Integrating a Motion Prediction Baseline into an Autonomy Stack (Report)

Mahir Gulzar
Institute of Computer Science
University of Tartu
Tartu, Estonia
mahir.gulzar@ut.ee

***Abstract*—The development of a software autonomy stack for self-driving cars involves various modules such as localization, perception, prediction and planning etc. Existing open source vehicle autonomy platforms develop these modules by making use of state-of-the-art research in each module. In this report, we explore the motion prediction module of an open source autonomy stack called Autoware. For this, a literature review was conducted over existing motion prediction baselines. We intend to reduce the gap between academic research and applied work by exploring ways to integrate a motion prediction baseline into Autoware and apply it on a real vehicle.**

***Index Terms*—autonomous vehicles, motion prediction, autoware, ROS (Robotic Operating System)**

I. Introduction

Since the Darpa 2007 urban challenge, the autonomous vehicle (AV) industry has evolved rapidly. With rapidly growing literature coming from both academia and industry to build self driving vehicles, more and more researchers are contributing in building up mature open-source based solutions to help the AV research community in streamlining AV development process. Developing a software autonomy stack for self driving vehicle involves various sub-modules. There are plenty of proposed software architectures for an AV software stack but all of them are essentially following a sequence of information flow from real-world sensing to all the way up-to actuation or control as shown in figure-1.

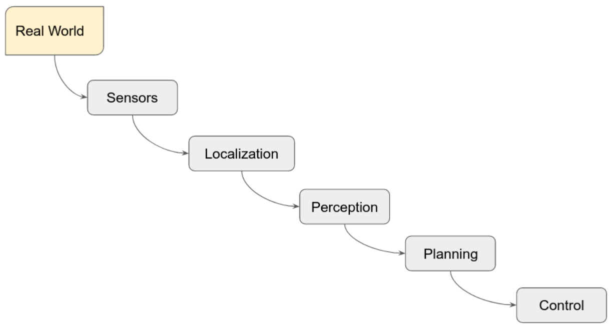

Fig. 1. Modules of an autonomy stack

There are plenty of open source based solutions for kick starting the development of an AV. The two most noted ones are Apollo [1] by Baidu and Autoware [2] by Autoware foundation. A comprehensive comparison of these two freeware architectures is done by [3]. Here, the final comparison between these two architectures is summarized by risk vs quality curve as shown in figure-2. Here, the risk factors include modularity, complexity, developer manuals etc and tells that up to what level it opposes the further development of the platforms whereas the quality factors tells the general applicability of these platforms across various scenarios. It is to be noted that since the publishing year of this article i.e. 2019, both of these autonomy stacks have undergone major upgrades. For example, Autoware's latest version is Tier IV architecture [4] and Apollo's latest version is Apollo 6.0 [1].

This report is centered around exploring the motion prediction module of autonomous driving. To be specific, we are interested in forecasting the future motion patterns of other vehicles. The main challenge of approaching such a problem is to have high quality track histories of other vehicles in addition to this the context of the environment should be considered before a motion prediction module outputs the motion patterns of other vehicles. Further details about motion prediction module and the types of existing models is explained in motion prediction baselines section.

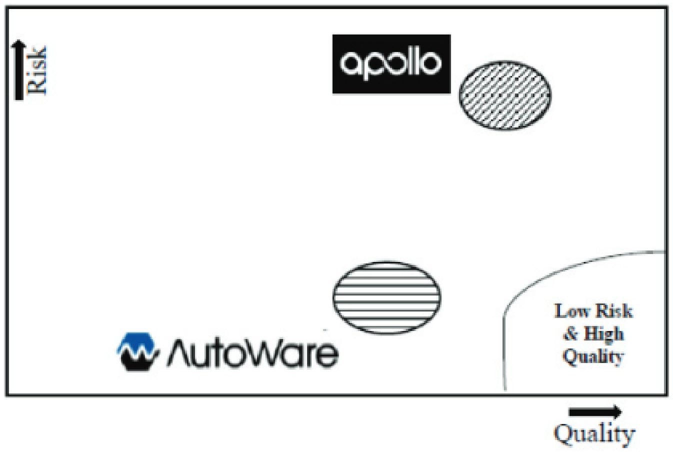

Fig. 2. Risk vs Quality curve for Autoware and Apollo. Source [3]

II. Choice of Stack and its challenges

University of Tartu's Autonomous Driving lab (ADL) is developing a self driving vehicle as a part of an on going project with Bolt. The autonomy engineers in ADL make use of open-source off the shelves solutions to build the AV. Our current choice of autonomy stack is Autoware mainly because this autonomy stack is quite modular and every piece of code is accessible to the developers with plenty of contributions from the AV developer community. We use Autoware.Ai [5] as our autonomy stack. While Autoware gives us very strong baselines to start working with, we still lack in terms if good motion prediction baselines. The motion prediction module of Autoware.AI makes use of a third party software called Open Planner (OP) [6]. Open planner predicts future motion estimates and trajectories of other vehicles using particle filter and fuses HD map information into the prediction. The focus of this report is to research a simplistic yet strong data driven motion prediction baseline which follows the current trends in motion prediction research and give a proof of concept to integrate this baseline into Autoware.

A. *Current Prediction Functionality*

As mentioned earlier, Autoware.ai uses open-planner for prediction, planning & control. Open-planner's prediction functionality uses Autoware's vector map information along with tracked obstacle information and simulates forward motion of the tracked obstacle based on its velocity and orientation up to a fixed number of future time steps.

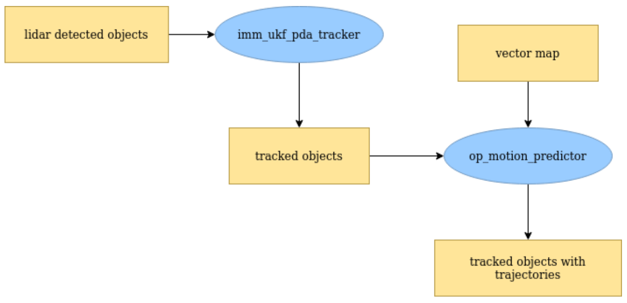

Fig. 3. An illustration of information flow from detection to prediction step. Here blue ovals represent ROS nodes and yellow rectangles represent data structures (ROS messages) holding relevant information which ROS nodes use and process.

Figure-3 shows the information flow for prediction in Autoware.ai. The lidar detections are passed on to a IMM_UKF_PDA tracker where IMM abbreviates to interactive multiple models, UKF abbreviates to uncented kalman filter and PDA abbreviates to probabilistic density association. The op_motion_predictor node subscribes to vector map and tracked objects. For each tracked object, a set of candidate trajectories are generated (one or more modes) where predicted trajectories are then ranked with confidence values using a particle filter. Once tracked objects are populated with candidate trajectories, this information is published as predicted objects which is then passed over to planning module.

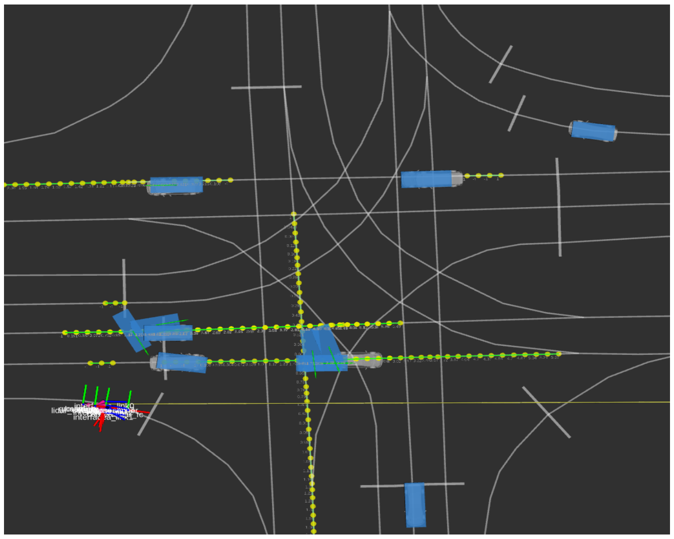

Fig. 4. A top-down snapshot of Rviz (ROS visualzation window). Here, blue rectangles denotes generated tracks for vehicles where small green arrows point towards heading of the respective vehicles. Light gray lines denote lane markings where predicted trajectories are shown on top of some lanes

The above figure-4 gives a rough idea that how open-planner's prediction look like. The very first limitation we already see here is that the trajectories are bounded by the lane markings i.e. trajectory will always be predicted on a lane. One problematic scenario for such predictions is lane change maneuver i.e. if there is no connection between adjacent lanes and a target vehicle (TV) intends to do a lane change then the current prediction module won't be able to predict this lane change maneuver. Here, TV is a vehicle of interest whose trajectory we intend to predict. In addition to this, the current prediction module doesn't take into account behaviour of other surrounding vehicles (SVs) e.g. if a vehicle ahead of TV applies emergency brakes, this would effect the behaviour of TV but the prediction module is not aware of it. In short, the current behaviour prediction module works but lacks in incorporating more contextual information and is strictly bounded by the lane centre lines.

The prediction module expects to have good track estimates which include track position, orientation and velocity. Figure-5 shows an example of wrongly estimated tracks for a vehicle. A tracker should associate only one detection to a vehicle but in this example there are three. These wrong positional and orientational estimates are propagated ahead to prediction module which falsely assumes that vehicle has switched directions giving false prediction estimates for it.

B. *Other Stacks*

In comparison to OP, Autoware's tier iv architecture does something very similar. They do map-based predictions using lanelet path. They generates several trajectories based on the map lanes and mark them with confidence values similar to how its done in OP. Here, instead of using particle filter based predictions they use a simple confidence metric that uses distance of the obstacle to the corresponding lane and

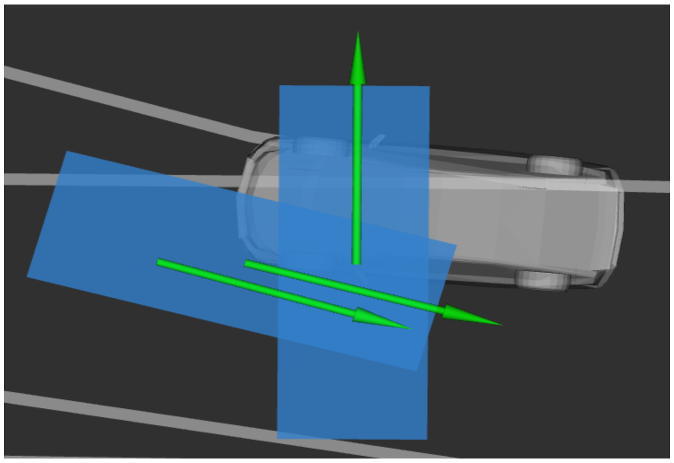

Fig. 5. An example of false tracking estimates where gray car model shows the detection from lidar detector while blue arrowed boxes represent track estimates.

assigns higher confidence to a trajectory where the obstacle distance is less [7].

$$confidence = 1/d \tag{1}$$

where d denotes the distance to the lane center.

In comparison to Autoware's prediction functionalities, the prediction module in Apollo make use of top-down rasters where the context of the scene is provided in a top-down view to a CNN (Convolutional Neural Network) which extracts map and obstacle features. The obstacle features are embedded into an RNN that captures temporal changes while map features guides the predicted estimates to follow semantic context of the scene [8].

III. Motion Prediction Baseline

Motion prediction in autonomous driving has seen a decent amount of literature in during the past decade. [9]–[12] surveys vehicle motion prediction literature categorizing models based on some proposed taxonomies. As we are mostly interested in a data driven or in other words a pattern-based motion prediction model as a baseline, We leverage the model categorization proposed by [10]. A pattern based model can classified based on the model's input type and output type. This is illustrated in figure-6.

In figure-6, the highlighted input representations along with semantic view of the map can give a better context of the environment similar to what Apollo's prediction module is using. The output can either be unimodal or multimodal. Ideally, multi-modal trajectories gives more information about possible future behaviour of a vehicle but in this work we can settle with any of them. The reason of trimming the input and output type here is to reduce the search space of finding a baseline model.

There are plenty of datasets available for vehicular motion prediction. For example, some of the well known ones are Agroverse dataset [13], Nuscense dataset [14], Lyft l5 motion

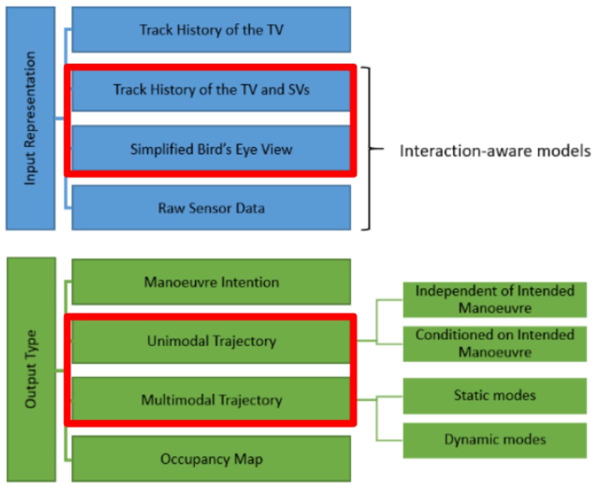

Fig. 6. Model categorization taken from [10]. Here the red highlighted sections are some traits which we want a motion prediction baseline to have.

prediction dataset [15] and Waymo open motion dataset [16] etc. While experimenting with these datasets and available motion prediction models trained over them, we chose to work with Lyft dataset as Lyft's library for preprocessing the input data and training motion prediction model is well documented and convenient to work with. While looking for a strong baseline to work with, we stumbled upon [17] which is one of the winning solutions of Lyft dataset. This baseline uses a simple CNN with Resnet [18] backbone and predicts multimodal trajectory output. This baseline is illustrated figure-7 below.

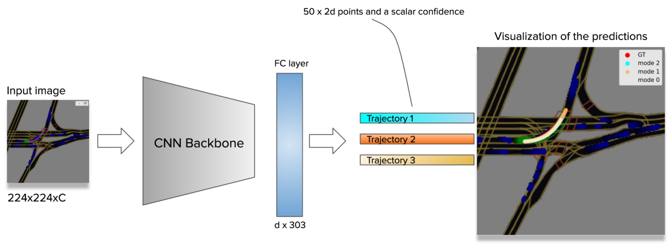

Fig. 7. A simple yet strong motion prediction baseline. Source [17]

A simple and working implementation of such a model is available [19]. The model takes in top-down rasters and semantic map and outputs multimodal trajectories. Formally, the input and output shapes are given as follows:

$$input\_channels = 2(h+1)+3 \tag{2}$$

$$output = (T \times 2 \times k) + c \tag{3}$$

Here, in equation-(2), h denotes the number of history frames. Here, $h+1$ is multiplied by 2 to take into account both ego vehicle and TV past histories. The trailing addition of 3 channels is for concatenating top-down semantic rasters of the corresponding scene. Figure-8 shows an illustration of how these input rasters look like.

On the other hand, T in output size denotes number of future prediction timesteps where k is number of modes and c is confidence value for all predicted trajectories. Intuitively, c = k in equation-(3).

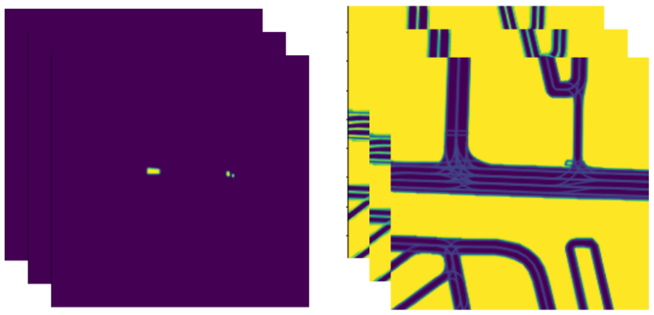

Fig. 8. An illustration of top-down input rasters to [19] model. The left rasters show h history frames of vehicles while on right there are three different contrasts of top-down rastered semantic map of the corresponding scene.

IV. Pipeline

As autoware.ai is built on ROS. Let us first define the core steps of the integration pipeline. Assuming we have perfect object detections and track histories to work with, we intend to write a ROS node which implements a rasterization module that takes in HD map information from autoware and past history of tracks of detected vehicles and generate a top-down raster encoding the complete semantic information of the scene. Afterwards, a baseline model that most probably will be trained on our data takes these rasters and outputs trajectories of intended vehicle. These trajectory outputs will then be passed back to ROS for the planner module to work on. These steps are illustrated in figure-9 below.

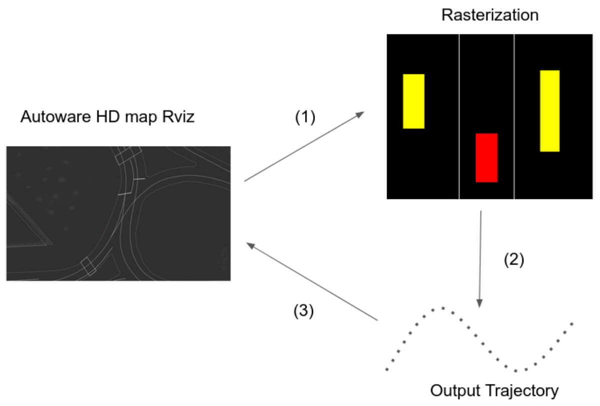

Fig. 9. An illustration of integration pipeline loop where information of HD maps along with object tracks is rasterized into an image at each timestep, this raster is then passed to a pattern based motion prediction model and output trajectories are extracted

Although the illustration in figure-9 gives an overview of the integration pipeline, there are many essential details obscured within the pipeline itself. The idea of doing this integration is to apply this pipeline on a real vehicle. Datasets which are usually available online are cleaned and preprocessed for researchers so that they can readily make use of them and come up state-of-the-art modelling techniques. In our case, we don't intend to improve some online benchmarking scores instead we intend to apply the domain knowledge gathered through different modules of an autonomy stack and upgrade the prediction functionality on a real vehicle. This means whatever bottlenecks and errors we have in modules prior to prediction module as shown in figure-1 needs to be corrected in parallel. Looking at bigger picture, we decided to make improvements to other modules first whose outputs are taken in by the prediction module. Specifically, we first improve the perception stack i.e. the object detection & tracking before hooking an external baseline into our stack.

A. Improving Object Detection & Tracking

The very first bottleneck comes from the object detection. For object detection we do ground filtering and clustering over LiDAR points, and afterwards a 3D bounding box fitted over the clustered points. This object detection suffers alot for correctly estimating the actual shape of an object. On top of that, we don't get any classification of the detected objects which means we can't really differentiate between a dynamic and static obstacle. As we intend to predict trajectories of vehicles only, using clustering based detector we don't really know which obstacle is a vehicle. To get rid of this problem, we employee a 3D LiDAR based object detector called Point-pillars [20]. Luckily, there is an open-source pretrained implementation of pointpillars model available in Autoware.AI. We applied and tested pointpillars detections and compared the results as shown in figure-10 below.

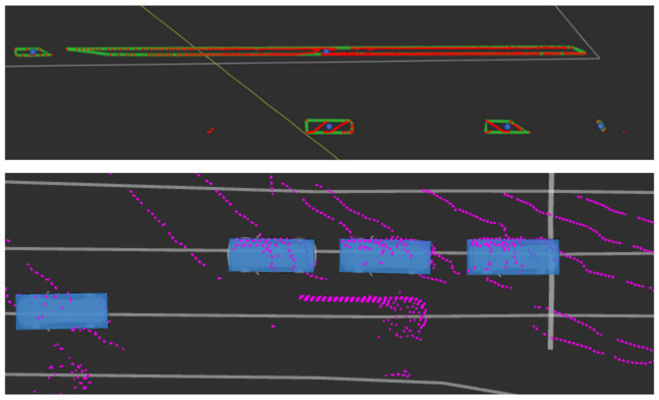

Fig. 10. A comparison of clustering (top) vs pointpillars (bottom) based detector where clustering based detector shows building as an obstacle whereas pointpillars based detector gives precise detection of dynamic obstacles which here are vehicles.

Once we have object classification, we can then apply tracking on those objects which are classified as car or vehicle. As explained previously in figure-5, the tracker provided by Autoware.Ai gives wrong positional and orientational estimates.

One of the root cause of wrong orientational estimates is the wrong orientational estimates by the detection model itself. For example, pointpillars model can roughly estimate the shape of car but it gets confused in assigning forward or reverse direction to it. This is probably because LiDAR points look very similar for a vehicle going forward or coming towards us. This leads, to approximately 180 degree rotation error in direction of the detected objects. The IMM_UKF_PDA tracker tracks the orientation of the the vehicles. When a wronge orientational estimate is passed to the tracker as latest measurement for a vehicle, it assumes that vehicle has made a quick turn or something similar. This results in wrong tracking estimates.

To rectify this problem, we wrote our own simple constant velocity (CV) nearest neighbour data association (NNDA) tracker that tracks the positional estimates of the object only. So by not tracking the turn rate of a vehicle, we remove the possibility of having wrong orientational estimates. Though, we still need to assign a direction to the vehicle's track for which we take the last known orientation of the associated detection and assign it to the corresponding vehicle track.

B. Rasterizer

The motion prediction baseline we considered needs rasterized top-down input where target vehicle and semantic map is rasterized on an image. To achieve this, we wrote a ROS node that subscribes to tracked objects provided by the NNDA tracker we discussed previously. This node iterates over the tracked objects (TV's & SV's) and keeps last h history of the active tracks. These history frames along with vehicle dimensions and orientations are passed over to rasterization function that transforms 3D positions and orientations of tracked objects to a 2D image relative to the ego-vehicles frame.

V. Conclusions & Future Work

In this report, we explored the motion prediction module of Autoware. We discussed the short comings of current prediction functionality and compared its working with Apollo. We briefly explored what motion predictions baselines are available on some well known datasets and how can we make use of them. We also, discussed the bottlenecks and errors which originate in our perception stack and how they are propagated into the motion prediction module. We tried to rectify the problems in object detection and tracking by using a mature 3D detector and writing our own CV-NNDA tracker. We discussed the input representation of the chosen baseline and lastly discussed the rasterization module that generates input for the chosen baseline.

At the time of finalizing this report, we are currently working on improving the rasterization module. For future work, following are the steps we are planning to do:

1) Similar to TV's and SV's, translate Autoware's vector map into a top-down semantic image.
2) Using the motion prediction baseline discussed above, train another network on our own rasterized data.
3) Translate network outputs into OP's understandable format and compare its performance to OP's prediction module.

References

[1] ApolloAuto, "Apollo," https://github.com/ApolloAuto/apollo, 2017, [Online; accessed 23-November-2021].

[2] S. Kato, "Autoware," https://www.autoware.org/, 2017.

[3] V. M. Raju, V. Gupta, and S. Lomate, "Performance of open autonomous vehicle platforms: Autoware and apollo," in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. IEEE, 2019, pp. 1–5.

[4] A. T. IV, "Autoware tier iv overview," https://tier4.github.io/autoware.proj/, 2019, [Online; accessed 23-November-2021].

[5] Autoware.AI, "Github Autoware.AI," https://github.com/Autoware-AI/autoware.ai, [Online; accessed 23-November-2021].

[6] H. Darweesh, E. Takeuchi, K. Takeda, Y. Ninomiya, A. Sujiwo, L. Y. Morales, N. Akai, T. Tomizawa, and S. Kato, "Open source integrated planner for autonomous navigation in highly dynamic environments," *Journal of Robotics and Mechatronics*, vol. 29, no. 4, pp. 668–684, 2017.

[7] Autoware, "Autoware Tier IV Prediction," https://github.com/tier4/AutowareArchitectureProposal.iv, [Online; accessed 23-November-2021].

[8] K. Xu, X. Xiao, J. Miao, and Q. Luo, "Data driven prediction architecture for autonomous driving and its application on apollo platform," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 175–181.

[9] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH journal*, vol. 1, no. 1, pp. 1–14, 2014.

[10] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis, "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[11] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 895–935, 2020.

[12] M. Gulzar, Y. Muhammad, and N. Muhammad, "A survey on motion prediction of pedestrians and vehicles for autonomous driving," *IEEE Access*, 2021.

[13] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan *et al.*, "Argoverse: 3d tracking and forecasting with rich maps," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8748–8757.

[14] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[15] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," *arXiv preprint arXiv:2006.14480*, 2020.

[16] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou *et al.*, "Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset," *arXiv preprint arXiv:2104.10133*, 2021.

[17] A. Sanakoyeu, "Winning solution for Kaggle challenge: Lyft Motion Prediction for Autonomous Vehicles," https://gdude.de/blog/2021-02-05/Kaggle-Lyft-solution, [Online; accessed 23-November-2021].

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] M. Gulzar, "Behavior prediction model for self driving vehicle," https://mahirgulzar343.medium.com/behavior-prediction-model-for-self-driving-vehicle-19b8664d6a7f, [Online; accessed 23-November-2021].

[20] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.