International Journal of Sciences: Basic and Applied Research (IJSBAR)

ISSN 2307-4531
(Print & Online)

http://gssrr.org/index.php?journal=JournalOfBasicAndApplied

Implementation of Vision and Lidar Sensor Fusion Using Kalman Filter Algorithm

Reshma Kunjumon[a*], Sangeetha Gopan G. S.[b]

[a]*College of Engineering Trivandrum, Sreekaryam PO, Trivandrum, 695 016, India*

[b]*Assistant Professor, College of Engineering Trivandrum, Sreekaryam PO, Trivandrum, 695 016, India*

[a]*Email: reshma20409@gmail.com*

[b]*Email: sangeethagopangs@gmail.com*

Abstract

Self-driving car is the next milestone of the automation industry. To achieve the level of autonomy expected in a self-driving car, the vehicle needs to be mounted with an assortment of sensors that can help the vehicle to perceive its three dimensional environment better which leads to better decision-making and control of the vehicle. Each sensor possesses different strengths and weaknesses; they can complement each other better when combined. This is done by a technique called sensor fusion wherein data from various sensors are put together in order to enhance the meaning and accuracy of the overall information. In real time implementations, uncertainty in factors that affect the vehicle's motion can lead to overshoot in parameters. In order to avoid that, an estimation filter is used to predict and update the fused values. This project focuses on sensor fusion of Lidar and Vision sensor (camera) followed by estimation using Kalman filter using values available from an online data set. It can be seen how the use of an estimation filter can significantly improve the accuracy in tracking the path of an obstacle.

Keywords: Vision Sensor; Lidar; Sensor Fusion; Kalman Filter.

---

* Corresponding author.

1. Introduction

Autonomous vehicles have become an important field of interest due to their capability of autonomous navigation. The autonomous (or ego) vehicles can operate and react to the environment without the help of driver as they can make decisions based on the information perceived through the sensors. Sensors are the vital components of the self-driving vehicles and the robust sensing and fusion of information passed down from these sensors and their interpretation are key for the control of vehicle. The Intelligent Transport Systems that make use of various technological aspects has largely benefited from the recent rapid technological advancements. These systems are mostly based on sensor technologies like RADAR, Lidar or Camera. Certain systems are designed such that they alert the driver through audio, visual or haptical signal about the danger. Also, some systems are designed such that they can take over the control of vehicle during emergencies, which is intended to avoid accidents that result from mistakes and carelessness in complex traffic [1]. Furthermore, these sort of systems enable a better driving environment. Though the driving experience and vehicle safety measure has increased drastically over the years, the ever-changing traffic culture has sparked an extreme interest in the research and development of systems with high level of autonomy. Rather than just providing driving assistance, an autonomous car can drive itself to reach the desired location. Such high level of autonomy requires the vehicle to perceive the environment as done by the humans, i.e. the vehicle will have to sense the environment and identify distance to the obstacle, location, movement of pedestrians etc. Also, it will have to carry out a decision making process like humans. This results in need for new array of sensing devices, information processing algorithms, computing units and an entirely new Electrical and Electronic (E/E) architecture. The main feature that sets the autonomous vehicle different from the conventional vehicle is the devices that are used for perceiving and understanding the surrounding environment of the vehicle. Such systems consist of devices like RADAR, Lidar, GPS, Ultra Sonic, Camera, Inertial Measurement Unit (IMU) etc. These devices help in integrating information processing algorithms like Signal Processing, Machine Learning, Encryption/Decryption and decision making. The combination of all devices and associated algorithms helps in determining the autonomy level [2] of the autonomous vehicles. The sensors play a vital role in autonomous driving as they help in better perception of environment around the vehicle. The selection of sensors is essential for the design of an efficient autonomous driving system. This research is focusing on the use of LiDAR and Vision sensors. The use of these sensors will reduce the complexity that is involved in computation. The higher number of sensors will require a performance intensive processor. Another problem with using more than two sensors is writing the library function to read the data from the sensors. It is possible to write a library for a LiDAR or the Vision sensor, but if the system is having a greater number of sensors, writing libraries could take a large amount of time. This is the reason for choosing two sensors.

2. Sensor fusion

Each of these sensors has its own advantages and disadvantages. Among all perception functions, visual perception is one of the most important components. When we consider a camera, it has better performance when the object detection is considered. The advantage of a camera is that it can provide specific information related to the target. It interprets visual data and performs critical tasks such as vehicle and pedestrian detection. In an autonomous driving system, cameras are essential because they mimic human eyes and most traffic rules

are designed by assuming the ability of visual perception. For example, many traffic signs share similar physical shapes and they are differentiated by their colored patterns that can only be captured by a visual perception system. The better resolutions of camera at both lateral and elevation are also an added advantage. However, the camera's functioning is largely affected by the weather. The night condition or a bad weather can affect it performance. This is totally against the idea of a safety sensor. In such cases, a RADAR or Lidar can be implemented. A comparison between RADAR and Lidar shows that a Lidar based sensor can provide good resolution about the position [5]. The short wavelength of Lidar lets us detect small objects and a LIDAR can build an exact 3D monochromatic image of an object. The Lidar also has its working principle that facilitates the calculation of distance and velocity to an object. The camera can measure the distance only when it is used in the stereo vision configuration. There is a lot of limitation in the amount of data about a physical quantity, procured by one sensor [3]. This is because one sensor has many of the following disadvantages:

- Field of View (FOV) coverage is limited.
- Temporal coverage is limited because rate of data acquisition by sensor is limited.
- The malfunction of one sensor affects the reliability of the entire system.
- The precision of the entire system is determined by the precision of the sensing element.
- When some features are missing (e.g. occluded objects), measured data becomes uncertain.

Thus, a probabilistic fusion of radar and video data improves the estimation accuracy by compensating the weaknesses of one sensor with the strengths of another. Combining information from multi-sensor system introduces new challenges [4]. One of the important challenges is a spatio-temporal task: the spatial part is the alignment of frame sensors while the second is handling the update rates of sensors. In the alignment process, a relation is found between the different coordinates in each sensor frame to ensure the transformation from one frame into another. Another major challenge is the timing of operation. This applies to both homogeneous and heterogeneous sensors. The operation frequencies of the sensors are different. For above mentioned reasons, sensor fusion algorithms have to tackle temporal, noisy input and output a probabilistically sound estimate of kinematic state.

3. System design

The proposed system consists of the following features:

- Targets Data Acquisition: Taking input from the sensors i.e, camera and Lidar in the respective formats and storing them.
- Camera Image Processing: Detecting targets in image captured by camera, classification and boundary of targets and the centroids are obtained.
- Projection: The cloud of points obtained from Lidar is projected from the 3 dimensional spaces onto the 2 dimensional images procured by camera.
- Lidar Frame Processing: Segmentation of the Lidar point cloud and extraction of centroid from segmented cloud.
- Data Association: This step is being done to ensure that the same obstacle is being detected by the 2

sensors.

- Bayes Fusion: To create a less noisy resultant measurement.
- Estimation Filter: Obstacle Path Estimation using Kalman Filter.

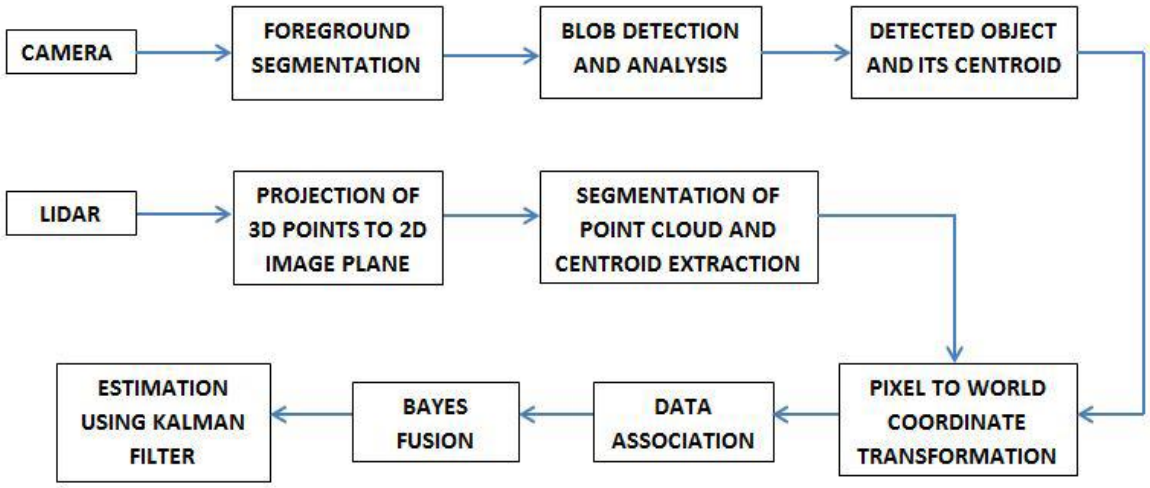


Figure 1: Block diagram of the proposed system

4. Data acquisition and pre-processing

4.1. KITTI vision benchmark suite

In order to prove my hypothesis practically, an experimental set up is needed with practical real-time measurement of the parameter values. But such a set up is outside the scope of this project tenure due to pandemic restrictions. Hence, I had to rely on an online data set called KITTI VISION BENCHMARK SUITE. It is a joint venture of Karlsruhe Institute of Technology, Germany and Toyota Technological Institute, Chicago for Sensor Fusion and Robotic Analysis purposes. It has been made available for academic purposes. For the KITTI Database, a standard station wagon like Volkswagon Passat was fitted with 4 cameras: 2 high resolution color cameras and 2 gray scale video cameras, an HDL-64E Velodyne Laser Scanner and a GPS localization system along with an Inertial Measurement Unit. The location chosen was a highway in Karlsruhe that contains upto 15 cars and 30 pedestrians. Both raw images and benchmarked images were made available of which I have chosen the latter subject to evaluation metrics and real world benchmarks. The calibration files have also been made available by the authors. The cameras provide images at the rate of 10 frames per second. There were 360 images provided per camera. Each grayscale frame is an 8 bit PNG image consisting of 1392x512 pixels. In effect, each image was around 5.6 MB size. While each frame of the RGB image is a 16 bit PNG image, amounting to double the size. The HDL-64E Velodyne Lidar is a 360^{o} counterclockwise rotating Lidar at 10Hz. The sensors were time synchronized by the authors. It procures the 3 dimensional coordinates of about 1.3 million points per second and their reflectance values in the single path mode. These points are available in the form of text file and BIN files. The values stored as binary float matrices in BIN files which allows them to have greater accuracy. Hence, they were chosen for my project. Also, due to the storage limitations, the number of points was rectified to about 21947 points per frame.

4.2. Camera image processing

Once the sensor data has been procured, relevant obstacles have to be detected from the data. In case of camera, we know that the sensor data consists of 2 dimensional images. Advanced image processing techniques in object detection can be used to identify and recognize the obstacles. There are many object detection algorithms like HOG detector, R-CNN, SSD, YOLO, etc which can be implemented. In this case, considering hardware limitations, I have opted for a simple object detection based on foreground segmentation and blob analysis technique. This technique is based on Gaussian Mixture Models and Blob Analysis. It consists of four steps:

- Foreground segmentation
- Blob Detection
- Blob Analysis
- Blob Tracking

The vision toolbox of Matlab is made use of for this purpose. Using the command vision.ForegroundDetector(), the image frames are compared to a mixture of built-in gaussian mixture models. The parts of frame that match the model is considered as background and rest of the frame is considered as foreground. Depending on this conclusion, the image is binarized with pixel intensity 0 or black given for background pixels while intensity 255 or white given for foreground pixels. This step is followed by a morphological opening operation to remove the noise and fill the gaps in detected objects. This is done using the imopen() command making use of a structural element (strel()) of size 1 percent of video's width. Following this, the minimum area of the foreground required to be detected as a blob is specified as 8 percent of the video's frame area. Using vision.BlobAnalysis() command, the object is detected and using insertShape() a green bounding box is drawn around the detected obstacle and its centroid is displayed.

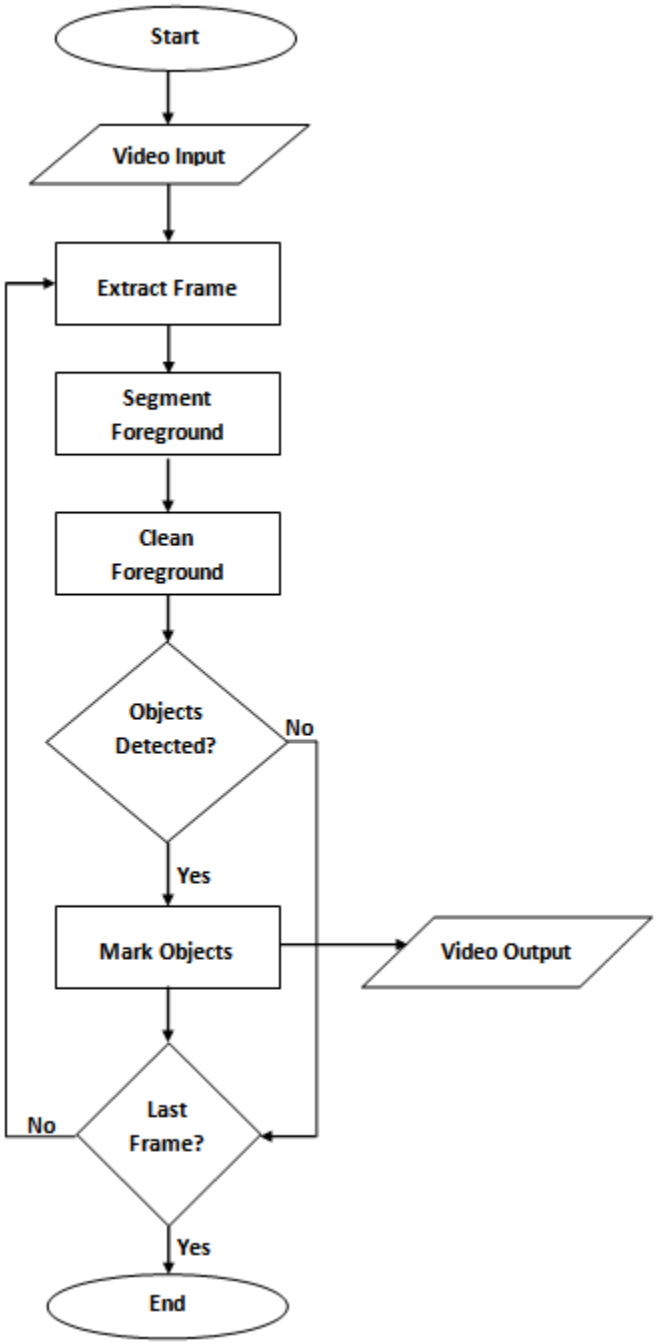


Figure 2: Algorithm for object detection using blob analysis

It is worth noting here that when multiple objects are detected, we have to go for object recognition algorithms and prioritization step, in order to assign priorities to the detected obstacles in further tracking. In order to reduce complications, I have stuck to the main obstacle captured by the sensors, i.e, the vehicle in front of the ego vehicle and skipped the prioritization step. In future scope of the project, these steps can be included to maximize the output.

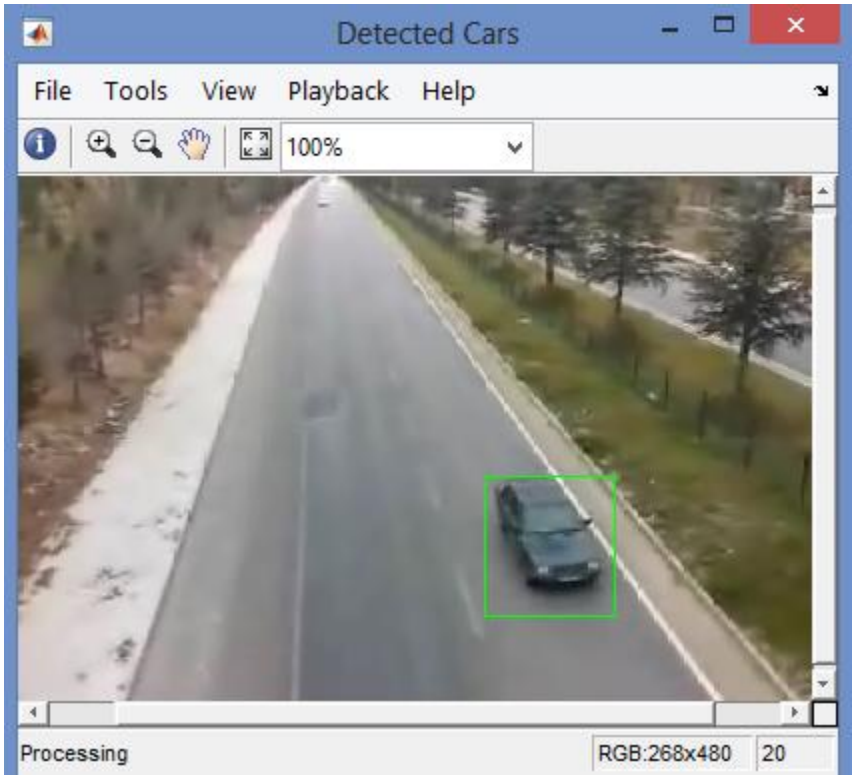


Figure 3: Output of the object detection code

4.3. Multi sensor data fusion network

After the process of data acquisition, before doing fusion, we need to make sure that the same object is being detected by the sensors. For this we need to do the process of data association. This is discussed in the next section. For data association, we need to make sure that the data from different sources belong to the same frame of reference. This process is called sensor alignment. We can extend the proposed approach to n sensors and append the alignment process. These sensors are homogeneous and/or heterogeneous and their task is to measure the position of the detected obstacles. It consists of two main processes:

- Sensor alignment process (off-line). The inputs of this process are sensor data while the outputs are the calibration parameters (rotation matrix and translation vector). This process is an extrinsic calibration between different sensors (source and targets). It allows estimating the relative position of point p in a common frame.
- Object detection process (on-line). In this step, there are n processing chains. Each of them provides a list of the detected objects.

With the conjunction of the calibration parameters obtained from sensor alignment process, we are able to fuse the data together to better detect the objects in the surroundings. In this work, focus is made on Lidar and camera sensor. In the following sections, each task will be detailed.

4.4. Sensor alignment : calibration parameters

To efficiently perform sensor fusion, the sensors should be calibrated. The calibration process is an alignment procedure of the given sensor frames. That is to say, to find the relation between the coordinates of sensor frames to ensure the transformation from a frame into another. To carry-out this process, we need two calibration parameters i.e, the rotational matrix and the translation matrix. In this case, these parameters were already provided in the KITTI data set. Else, they have to be experimentally calculated. The data set contains 3 calibration files: camera to camera, Velodyne to camera and IMU to Velodyne. Of these, the third file is of no use to this project as IMU sensor is not included. The other two files contain the various parameters needed to calibrate the camera and project the Lidar points from the 3 dimensional space to the image plane. Of these parameters, the ones that we need in order to calculate the transformation or projection matrix are R_rect and P_rect. Also, each of these parameters has been listed for the four cameras used in the experiment. We have considered only one color camera for this project. Hence, only the first camera's parameters will be used. The second file we need is called Tr_velo_to_cam.txt. It is dependent on the camera's external parameters. It is a 4x4 double matrix. This is used for velodyne to camera transformation. It contains the rotational matrix values in the first 9 cells, translation matrix in the fourth column and 2 small delta values in the fourth row. We define a 4x4 double matrix called R_cam_to_rect. This contains the R_rect values from the first file copied to a 4x4 matrix and the remaining values appropriately filled. This is done to correct the order of matrix for proper multiplication. The screenshot of the required parameter values in Matlab workspace is given in figure 4. The next step is to calculate the Projection Matrix. The variable name given is P_velo_to_img. It is a 3x4 matrix obtained by multiplying the cameras internal parameters (P_rect), rectified rotational matrix (R_cam_to_rect) and external parameters (Tr_velo_to_cam). The point cloud in LiDAR plane can be converted to the 3D camera plane by using this equation.

$$P\_velo\_to\_img = P\_rect\{cam + 1\} * R\_cam\_to\_rect * Tr\_velo\_to\_cam \quad (1)$$

Figure 5 illustrates the values of the projection matrix as calculated using the above equation. Once the projection matrix is calculated, we can easily convert the point cloud from Lidar plane to camera plane by multiplying the points with the projection matrix. As mentioned already, the Lidar data provided by the KITTI data set contains the x, y and z coordinates of the points along with a fourth column containing their reflectance values. The number of points has been rectified by KITTI authors from 1.3 million to a useful set of 21947 points. These are stored per frame in BIN files. The BIN files are read using a file descriptor into a 21947x4 double matrix called velo. In order to further reduce the computation time, we have scaled the original set by choosing a set of relevant 6535 points. Also, reflectance values have been neglected because it is outside the scope of this project. The projection equation is as follows. Here, P_t contains the transformed points to the camera plane. It is a 6535x3 matrix.

$$P\_t = P\_velo\_to\_img * velo \quad (2)$$

VARIABLE SELECTION

calib | calib.R_rect | calib.R_rect{1, 1}

calib.R_rect{1, 1}

	1	2	3	4
1	0.9999	0.0098	-0.0074	
2	-0.0099	0.9999	-0.0043	
3	0.0074	0.0044	1.0000	
4				

VARIABLE SELECTION

calib | calib.P_rect | calib.P_rect{1, 1}

calib.P_rect{1, 1}

	1	2	3	4
1	721.5377	0	609.5593	0
2	0	721.5377	172.8540	0
3	0	0	1	0
4				

VARIABLE SELECTION

Tr_velo_to_cam

4x4 double

	1	2	3	4
1	0.0075	-1.0000	-6.1660e-04	-0.0041
2	0.0148	7.2807e-04	-0.9999	-0.0763
3	0.9999	0.0075	0.0148	-0.2718
4	0	0	0	1
5				

Figure 4: Calibration parameters

VARIABLE SELECTION

P_velo_to_img

3x4 double

	1	2	3	4
1	609.6954	-721.4216	-1.2513	-123.0418
2	180.3842	7.6448	-719.6515	-101.0167
3	0.9999	1.2437e-04	0.0105	-0.2694
4				

Figure 5: Projection matrix

Now the transformed points are in the camera plane but they are still in 3 dimension. They need to be converted to 2 dimensional points to project them to the image plane. Dividing the x and y coordinates with the z coordinates will project the 3D point to a 2D plane. This is contained in the P_p matrix which is of size 6535x2 and contains only x and y coordinates. This is plotted on the image plane as shown in Figure 6.

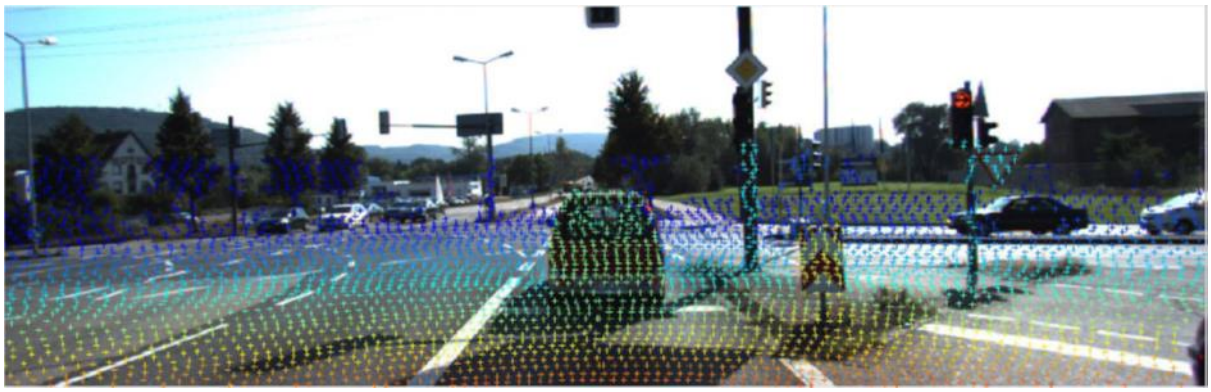


Figure 6: Lidar points projected onto camera plane

4.5. Lidar frame processing

Next step is to segregate the cloud of points such that points belonging to relevant obstacles can be isolated and segmented. This process is called segmentation. To extract the projected points of the source sensor on the calibration board, the automatic extraction approach by differentiation of the measurements and background data in static environments is often used. However, since this work is not based on experimental set up, this approach cannot be used. Hence, this portion is out of scope this project. Rather, what has been done is the manual selection of the centroids from the given frames. The coordinates of the centroid of the obstacle obtained from image frames and Lidar points were collected to an Excel sheet which has been displayed in Figure 7.

Clipboard Font

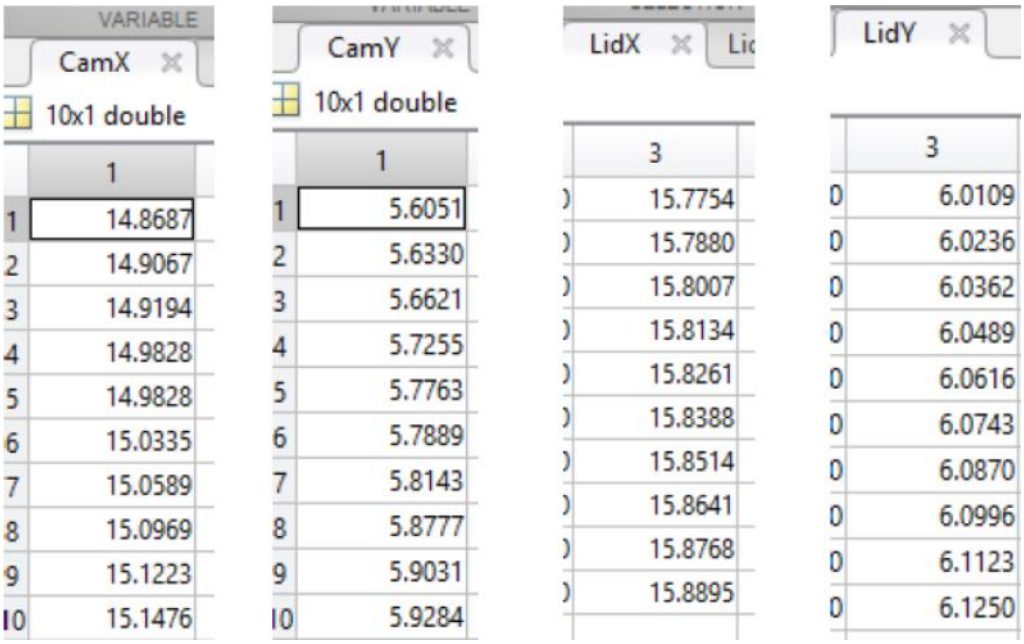
J11 f_x

	A	B	C	D
1	CamX	CamY	LidX	LidY
2	586.25	221	622	237
3	587.75	222.1	622.5	237.5
4	588.25	223.25	623	238
5	590.75	225.75	623.5	238.5
6	590.75	227.75	624	239
7	592.75	228.25	624.5	239.5
8	593.75	229.25	625	240
9	595.25	231.75	625.5	240.5
10	596.25	232.75	626	241
11	597.25	233.75	626.5	241.5

Figure 7: Coordinates of obstacle's centroid

4.6. Pixel to world coordinate transformation

The coordinates of the centroid are available in pixel values. It needs to be converted in terms of real world distances because the points of Lidar correspond to real world distances. For this, we need the actual measurement of road width, size of the obstacle, etc to find the scale between pixel value and real world distance. Since that information was not available in the online data set, an approximated scaled down transformation was done considering the real world length of the image frame and the obstacle size. This data is recorded and shown in Figure 8.



VARIABLE — CamX — 10x1 double

	1
1	14.8687
2	14.9067
3	14.9194
4	14.9828
5	14.9828
6	15.0335
7	15.0589
8	15.0969
9	15.1223
10	15.1476

CamY — 10x1 double

	1
1	5.6051
2	5.6330
3	5.6621
4	5.7255
5	5.7763
6	5.7889
7	5.8143
8	5.8777
9	5.9031
10	5.9284

LidX — Lic

	3
)	15.7754
)	15.7880
)	15.8007
)	15.8134
)	15.8261
)	15.8388
)	15.8514
)	15.8641
)	15.8768
)	15.8895

LidY

	3
0	6.0109
0	6.0236
0	6.0362
0	6.0489
0	6.0616
0	6.0743
0	6.0870
0	6.0996
0	6.1123
0	6.1250

Figure 8: Centroid coordinates in terms of physical distance

5. Data association

Now that we have the coordinates of the centroid of the obstacle obtained from different frames of the camera and Lidar in terms of physical distances, we need to make sure that the coordinates procured by camera and Lidar belong to the same obstacle. If not, the next step which is Fusion will yield highly erratic results. In order to do this step, we need to calculate the distance between the centroid coordinates obtained by the 2 sensors. If this calculated distance lies below a certain preset threshold, which is subject to real world sizes of objects, then we can safely say that the coordinates belong to the same object. Another thing to be noted here is that, when we use the original Lidar point cluster, the centroid of the cluster will not be in the centre like the one received from the image because in Lidar, the distribution of point cloud is not homogeneous or uniform. So to calculate the distance between such a centroid of the cluster, we use Mahalanobis Distance instead of the regular Euclidean Distance. The simplified version of this is given below. Here the denominator stands for the variance of the data. If the same object is detected, "Same target located" is printed.

$$d_k(a,b) = \frac{(x_a - x_b)^2}{\sigma_{x_a}^2 + \sigma_{x_b}^2} + \frac{(y_a - y_b)^2}{\sigma_{y_a}^2 + \sigma_{y_b}^2} \quad (3)$$

6. Bayes fusion

Once it is confirmed that the centroids belong to the same object, it is now time to fuse these values to get a less noisy resultant. The sensors are employed to detect the positions of obstacles. Position uncertainty is represented using the standard variation. Therefore, if X is the true position of the detected object, by using the Bayesian fusion, the probability of fused position P_F $[\ x_F\ \ y_F\ ]^T$ by the two sensors is given as:

$$P_{prob}\left(\frac{P}{X}\right) = \frac{e^{\frac{-(P-X)^T R^{-1}(P-X)}{2}}}{2\,\pi\, R^{0.5}} \quad (4)$$

where P is the fused position and R is the covariance matrix are given as:

$$P = \frac{\frac{P_1}{R_1}+\frac{P_2}{R_2}}{\frac{1}{R_1}+\frac{1}{R_2}} \quad (5)$$

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} \quad (6)$$

where P_1 and R_1 are respectively the position and covariance matrix of sensor 1 and P_2 and R_2 are that of sensor 2. So, the outcome is a combination of the two measurements weighted by their noise co-variance matrices. The fused results using Bayesian approach follow the measurements provided by the sensor which has the smallest covariance matrix and gives more trust to it. So finally, the value of fused measurement vector is:

$$\hat{y} = \{\sum_{k=1}^{m} R_k^{-1}\}^{-1} \sum_{k=1}^{m}\{R_k^{-1}\, y_k\} \quad (7)$$

In case of 2 sensors, the equation further simplifies to:

$$\hat{y} = \{\, R_1^{-1} + R_2^{-1}\,\}^{-1}\{\, R_1^{-1} y_1 + R_2^{-1} y_2\} \quad (8)$$

R is the covariance matrix. It is an n x n positive definite symmetric matrix.

$$R_{cam} = \begin{bmatrix} \sigma_{cam_x}^2 & 0 \\ 0 & \sigma_{cam_y}^2 \end{bmatrix}$$

$$R_{lid} = \begin{bmatrix} \sigma_{lid_x}^2 & 0 \\ 0 & \sigma_{lid_y}^2 \end{bmatrix}$$

Sensor fusion is a technique that has to be implemented in real time on a moving car and has to take in continuously changing values. In such situations, 2 types of errors are prominent. One is called measurement noise. It is the error inherent in any sensing system. The more important error is the plant or driver noise. When a sensor is calibrated, it is done using the simplest model: a target moving in a straight line at constant speed. But when implemented practically, the sensor is mounted on a moving car. Also, the targets need not be moving in a straight line or at constant speed. So their paths will be an accelerated curvy flight which causes errors with

the Bayesian filter because it doesn't have any mechanism to update the values according to the error value. At each time step, the sensor values are influenced by a number of factors like acceleration, gravity, angular velocity, air friction, stream line effects, etc. Hence the error keeps accumulating. Another technique called the Kalman Filter technique is efficient in handling continuously changing real time data. It has 2 cycles: a prediction and an updation cycle. This allows it to update the next values based on the error of the previous stage. Thus it minimizes error with each iteration of the algorithm. Also, it can handle both position and velocity (state) of the target simultaneously. It also doesn't need to remember previous data which significantly saves time and cost.

7. Kalman filter

The Kalman Filter is the most commonly used algorithm in the tracking and prediction tasks. Its use in the analysis of visual motion can be seen quite frequently. The purpose of filtering is to extract the required information from a signal, ignoring everything else. Kalman filter has found great importance as the filter is defined in terms of state space methods which help to simplify the implementation of filter in the discrete domain. Since this project is based on simulation results and not experimental set up, the initial values of a number of parameters that have to be practically derived are deficient. Hence, we had to stick to the most basic model i.e, the constant velocity model designed based on kinematics equations. The ego vehicle is assumed to be moving at a constant velocity of 0.37 m/s. The Kalman filter is a closed form solution to the Bayesian Filter. The filter assumes linear Gaussian motion and measurement models and the prior distribution

$X_0 \sim N(x_0, P_0)$

Where x_0 and P_0 are the initial state and covariance matrices.

$$x_0 = \begin{bmatrix} Initial\_x\_position \\ Initial\_y\_position \end{bmatrix};$$

$$P_0 = \begin{bmatrix} Error\_in\_predicted\_position \\ Error\_in\_predicted\_velocity \end{bmatrix} = \begin{bmatrix} 0.001 \\ 0.0001 \end{bmatrix}$$

With the previous assumptions we get the following

$$x_{kp} = AX_{(k-1)} + w_k \quad (9)$$

Where x_{kp} is the predicted state matrix. A is the state transition matrix of the motion model. It is a square matrix $\begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$. Δt is the time stamp gap. It is taken equal to 0.1. w_k is the error in prediction. We neglect it here.

$$P_{kp} = AP_{(k-1)}A^T + Q_k \quad (10)$$

Where P_{kp} is the Process Co-variance Matrix. Q_k is the error in estimation. It is also called Driver Noise Matrix. These 2 steps are called the prediction step. Next is the update step. It is here we calculate the Kalman Gain

matrix.

$$K = \frac{P_{kp}H^T}{HP_{kp}H^T + R} \quad (11)$$

Here, H is an identity matrix used to correct order. It transforms the given state vector into measurement. R is the random measurement noise. It is also called observed error. Its value is taken as 0.03*H. Using the Kalman Gain Matrix, we can estimate the next position or velocity to a high degree of accuracy.

$$X_k = X_{kp} + K\,(Y - HX_{kp}) \quad (12)$$

X_k is the estimated state vector i.e, position and velocity. It is based on the predicted value as well as the actual measurement, Y. As we can see, the Kalman Gain Matrix acts like a weightage. Depending on whichever has the least co-variance in the previous step, the Kalman Gain Matrix will be adjusted to support that particular variable in the next round which sort of acts like a self check.

$$Y_k = HY_k + Z_m \quad (13)$$

Y_k is the observed value or the input measurement, the data that comes from the sensors. In our case, this will be the coordinates obtained after Bayes fusion. Z_m is the random measurement noise, taken to be zero.

$$P_k = (I - KH)P_{(k-1)} \quad (14)$$

One more value has to be updated i.e, P_k is the updated process co-variance matrix. I is the identity matrix used for order correction. In the next iteration, the estimated state vector, X_k and updated process co-variance matrix, P_k becomes the $X_{(k-1)}$ and $P_{(k-1)}$.

8. Observation and results

After implementing the Kalman Filter on the Bayes fused results, we plotted a graph of the obstacle path on the Cartesian plane. 2 graphs were plotted. The orange line is the path plotted using Bayes fused values from the sensor input while the blue line is plotted using values estimated by the Kalman Filter. We know that in order to avoid complications, we had gone for a constant velocity model, which means that the direction of the ego vehicle or the obstacle will not have change over the entire duration of the journey. As we can clearly see from Figure 9, the blue line resembles the constant velocity model more than the orange line. Hence we can conclude that an estimation filter like Kalman Filter can predict the path of the tracked objects more accurately and reduce the noise in measurement of the sensors rather than simply employing a fusion algorithm on the sensor data. , It can also be observed that Kalman filter generates a fused value that is closer to the original value. This in turn proves that we are tracking the object as desired. In order to get these results, only data from online sources have been made use of. Experimental setup and real world data collection was hampered due to pandemic restrictions. Parameters involved in using Kalman Filter require accurate data from experimental set up, many of which were absent from the online database. Approximate values have been substituted wherever possible. But

such approximations would affect the performance of the Kalman Filter and accuracy of the final result. In a highway simulation setting involving a physical system and more parameters, the Kalman filter can generate the sensor fusion values much faster and accurately that can help the ego vehicle to make proper decisions.

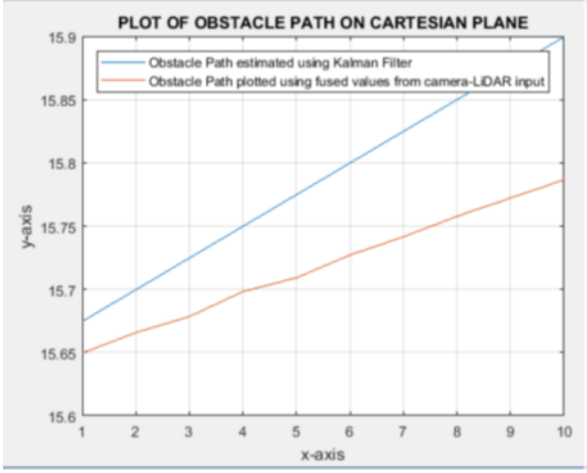


Figure 9: Plot of obstacle path on Cartesian plane

9. Conclusion

The main purpose of this project was to achieve the fusion of data obtained from the Lidar and Camera sensors attached to the ego vehicle. The system was modeled successfully based on the state space model and an algorithm was implemented to fuse the data. The fusion algorithm devised helped to achieve the prediction and Kalman gain calculation that are important to the operation. It helped to perform specific task like data association which in turn resulted in coordinate transformation, data fusion etc. It can be observed that even though we have considered a constant velocity model, the final result is very similar to the original path. When other parameters are brought in and more factors are modeled, the accuracy of the system will improve and so will the complexity. For now, under the scope of this project, the constant velocity model serves the purpose of understanding Sensor fusion using Kalman Filter for Lidar and Camera sensor. Thus, the fusion of data from the camera and radar sensor was achieved successfully using the Kalman Filter. This research can be extended to an experimental set up using real world values for the parameters and by considering the effect of many practical factors that affect the system, the accuracy of prediction and updation by Kalman Filter can be further improved. Also, based on the generated result, a decision-making system can also be created so as to control the operation of the ego vehicle in real time.

References

[1]. **Adam Ziebinski, Rafal Cupek , Hueseyin Erdogan , and Sonja Waechter**, “A Survey of ADAS Technologies for the Future Perspective of Sensor Fusion”, N.T. Nguyen et al. (Eds.): ICCCI 2016, Part II, LNAI 9876, pp. 135–146, 2016

[2]. SAE J3016, Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-road Motor Vehicles, 2016.

[3]. **Mokhtar Bouain, Karim M. A. Ali, Denis Berdjag, Nizar Fakhfakh, and Rabie Ben Atitallah**, "An Embedded Multi-Sensor Data Fusion Design for Vehicle Perception Tasks", Journal of Communications Vol. 13, No. 1, Jan. 2018

[4]. **B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi,** "Multisensor data fusion: A review of the state-of-the-art," Information Fusion, vol. 14, no. 1, pp. 28–44, 2013.

[5]. **T. Ogawa and K. Takagi,** "Lane recognition using on-vehicle lidar," in Proc. IEEE Intell. Veh. Symp., 2006, pp. 540–545.

[6]. **J. Becker.** "Fusion of data from the object-detecting sensors of an autonomous vehicle", in Proc. 1999 IEEE Int. Conference on Intelligent Transportation Systems, pages 362-367, Oct. 1999.