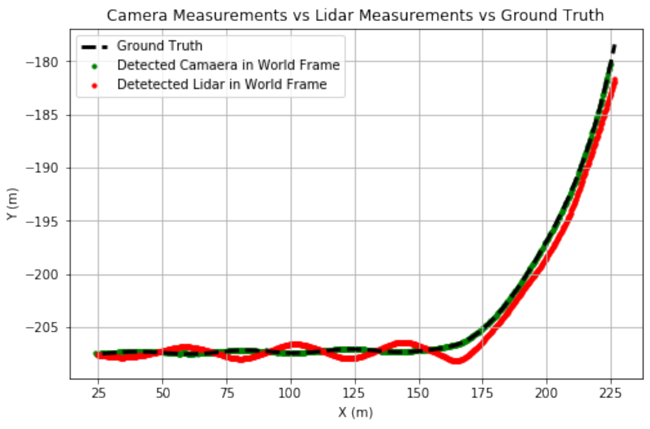


Fig. 6. Sensor Measurements vs Ground Truth (X, Y) in world frame

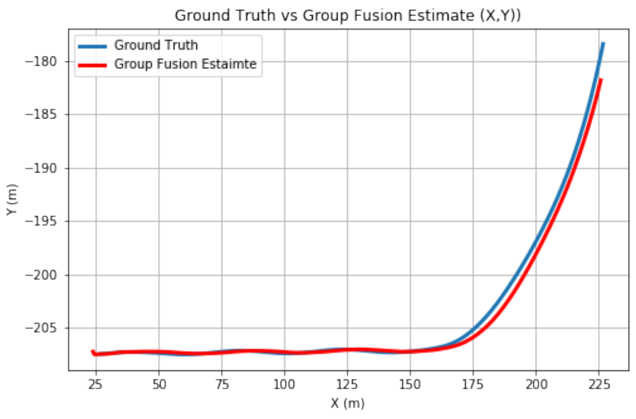


Fig. 7. Group Fusion estimate vs Ground Truth (X, Y)

with the ground truth to visualise the performance as shown in Figure 8

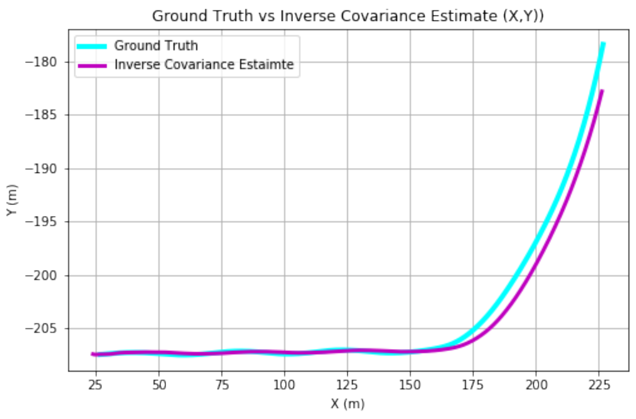


Fig. 8. Inverse Covariance Fusion estimate vs Ground Truth (X, Y)

We have two regression based fusion methods. The Linear Regression fusion has a RMSE of 0.3m and the Artificial Neural Network Based Regression yields a RMSE of 1.76m. Figure 16 and 17 in the appendix show the estimates vs ground truth for the linear regression fusion and ANN Based fusion in X and Y respectively. Figure 9 shows the plot for the ground truth, estimates from the Linear Regression Fusion and ANN Based Regression Fusion in X and Y

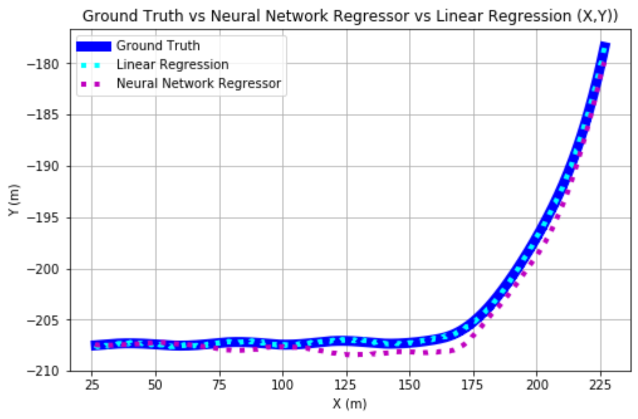


Fig. 9. Inverse Covariance Fusion estimate vs Ground Truth (X, Y)

We can test the machine learning and ANN based regressors for metrics such as Accuracy and Mean Squared Error, but we for the scope of this study, we only focus on the RMSE for the different fusion algorithms. The linear regression based fusion performs better at vehicle pose estimation when compared to the ANN based fusion. ANN's work better at complex problems with higher orders of input features and require a much larger dataset to train, so we would need a lot more measurements. Another caveat of using regression based fusion techniques is that they are trained on a specific sensor suite and might not perform well with other similar sets of sensor fittings. Both Group and ICF Fusion methods show promising results as seen in this section. We can additionally tune the measurement noise covariance matrix so that it relies on sensors with less internal noise. For example, we modeled our measurement noise $v_1(k)$ and $v_2(k)$ such that the Kalman Filter gives the camera measurements more weights than the LiDAR measurements when computing the estimates. This is due to the fact that our LiDAR measurements as seen in Figure 6 are noisier compared to the camera measurements.

IV. CONCLUSIONS

The study presented shows the development of vehicle pose estimation based on inputs from the camera and LiDAR sensor. Three different fusion techniques were modeled and evaluated using the CARLA simulator which were Group , Inverse Covariance, and Regression Based Fusion. Regression based fusion had the lowest RMSE followed by the group fusion. Even though regression based fusion techniques performed better, they need to be tested on new data with new sensor measurements.

Future work should include improved measurement techniques within the simulator, process and measurement noise variation analysis, larger training data-set which includes more measurements for regression based fusion techniques, and applying these fusion methods to real world data.

References

[1] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An Open Urban Driving Simulator,” in Proceedings of the 1st Annual Conference on Robot Learning, 13–15 Nov 2017, vol. 78, pp. 1–16.

[2] “Create a hello world project in ROS – automatic Addison.” https://automaticaddison.com/create-a-hello-world-project-in-ros/ (accessed Dec. 05, 2022).

[3] Leggedrobotics, “Leggedrobotics/darknet_ros: Yolo Ros: Real-time object detection for Ros,” GitHub. [Online]. Available: https://github.com/leggedrobotics/darknet_ros. [Accessed: 06-Dec-2022].

[4] “Coursera,” Coursera. https://www.coursera.org/learn/visual-perception-self-driving-cars/home/week/4 (accessed Dec. 05, 2022).

[5] “Coordinate systems in vehicle dynamics blockset - MATLAB Simulink.” https://www.mathworks.com/help/vdynblks/ug/coordinate-systems-in-vehicle-dynamics-blockset
.html (accessed Dec. 05, 2022).

[6] Seldon, “Machine learning regression explained,” Seldon, Oct. 29, 2021. https://www.seldon.io/machine-learning-regression-explained (accessed Dec. 05, 2022).

[7] Wikipedia contributors, “Linear regression,” Wikipedia, The Free Encyclopedia, Nov. 27, 2022. https://en.wikipedia.org/w/index.php?title=Linear_regressionoldid=112 4084548.

[8] “sklearn.linear_model.LinearRegression — scikit-learn 0.22 documentation,” Scikit-learn.org,2019. https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.L inearRegression.html

[9] R. Srivignesh, “A walk-through of regression analysis using Artificial Neural Networks in Tensorflow,” Analytics Vidhya, Aug. 16, 2021. https://www.analyticsvidhya.com/blog/2021/08/a-walk-through-of-regression-analysis-using-artificial-neural-networks-in-tensorflow/ (accessed Dec. 05, 2022).

[10] “Module: tf.keras — TensorFlow Core v2.4.1,” TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras

V. APPENDIX

Equations

Step 1)

$$M(k) = A_{k-1}P(k-1)A_{k-1}^T + G_{k-1}WG_{k-1}^T \tag{23}$$

where M is the prediction error covariance matrix.

Step 2)

$$\bar{x}(k|k-1) = A_{k-1}\hat{x}(k-1|k-1) + B_{k-1}u(k-1) \tag{24}$$

Step 3)

$$K(k) = M(k)H_k^T(H_kM(k)H_k^T + V)^{-1} \tag{25}$$

where K is the Kalman Gain

Step 4)

$$P(k) = (I - K(k)H_k)M(k) \tag{26}$$

where P is the estimate error covariance matrix

Step 5)

$$\hat{x}(k|k) = \bar{x}(k|k-1) + K(k)[z(k) - H_k\bar{x}(k|k-1)] \tag{27}$$

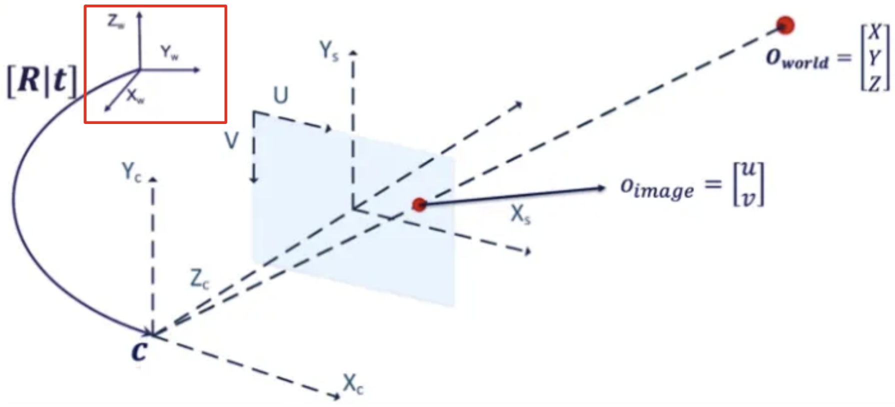
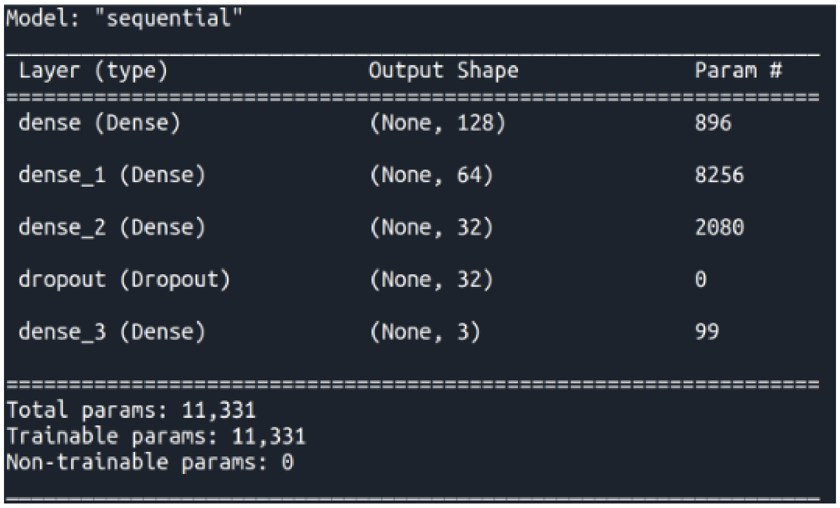


Fig. 10. Relationship between a 3D point in the world frame, the relative pixel coordinates and vice-versa. Image modified from [4]



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               896

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_3 (Dense)             (None, 3)                 99

=================================================================
Total params: 11,331
Trainable params: 11,331
Non-trainable params: 0
_________________________________________________________________
```

Fig. 11. Sequential Artificial Neural Network model summary

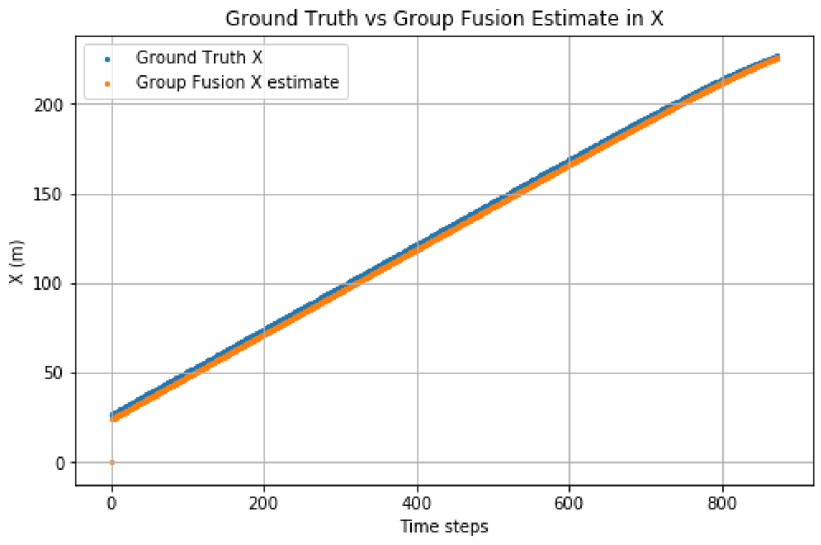


Fig. 12. Fused estimate vs Ground Truth for Group Fusion in X

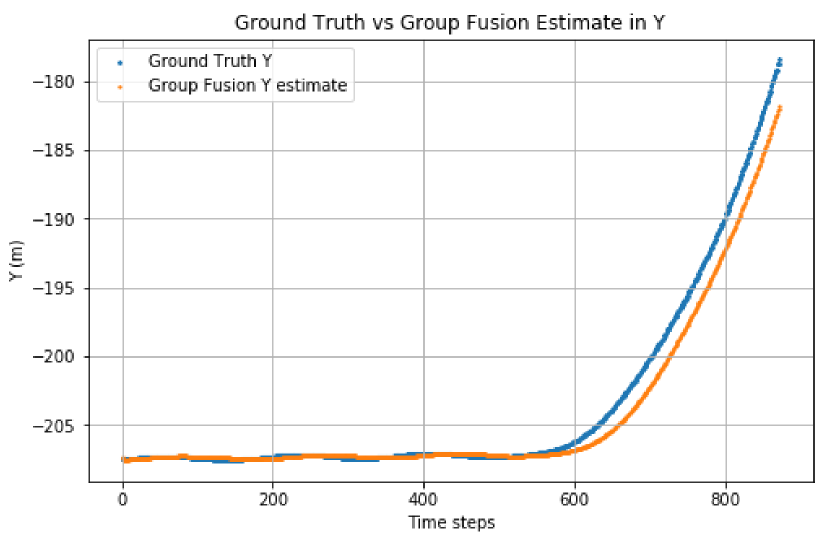


Fig. 13. Fused estimate vs Ground Truth for Group Fusion in Y

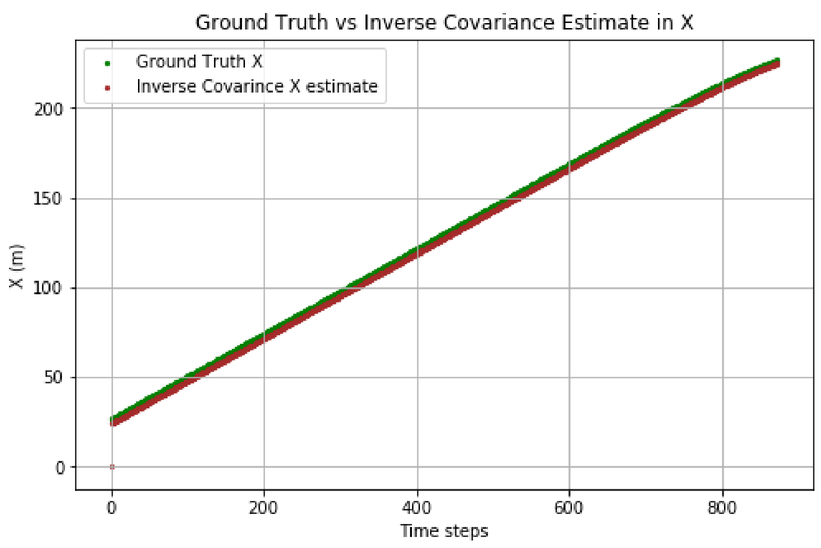


Fig. 14. Fused estimate vs Ground Truth for ICF in X

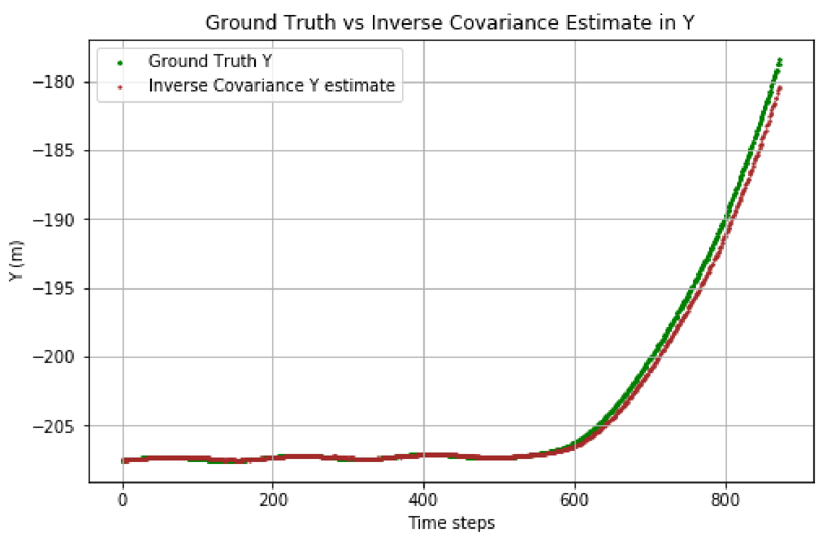


Fig. 15. Fused estimate vs Ground Truth for ICF in Y

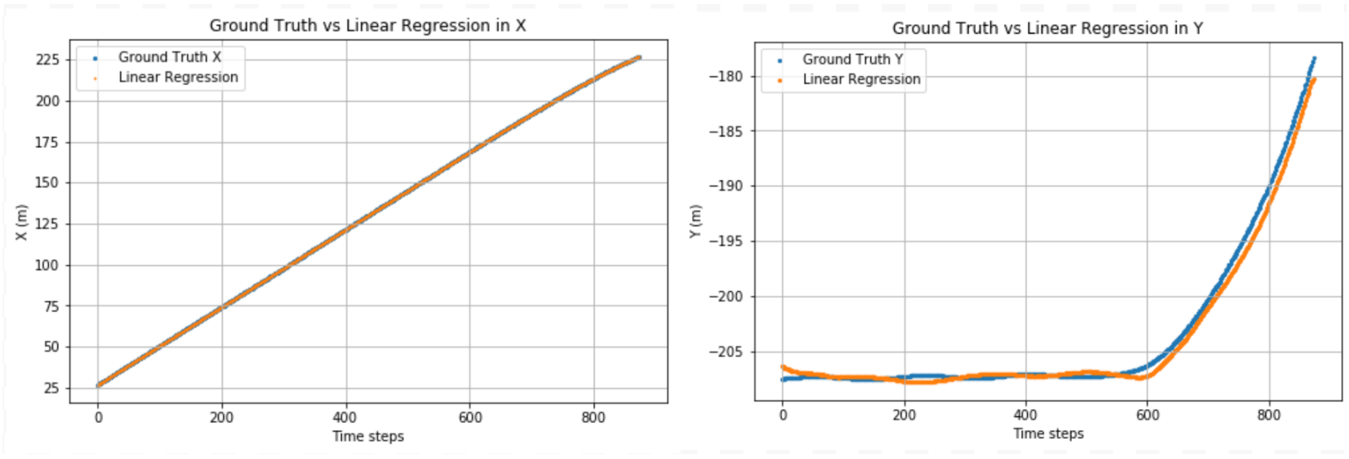


Fig. 16. Linear Regression Fusion vs Ground Truth in X and Y respectively

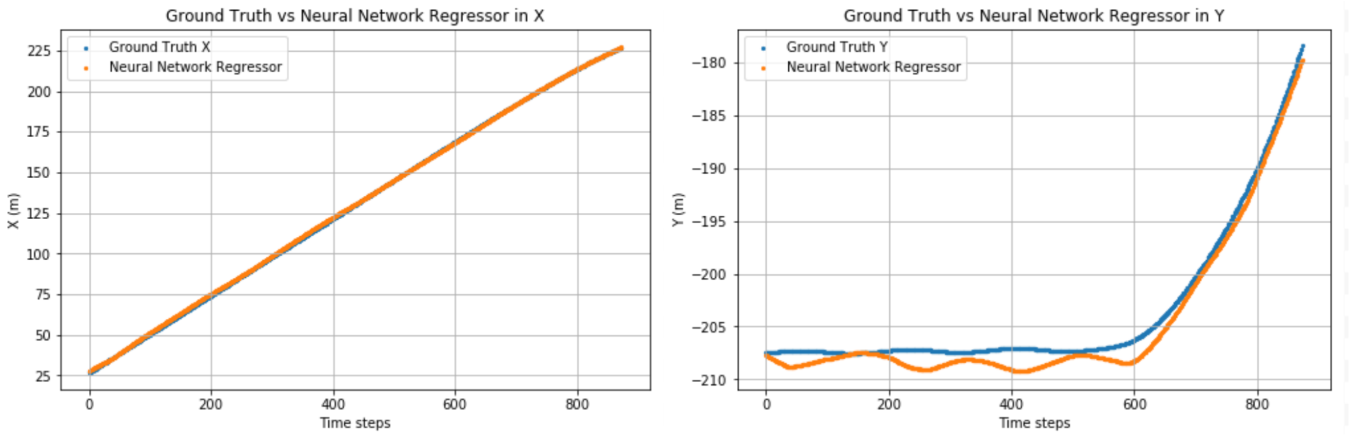


Fig. 17. ANN Based Regression Fusion vs Ground Truth in X and Y respectively