Retrofitting an old Car with Modern Perception Hardware and Deep Learning Based Algorithms

Mengshi LI
SID: 013818454
Email: mengshi.li@sjsu.edu

Tomasz Lewicki
SID: 013855803
Email: tomasz.lewicki@sjsu.edu

Shafie Mukhre
SID: 013771043
Email: shafie.mukhre@sjsu.edu

Abstract—**In this project, we present an end-to-end early warning system for an old car. We have setup the hardware platform based on Jetson Nano, consisting of sensors such as RTK GPS, Radar and Stereo Camera. With these sensors we have collected data under the ROS framework. With the collected raw data we have studied several deep learning based perception algorithms with emphasis on 2D object detection with radar fusion and 3D object detection. The system will generate buzzing sound to assist the driver if it detects potential risks, such as unplanned lane change, or a pedestrian in the way.**

I. Introduction

Many accidents involving pedestrians could be prevented by drivers faster reaction or more attentive road observation. Human reaction times at the wheel are often measured in seconds, instead of milliseconds. That is the motivation for this Driving Assistant System. In this project we developed an end to end early warning system by setting up the hardware platform, collecting multiple types of raw sensor data and applying deep learning based algorithms on the data to obtain perception of the driving surroundings and notify the driver in case of potential risks. The overall architecture of the project is demonstrated in Fig 1.

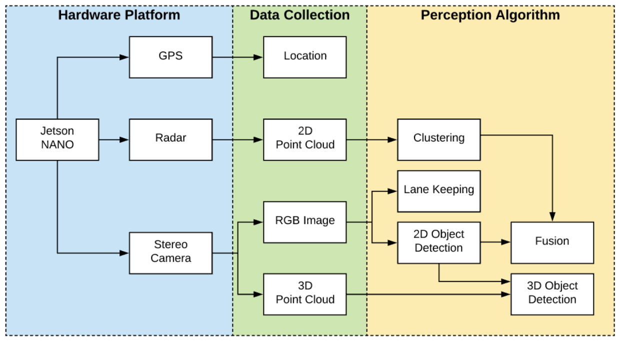


Fig. 1: Project Overview Diagram

The remaining of this report is organized in the following way: in section II we briefed the background on the key components of the whole system; in section III we described the hardware platform setup; in section IV we demonstrated the ROS-based software setup and specifically the 2D object detection and camera-radar fusion pipelines; in section V we studied two kinds of 3D object detection algorithms; in section VI we illustrated the lane keeping algorithm. The whole report is wrapped up in section VII with plan of future work.

II. Background on Key Components of the System

A. Background on 2D Object Detection

We can currently identify two main approaches to object detection problem: region-proposal based models and single-shot detectors. The former approach has a two-stage pipeline, where the first stage is responsible for selecting region proposals, and the second stage performs classification on the selected regions. The second approach handles object detection as a regression problem with a single stage architecture. A recent in-depth review of different detection models, including their trade-offs and performance results has been described in[24]. The topic of detection models has been a field of intensive research over the recent years, bringing significant improvements to object detection on images via CNN-based models. Most importantly, the inference time was greatly reduced from multiple seconds for a single frame in 2013[5], to multiple FPS (Frames Per Seconds) as reported from state of the art models, such as YOLO[16], Faster-RCNN[17] or SSD[9] and their variations. This inference-time performance improvement makes CNN-based object detection feasible for use in real-time applications such as autonomous driving.

In this paper, we choose mask-RCNN[6] as the 2D image detector. Mask-RCNN, aside from the fine-tuned Faster-RCNN-based object detection, provides an additional instance segmentation branch. This additional information could be very useful for autonomous driving scenarios as it could enable pedestrian pose estimation and tracking. We are hoping to leverage it in the future work.

B. Background on Radar Detection

Radar has been long adapted by the automotive industry for basic autonomous driving assistant tasks such as emergency braking and cruise control. However, the recent radar technology advancements and the introduction of 77GHz radars increased the resolution of these sensors to the point in which radar became a candidate for object detection and sensor fusion with camera. In this paper we present a simple radar point cloud processing pipeline, as well as fuse the results with camera, by matching the detections from the 2D image with the point cloud clusters.

We're hoping that the resulting comprehensive fusion information consisting of: 3D position, object label, confidence score and reflectivity to radar, could be groundwork for more comprehensive pedestrian tracking, especially when supplemented with human pose estimation and doppler-based speed measurement.

C. Background on 3D Object Detection

A comprehensive summary regarding 3D object detection has been presented in [4]. As depicted in TABLE III of this article, 3D object detection could be based on RGB image from monocular camera, or on point clouds from Lidar, or a fusion of these two sensors. For the methods working on point clouds, the main challenge is the vast sample space. There are currently three categories of methods to represent the 3D space. The first one is by projection, either spherical projection[19][20], or bird eye view projection[7]. The second method is to voxelize the 3D space and represent each voxel with a single point. The third method is to use the point cloud directly, such as PointNet[13][14]. As expected, the fusion method outperforms the others since it incorporates the information from RGB image with the point cloud. There are three most popular fusion based algorithms, MV3D[21], AVOD[8], and Frustum PointNet[12].

In this project, we have studied the SqueezeSeg algorithm and the Frustum PointNet algorithm. But a booming algorithm[22] has come to world, which, other than providing a bounding box for the object, uses a key point to represent the object and regresses to obtain all the other features such as 2D BBox, 3D BBox and even person pose. This will be our future work.

D. Background on Lane Keeping

Lane control is one of the most important aspects for an autonomous car in order to be able to change lane autonomously. This project focuses on giving the modern perception ability input to the system and convey a warning output to the driver which will alert the driver to be more careful while on the road. This feature is also in the same family with some of the existing Lane Departure Warning Systems such as Lane Keeping Assist (LKA) and Lane Centering Assist (LCA) that are available in some of the recent cars. The early technique for Lane Keeping includes the use of infrared sensors[3] while most of the recent technology utilize a camera[18] and real-time image processing.

III. Hardware Setup

The hardware platform we used is a 1999 Mercedes CLK320. It does not have the sensory or the computing hardware, which are typically components of an autonomous vehicle. We retrofitted the car with the following sensors:

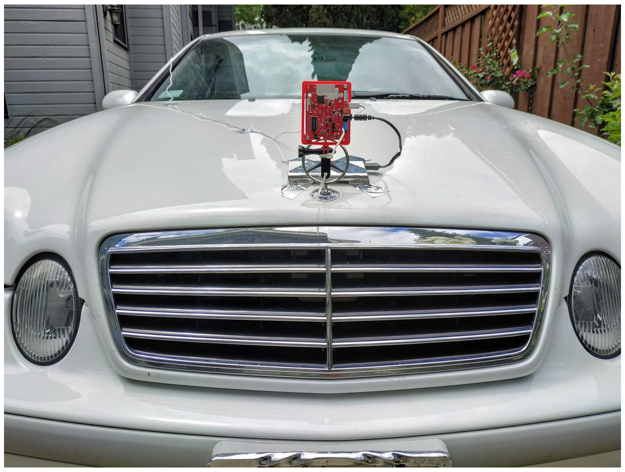


Fig. 2: Radar Mounted on the 1999 Mercedes CLK

1) Texas Instruments 76-81GHz AWR1642BOOST evaluation module
2) 2x Emlid Reach RTK GNSS module + 3DR SiK Telemetry module
3) Stereolabs ZED Camera

The computing platform we used is Nvidia Jetson Nano. The full setup is shown in Figure 3.

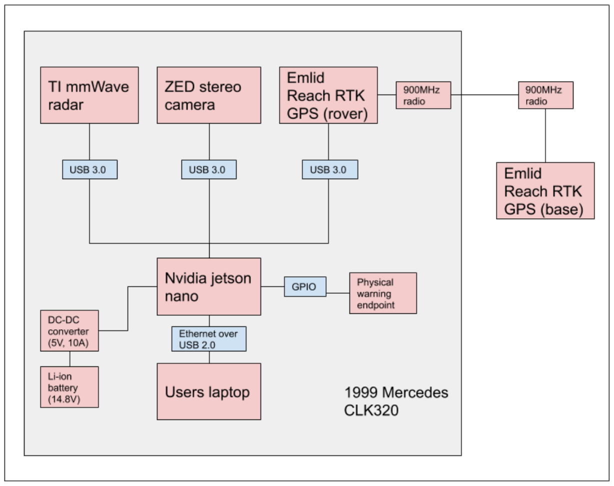


Fig. 3: Hardware Platform Overview

It is important to emphasize that parts of the inference, especially those related to 2D and 3D detection, were carried out on collected data on an external PC. Nevertheless, the system is capable of real-time deployment, given that Nvidia Jetson Nano were replaced with a more capable computing unit.

IV. 2D Detection, Camera-Radar Fusion and Software Setup

A. ROS-based Architecture

Most of the software components described in this paper are implemented as ROS[15] nodes. However, the use of ROS is mainly as a wrapper to provide real-time message exchange between nodes and could be substituted with a different message queue middleware. Even though ROS doesn't officially support versions of python above 2.7, the nodes were purposefully implemented in python 3.6 for better compatibility with other dependencies. The schematic with the overview of the architecture is available in Figure 4.

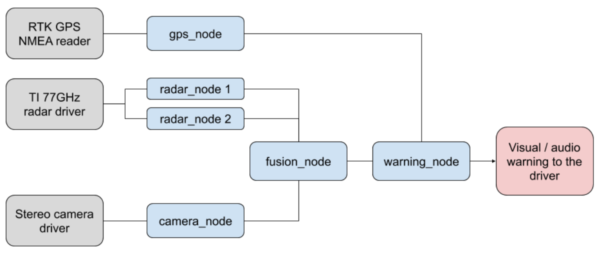


Fig. 4: ROS Architecture Schematic

B. 2D Object Detection Pipeline

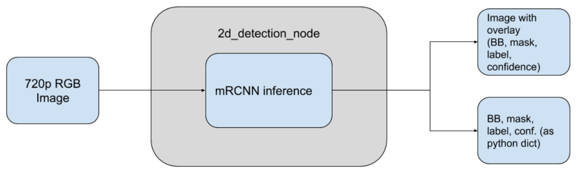

720p RGB Image
2d_detection_node
mRCNN inference
Image with overlay (BB, mask, label, confidence)
BB, mask, label, conf. (as python dict)

Fig. 5: 2D Object Detection Node Schematic

We use a keras implemenation[1] of mask-RCNN[6] model. The node subscribes to the images from the camera, performs inference, and publishes information about each detected object including:

- bounding box
- predicted label
- confidence score for the label
- mask (instance segmentation result)

Figure 6 contains an example of mask-RCNN inference on the collected data.

1) Training: We performed 48-hours of training of the model on COCO dataset using Nvidia P100 GPU on HPC. After hyperparameter optimization we used learning rate of 0.0025, which is different than in the original paper[6]. The reason for this difference might be a smaller mini-batch size (due to 1 GPU setup we train with mini-batches of 2 images, instead of 16 images), or differences between how keras and

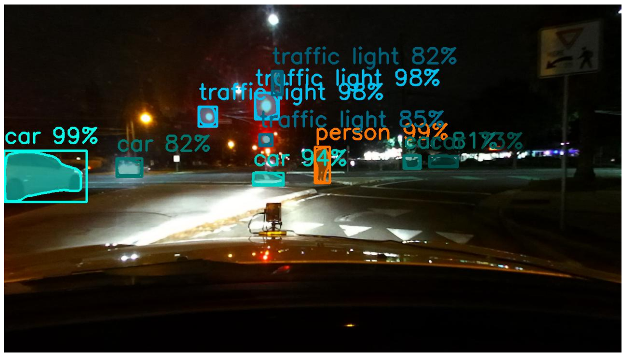


Fig. 6: mRCNN Inference Example

pytorch implement adam optimizer. After visual inspection of the inference results, we decided to use the weights from[1] trained for a much longer time.

2) 2D object detection performance: The model performs very well, even in night driving scenarios (Figure 6). The model is very robust in detecting key self-driving objects such as cars, people and traffic lights. The model also performs very well at detecting object in context and from partial information, such as in the person at the wheel example in Figure 7.

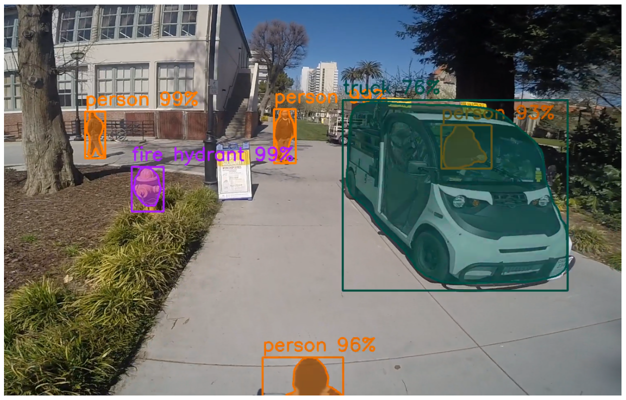


Fig. 7: Inference on SJSU Campus

3) 2D object detection real-time performance considerations: The node does not yield very good real-time performance. A single inference takes around 450ms and we're able to run the model in a stable manner as a ROS node at 2Hz. This inference time is to be expected, as the paper's[6] authors achieved a 200ms inference time.

The state of the art models are capable of achieving results around 20FPS on newest GPU hardware, with some heavily downsized detection models being capable of achieving up to 30-45FPS[24] at the cost of accuracy. While 20FPS is sufficient for some autonomous driving tasks, it is insufficient for safety-critical parts of an autonomous driving system. Possible ways to slightly improve the performance include running the algorithms on multi-GPU setup and improving the memory pipeline for maximum GPU utilization.

C. Radar Pipeline

The radar node subscribes to the point cloud published by the radar driver and performs K-means clustering on the point cloud. The clusters whose points are too sparse are rejected as noise. The node publishes the following information:

- **centroids**: the X,Y position of the center of the cluster
- **sizes (x,y)**: the estimated size of the cluster (bounding box)
- **reflectivity**: total reflectivity of all the points in each cluster.

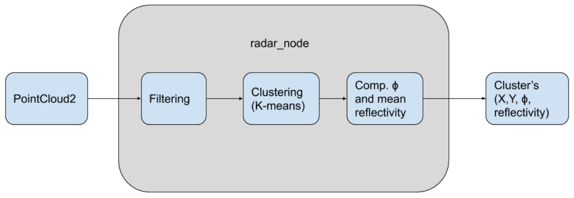

radar_node
PointCloud2
Filtering
Clustering (K-means)
Comp. ϕ and mean reflectivity
Cluster's (X,Y, ϕ, reflectivity)

Fig. 8: Radar Node Schematic

1) Radar pipeline real-time performance: The inference time of the radar pipeline is between 35-40ms, which results in a single node being unable to keep up with the 30Hz publish rate of the radar Point Clouds. In order to ensure processing in real-time, we split the load evenly between two nodes running on separate CPU cores and publishing to the same topic. Possibly, with a faster processor just one node would be sufficient.

D. GPS/GNSS Pipeline

One of the Emlid Reach RTK GNSS modules is set up as a base station, while the other one is set up as a receiver on the car. The 900Mhz radio is used for sending corrections from the base to the receiver.

The node subscribes to the NavSatFix message stream from the GPS driver. After receiving the message, it calculates the distances between consecutive points using Vincenty's Formula. Speed and acceleration are calculated by derivation of the calculated distance. The node publishes the vehicle's heading and current speed, as well as a GPS-based timestamp.

1) GPS pipeline real-time performance: The node operates at the frequency of NMEA frames reception from the GNSS module, which is between 1 and 10Hz. The inference takes less than 1ms.

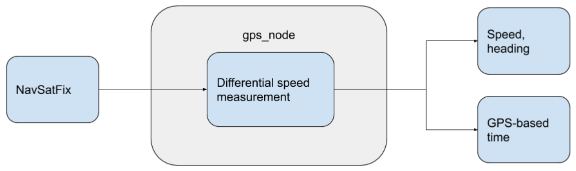


Fig. 9: GPS Node Schematic

E. Radar-Camera Fusion node

The fusion node works in the following way: it subscribes to the cluster information published by radar pipeline and to bounding boxes published by the 2D object detection pipeline. Then, the 90 degree field of view of the radar is mapped onto slightly bigger 110 degree field of view of the camera.

Next, the algorithm iterates over all the detected objects in the image and tries matching the objects with point clusters. If a match is made within the margin of +-10% the object's distance from the camera, the object is associated with the cluster. Figure 11 shows the fusion result overlaid on the radar scan point cloud after clustering.

Finally, the node publishes the combined information consisting of:

- label with confidence score
- X,Y,Z position (in camera's frame)
- object's reflectivity to radar

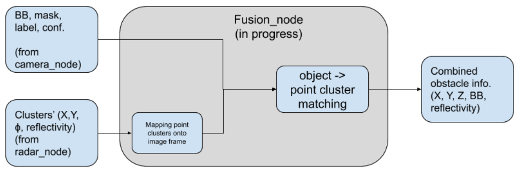

BB, mask, label, conf.
(from camera_node)
Clusters' (X,Y, ϕ, reflectivity) (from radar_node)
Fusion_node (in progress)
Mapping point clusters onto image frame
object -> point cluster matching
Combined obstacle info. (X, Y, Z, BB, reflectivity)

Fig. 10: Radar-Camera Fusion Result

The reflectivity information from the radar is not fully utilized in the fusion. In the future it could be used to differentiate between highly reflective objects (e.g. car body) and the ones with low reflectivity (e.g. bushes at the side of the road).

V. 3D Object Detection Algorithm

A. SqueezeSeg Algorithm

1) Algorithm principle: The SqueezeSeg algorithm projects the 3D point cloud into a 2D spherical surface. Given the projection resolution $\delta\theta$ and $\delta\phi$, each point (x, y, z) in the

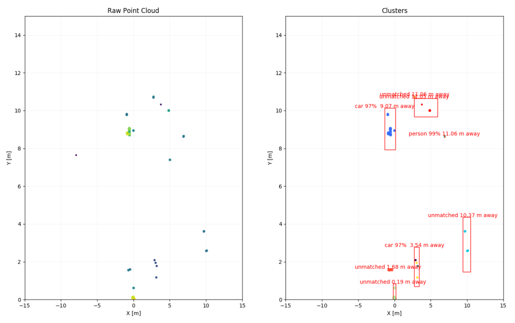


Fig. 11: Radar-Camera Fusion Result

3D space could be projected to a point in the surface at the position (i, j), where,

$$i = \left\lfloor \frac{arcsin(\frac{y}{\sqrt{x^2+y^2}})}{\delta\theta} \right\rfloor \quad (1)$$

$$j = \left\lfloor \frac{arcsin(\frac{z}{\sqrt{x^2+y^2+z^2}})}{\delta\phi} \right\rfloor \quad (2)$$

Each of the point on the surface contains the original information including coordination in 3D space, intensity, etc. The projected samples are further applied to a encoder-decoder structured CNN, with the purpose to further reduce the tensor dimension. The feature extraction is executed on the down sampled space and then up sampled to the original space.

2) Train the network: This part is performed on the SJSU HPC platform, using data set from KITTI, with tensor board to visualize the training procedure. This work has been detailed in Mengshi Li's homework submission, thus we omit it here to save space. To wrap it up, the main challenge is to setup the training environment under HPC, where we have no privilege to install packages and there is no GUI services.

B. Frustum PointNet Algorithm

1) Algorithm principle: The main idea of this algorithm[12], as shown in Fig-12, is to construct a frustum based on the 2D bounding box and the calibration matrix between the RGB image and the 3D point cloud, which reflects the relative geometry information for the camera and the Lidar sensor. The network assumes that there is only one object inside the frustum, and for the points which fall into this frustum, apply them to the PointNet[14] neural network directly, without any voxelization or projection. The PointNet will first do segmentation, then propose a 3D BBox for the clustered points. By constraining points within the frustum, the dimension of the tensor input to the PointNet is of reasonable size for the model training.

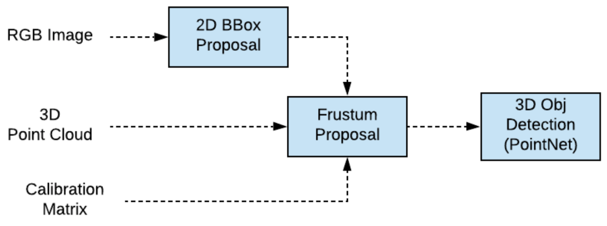
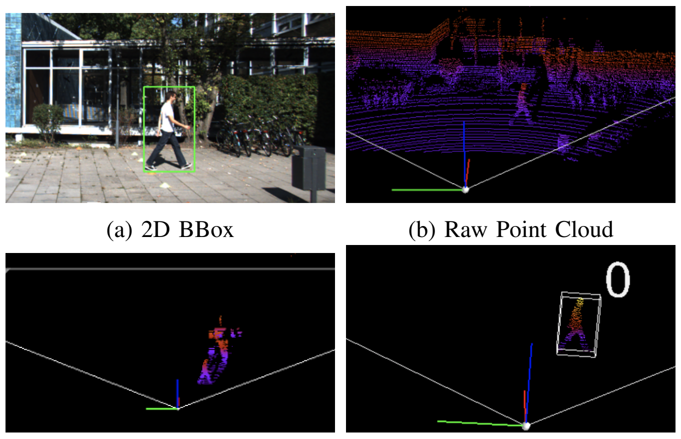


Fig. 12: Major Components of the Frustum PointNet



(a) 2D BBox (b) Raw Point Cloud

(c) Point Cloud inside Frustum (d) Ground Truth 3D BBox

Fig. 13: Data Preparation for PointNet

"Instead of solely relying on 3D proposals, this method leverages both mature 2D object detectors and advanced 3D deep learning for object localization, achieving efficiency as well as high recall for even small objects. Benefited from learning directly in raw point clouds, this method is also able to precisely estimate 3D bounding boxes even under strong occlusion or with very sparse points. Evaluated on KITTI and SUN RGB-D 3D detection benchmarks, this method outperforms the state of the art by remarkable margins while having real-time capability."[12]

2) Training the PointNet model: The training procedure is operated on the HPC platform. The data set we are using is from KITTI.

The key challenge in the training process is to prepare the data for the PointNet neural network, where we need to provide both the points inside a proposed frustum and the ground truth 3D BBox constructed out of the KITTI data set. The data preparation process is illustrated in Fig-13.

First we extract the 2D ground truth bounding box from the KITTI data set, as shown in Fig-13a. Combining it with the calibration matrix, we could construct the frustum, and with it as the boundary, we could select points inside it as the input tensor, as shown in Fig-13c. For training purpose, we also need the 3D ground truth bounding box, as shown in Fig-13d.

Once the data set is ready we could begin the training procedure and observe the evaluation results via tensor board. This is similar to what we did for the SqueezeSeg algorithm.

3) Inference using Frustum PointNet: On top of the git repository[23], we have developed a 3D object detection application. The diagram of this application is similar as Fig-12. It consumes as input the static RGB image, calibration matrix and the 3D point cloud; and outputs the 2D BBox and 3D BBox. Since there is no GUI service on the HPC, we deployed this application on the lab PC HP-870 with GTX-1080 GPU. The main framework/dependencies it is using includes:

- Python 2.7
- CUDA 8.0
- Tensorflow-GPU 1.4.0
- OpenCV 4.1.0
- Mayavi 4.6.2 (to visualize point cloud)

All the Python dependencies are installed inside the Python virtual environment for better isolation and easier mitigation.

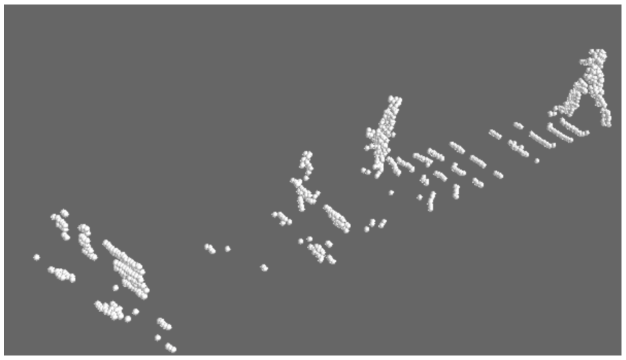
To propose the 2D BBox we need to utilize the mature 2D Object Detection Neural Network model. Supporting of 2D object detection model is straight forward, the user just needs to copy the pre-trained 2D model into the model folder and mark the configuration files accordingly. In this deployment we have included SSD[9], Fast RCNN[17]. The 3D object detection model is supported in a similar way. However the PointNet requires some custom tensorflow operators, which need to be pre-compiled before any inference.

Fig-14 shows a sample output of the application, where Fig-14a demonstrates the frustum point cloud and Fig-14b illustrates the 3D BBox, with yellow points as the vertexes and red point as the center point.

4) More to work on: We have trained the model and done some preliminary inference with the Frustum PointNet model. So far the model takes as input the point clouds of Lidar sensor. However Lidar is expensive, especially under the scenario where we aim to retrofitting an old car. So the first future network is to use stereo camera output as the data input. There are two main challenges for this work. First is that the point cloud from stereo camera is different from that of the Lidar sensor; the second one is that to construct the frustum, we need the calibration matrix between RGB image and the point cloud. This requires us to take into consideration the geometrical features of the stereo camera.

VI. Lane Keeping Algorithm

The lane keeping algorithm that was used is based on the image processing technique known as Canny Edge Algorithm developed by John F. Canny that was called Canny Edge Detector [2]. There are several steps in the image processing pipeline in order to create the result. The pipeline to identify the lane in the every single frame of the input video is a combination of several steps running in a loop using a Python script with OpenCV and Numpy libraries (Figure 15).


(a) Frustum Point Cloud

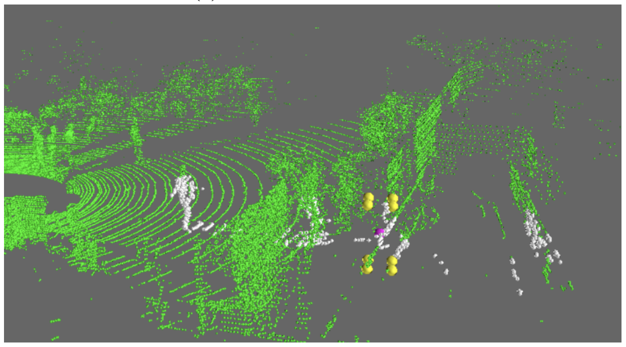

(b) Estimated 3D BBox

Fig. 14: Sample Output of Inference Application

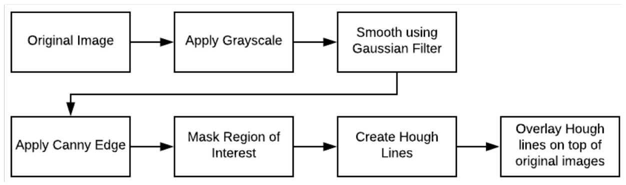
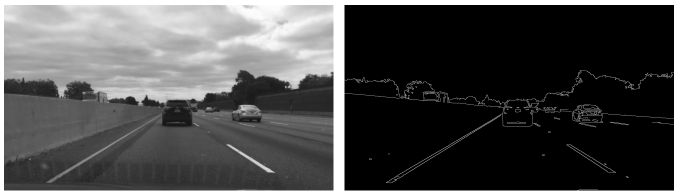


Fig. 15: Lane Detection Pipeline

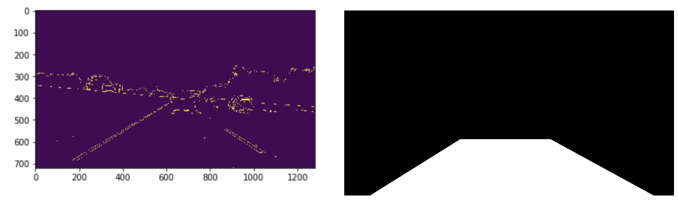
1) Image preparation: Every single frame of the image needs to be prepared in order for the Canny Edge Detector algorithm to work effectively. All raw images will be subjected to noise in the images array. So, the first step is to apply grayscale filter onto the image. This will both ease the image processing technique and reduce the computing time required. The next step is to apply a 5x5 Gaussian filter to remove any noise on the image. This Gaussian filter will average the value of the array on the image to prevent any inconsistent pixel value that is either too high or low. From the visible eye, the image will seem to be blurred (Figure 16a).

2) Canny Edge Detector: As the image is ready, we apply the Canny Edge Detector onto the image. This will filter the image with Sobel Kernel in both vertical and horizontal



(a) Gaussian Filter (b) Canny Edge Detector

Fig. 16: Apply Canny Edge Detector on the image



(a) Image plot (b) ROI Polygon Mask

Fig. 17: Create Region of Interest

direction to get the derivative in both directions, G_x and G_y [10]. The result is as shown in (Figure 16b). Then we will be able to find edge direction and edge gradient as follows:

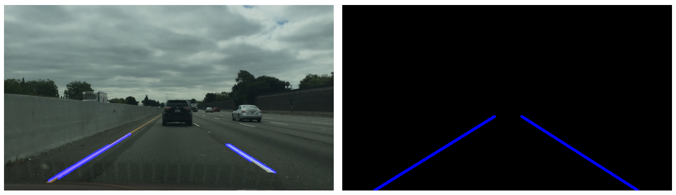
$$Edge_{G}radient(G) = \sqrt{(G_x)^2 + (G_y)^2} \tag{3}$$

$$Angle = tan^{-1} = (G_y/G_x) \tag{4}$$

3) Region of Interest: One important step before creating the line is to to create the region of interest for the Canny Edge. We create a polygon shape for the region of interest to allow only edges within this region of interest will be processed for the next step of creating the line (Figure 17b).

4) Hough Transform: We used the Hough Transform algorithm [11] to identify any lines particularly straight lines within the binary images previously produced. This will create straight lines for any straight edges. However, the length of the lane lines on the road are inconsistent as seen on (Figure 18a). A raw Hough Transform algorithm also produced more than 2 lines with varies length. Therefore, we combine the arrays of Hough Transform's lines on the left with a simple average function to produce one single line on the left side. The same average function applied to the right line. The result will product 2 blue straight lines with the same consistent length (Figure 18b).

5) Video output processing: We placed the the image of the lines on top of the original color image to create the perception for the car to see the lane lines on the road. We then modified the python script to take the video as an input and display the output of processed video at the same time. This lane detection python script can be flashed into Nvidia Jetson Nano or Rasberry Pi module to produce real-time video processing. Some of the challenges noticed based on



(a) Edge Detected (b) Lane Lines Detected

Fig. 18: Smoothen the Thickness and Fix the Length of the Line

the result of this work, it is not yet robust and not applicable for all type of road lines. Sometimes, there's only a line on one side of the road and it will crash the running script. The lane detection was tested successfully on a PC using a recorded video. An example of the final result of the lane detection is as shown on Figure 19.

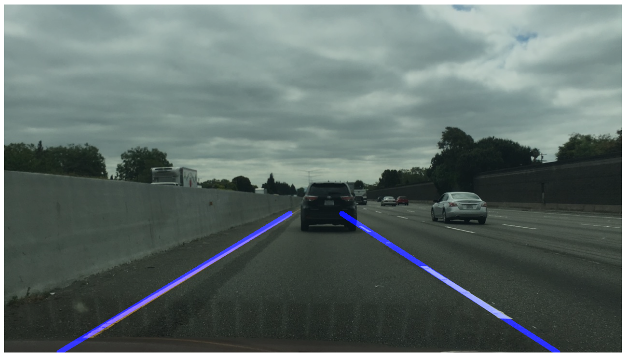


Fig. 19: Final Result for Lane Keeping

VII. Conclusions and Future Work

In this project we have developed an end to end perception system for an autonomous driving assistant. With Jetson Nano based sensor platform, we have collected data in the format of rosbag. We have studied several perception algorithms.

The key future work includes testing the system on a faster computing platform, capable of running the algorithms in the car in real time. Regarding the camera-radar pipeline, we plan to implement pedestrian tracking based on existing radar-camera fusion. Regarding the 3D object detection algorithm, we intend to take stereo camera point cloud to the model. But the most critical task is to improve the speed and accuracy of detection, we would look into the CenterNet for further study.

Acknowledgment

We would like to thank Prof. Kaikai Liu for lending us the hardware that made this project possible; we also want to thank Prof. Wencen Wu for providing access to HPC to facilitate our project.

Team Member Contributions

As shown in table I.

TABLE I: Team Member Contribution Dashboard

Task	Owner
Hardware setup	T. Lewicki, M. Li
ROS Architecture	T. Lewicki
Radar Pipeline	T. Lewicki
2D Camera Pipeline	T. Lewicki
RTK GPS Pipeline	T. Lewicki
SqueezeSeg Alg.	M. Li
Frustum PointNet Alg.	M. Li
Lane Keeping (Canny Edge)	S. Mukhre
Lane Keeping (Hough Lines)	S. Mukhre

REFERENCES

[1] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017.

[2] John Canny. A computational approach to edge detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1986.

[3] PSA Peugeot Citraeon. Avoiding accidents: Develop driver warning/information systems (lane following, bar codes). web.archive.org/web/20051017024140/http://www.developpement-durable.psa.fr/en/realisation.php, 2005.

[4] Mehrdad Dianati Saber Fallah David Oxtoby Eduardo Arnold, Omar Y. Al-Jarrah and Alex Mouzakitis. A survey on 3d object detection methods for autonomous driving applications. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, 2019.

[5] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[7] F. M. Moreno D. Cruzado F. Garcia J. Beltran, C. Guindel and A. de la Escalera. Birdnet: a 3d object detection framework from lidar information. *ITSC*, 2018.

[8] J. Lee A. Harakeh J. Ku, M. Mozifian and S. Waslander. Joint 3d proposal generation and object detection from view aggregation. *IROS*, 2018.

[9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[10] OpenCV. Opencv documentation on canny edge detector. https://docs.opencv.org/master/da/d22/tutorial_py_canny.html, 2017.

[11] OpenCV. Opencv documentation on haughlines transform. https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html, 2017.

[12] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. *arXiv preprint arXiv:1711.08488*, 2017.

[13] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[14] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[15] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[16] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[17] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[18] Green Car Reports Stephen Edelstein. Tesla model s adds 'speed assist,' lane-departure warning. https://www2.greencarreports.com/news/1094773_tesla-model-s-adds-speed-assist-lane-departure-warning, 2017.

[19] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. *ICRA*, 2018.

[20] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. 2019.

[21] J. Wan B. Li X. Chen, H. Ma and T. Xia. Multi-view 3d object detection network for autonomous driving. *CVPR*, 2017.

[22] Philipp Krhenbhl Xingyi Zhou, Dequan Wang. Objects as points. *CVPR*, 2019.

[23] Klein Yuan. Detect object in 3d with point cloud and image. https://github.com/KleinYuan/tf-3d-object-detection. Accessed: 2019-05-10.

[24] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018.