Learning interactions with Micro-mobility vehicles for Self-Driving cars

Deepak Talwar, Seung Won Lee, Abhishek Singh and Yang Chen
Department of Computer Engineering, San José State University
San José, CA
deepak.talwar@sjsu.edu, seungwon.lee@sjsu.edu, abhishek.singh@sjsu.edu, yang.chen.sv@sjsu.edu

***Abstract*—**

I. INTRODUCTION

The transportation industry is currently undergoing multiple major shifts. A few of the notable shifts are the rise in ride-sharing and personal mobility devices. While ride-sharing is increasing the number of riders per vehicle [1], personal mobility – also known as micro-mobility – is providing new ways for individuals to transport themselves in urban environments, fulfilling their first-mile and last-mile needs. The emergence of these shifts can be attributed to advancements in technology, needs for new use-cases as well as a renewed availability of large funding sources [2].

These shifts are not only changing how people transport themselves, but also show that the future of urban transportation is multi-modal [3], [4], [5] with different kinds of vehicles designed to fulfill very specific needs. This is a marked difference from a world where cars were the central, most important means of transportation. Although cars are the most versatile type of vehicle, they are also very inefficient for personal transportation needs. The low car occupancy rate is the single biggest contributor of congestion and jams in urban environments [6]. Given that this is poised to change, self-driving vehicles operating in urban-environments will also need to be able to safely and efficiently interact with and navigate around these newly added less versatile but efficient single-purpose vehicles.

While there are expected to be many different kinds of micro-mobility vehicles, ranging from bikes to scooters to self-navigating food delivery robots [7], the most popular and widely available types of vehicles in urban settings are personal transportation vehicles. These include vehicles such as electric scooters from companies like Lime™[8] and Bird™[9], skateboard-esque vehicles from Boosted Board™[10] to One-wheel™[11] and other smaller vehicles collectively known as hoverboards etc. Although all these vehicles are meant to transport a single person at one time, they have varying appearances, movement models, learning curves and safety. In addition, the usage of these vehicles is not very well defined for example, while skateboards and hoverboards can be used on sidewalks, electric scooters and bicycles are only meant be used in dedicated lanes, but can sometimes occupy entire lanes. The usage patterns also differ depending on the availability of dedicated lanes and other infrastructure such as docks and charging stations.

Given this variability in movement dynamics, usage patterns and appearances, the task of detecting, and predicting paths for these vehicles is a non-trivial one. Self-driving cars will need to expertly be able to detect, localize and predict paths for these vehicles in order to avoid hitting them. What makes this task even more challenging is the fact that these vehicle are small and difficult to spot from a distance. Although, self-driving cars are good at detecting pedestrians, it is difficult for them to distinguish between a pedestrian and a person riding one of these vehicles. It is not difficult to see why this is a major problem – max speed of a pedestrian on average is 2.8 miles per hour [12], whereas, max speed of an electric skateboard is 24 miles per hour [13]. A false detection of a skateboard rider as a pedestrian may lead to a widely wrong predictions of their future path and may lead to collisions and even fatal crashes.

In this project, we attempt to improve self-driving cars' capabilities of safely interacting with micro-mobility vehicles through the means of simulation. First, we create realistic 3D models of popular micro-mobility vehicles using computer-aided design (CAD) software and place them in the simulation environment provided by the LG SVL Automotive Simulator [14]. Second, we configure physics for a subset of the vehicles to recreate realistic movement. We then collect data from perception sensors provided by the ego-car in the LG SVL Automotive Simulator and use that data to train 2D and 3D object detection models. Lastly, we evaluate the performance of Baidu Apollo Autonomous Driving platform [15] with the introduction of micro-mobility vehicles in the environment and identify improvements.

The main contributions of this work are (i) 3D CAD models of popular micro-mobility vehicles, (ii) a modified simulation environment based on LG SVL Automotive Simulator [14] including micro-mobility vehicles, (iii) two fully annotated datasets with 10,000 datapoints each that can be used for custom training; these datasets include data from the main camera, depth camera and LiDAR sensors, and ground truth annotations in 2D and 3D, (iv) fully trained 2D Bounding box detection model in V based on YOLOv3 [16], (v) implementation of YOLO3D 3D Bounding box detection model [17] in Keras [18] with TensorFlow [19] backend, and (vi) performance evaluation of Baidu Apollo [15] with micro-mobility vehicles in simulation.

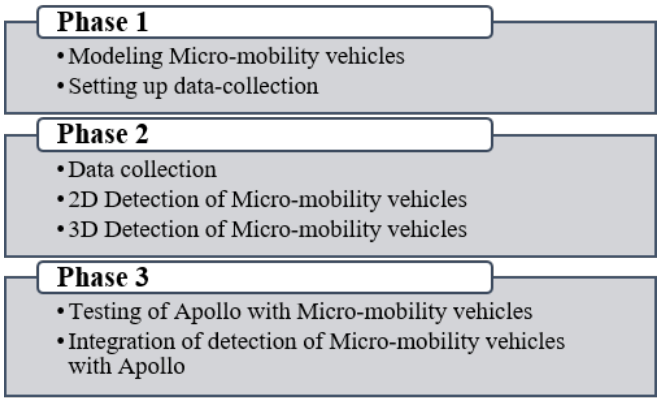

Fig. 1. Project Phases

Fig. 2. LG SVL Automotive Simulator [14]

II. PROJECT PHASES

We split this project into three phases. Figure **??** describes the phases and the goals for each phase. Please note that this report describes the procedures and results from phases 1 and 2, while phase 3 is left as future work. We do include results from our preliminary testing of the e-scooter model with Apollo from phase 3 in VII.

III. SIMULATION ENVIRONMENT

A. LG SVL Autonomous Driving Simulator

The simulation environment that we chose for this project is the LG SVL Automotive Simulator built by LG Silicon Valley Lab (LG SVL) [14]. This is a Unity [20] based simulator that provides out-of-box integration with Baidu's Apollo - Open autonomous driving platform [15]. It also provides a controllable car platform with an array of simulated sensors such as LiDAR, cameras, depth camera, radar, GPS etc. This allows for algorithms to be developed on the data collected from the simulator, and subsequently be tested on the same car. The simulator also allows for modifications to be made to the 3D environment, as well as quickly generate HD maps from the environment. Figure 2 shows a screenshot of the LG SVL Automotive Simulator with LiDAR sensor and 3D Ground Truth sensors turned on.

The flexibility and ease-of-use of the LG SVL Automotive Simulator made it a good choice for the purpose of this project as many modifications were needed to achieve our intended goals.

B. Addition of new vehicles

1) Modeling Micro-Mobility Vehicles: We decided to model 5 different kinds of micro-mobility vehicles:

- Electric scooter
- Hoverboard
- Skateboard
- Onewheel XR™[11]
- Segway Personal Transporter™[25]

These vehicles were chosen because for their varied appearance and motion models, and their popularity in urban environments. Figure 3 shows the process followed for modeling of these vehicles. Since the e-scooter will be used to test Apollo's ability to detect and avoid micro-mobility vehicles, we needed to model it such that its motion could be as realistic as possible. To that end we modeled the scooter's handlebar, its wheels and its body as separate objects constrained in a single assembly. For simplicity, all other vehicles were modeled as combined solid models, meaning that the wheels could not be rotated independently on their axis with respect to the vehicle model. Video at ?? shows how e-scooter's components can be manipulated separately.

After importing the vehicles into Unity, we added a `BoxCollider` component to the entire vehicle including the rider. The `BoxCollider` component determines the 3D boundary of the vehicle and triggers calculations for collisions as the vehicle moves and collides with other `GameObjects`. `BoxCollider` component is also what keeps vehicle in contact with the ground. Without a `BoxCollider`, the vehicle would just fall through the ground as no collisions would be triggered to calculate the position of the vehicle with respect to the ground. The `BoxCollider` component's dimensions are adjusted to fit the vehicle tightly. `BoxCollider` component's dimensions also define the Ground Truth Bounding box's dimensions which is why setting the dimensions correctly is extremely important. Its boundaries can be seen in 3(f).

To make handling of these vehicles easier, we created separate layers for each vehicle type listed above and distributed them into these layers. Prefabs for every type of vehicle were then saved.

2) Set up for data collection: After creating prefabs for the new micro-mobility vehicles, we duplicated each vehicle 30 times and placed them in different locations with different orientations in the San Francisco Downtown scene provided by LG SVL Simulator. Having multiple instances of the vehicles allows us to capture many instances of the same vehicles in different locations quickly. Since there is no automated way of placing the vehicles in the scene and we had to manually place the vehicles on different roads and sidewalks, the distribution of the vehicles was not entirely uniform. This resulted in more vehicles being placed in the beginning of the route, and fewer at the end, which contributed to varying frame-rate performance during the

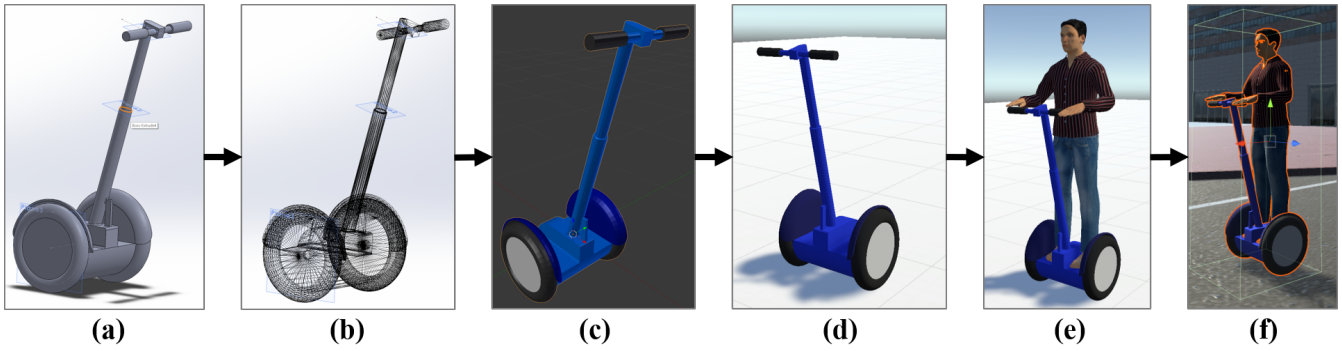

Fig. 3. This figure shows the design process for micro-mobility vehicles created for the project. The first step is to model the vehicle as a solid model in a solid-modeling CAD software. This step is performed in SolidWorks [21] and is shown in (a). The solid model is then converted into a mesh and saved in stereolithography (STL)[22] format as shown in (b). This file is then imported into Blender[23] which is a 3D mesh modeling, animating and rendering software. In this step (c), materials and colors are added to the mesh, and the origin of the model is translated to the center of gravity of the vehicle. The local axis is then rotated to follow the defaults followed by Unity[20]. This step is extremely important as a wrong axis configuration will lead to unexpected movement behavior in Unity. The mesh model is then imported into a test scene inside Unity (d). In this step, we ensure that the axis orientation is correct and scale the model to the correct dimensions if needed. We then add a RigidBody component to it and add a realistic mass in kg to it. Once the vehicle is correctly configured, we add the human model to the vehicle. The humanoid is downloaded from [24]. By default the human is positioned in a T-orientation. We adjust the joints of the human to conform it to the vehicle. This step is performed as a static movement or as an animation depending on the intended usage. The human and vehicle are then grouped under a single Prefab and saved in the Prefabs folder of the Unity project. In step (f), the prefab is imported into the San Francisco simulator scene where it will be used for data collection.

route. For instance, while we achieved average frame-rate of only 12 FPS at the beginning of the route, we were able to achieve average frame-rate on 30 FPS near the end of the route. Although this did not effect data collection, it did result in poorer real-time detection performance at the beginning of the route compared to at the end as explained in ??.

C. Modifications to Ego-car

LG SVL Automotive Simulator [14] provides an ego-car configured with sensors to work with Apollo [15]. We used this car as the starting point for our modifications. This ego-car is equipped with sensors to perceive its environment. Apart from the simulated counterparts of physical perception sensors such as LiDAR, cameras and RADAR, this car also includes sensors to "sense" ground truth bounding boxes (both in 2D and 3D) for obstacles. By default, however, these sensors only detect ground truth bounding boxes for existing NPC (Non-playable characters) cars, trucks and pedestrians and not for the newly added micro-mobility vehicles. To output bounding boxes for the new micro-mobility vehicles, we made the following changes.

1) Adding Ground Truth Sensors: In order to detect ground truths for micro-mobility vehicles, we added two additional sensors to the ego-car – `MMGroundTruth2D` and `MMGroundTruth3D` – for 2D and 3D ground truth bounding boxes respectively. These sensors are similar to the existing ground truth sensors except that they only output boxes for micro-mobility vehicles' layers. Bounding box colors are defined in this sensor and toggle switches to turn these sensors on or off are also added.

2) Modifying Culling Masks: Users can selectively choose the layers that perception sensors in the car, such as cameras, LiDAR and depth sensors, can "see" by choosing them the `CullingMask` selector. By default, the newly added layers for micro-mobility vehicles are not added to the `CullingMask`. Therefore, the new vehicles need to be selected in the `CullingMask` selector for the sensors to render them.

3) Modifications to Perception sensors: The number of channels in the LiDAR sensor were changed from 16 (default) to 64. In addition, motion blur was removed from the `DriverCamera GameObject` as it resulted in blurry images at lower frame rates.

4) Modifying `NeedsBridge` list: The `NeedsBridge` list contains references to all components that are only instantiated when a ROSBridge server is active and the simulator is connected as a client. This is done so that the simulator does not collect and publish messages to topics if they cannot be transferred over ROSBridge to any nodes that are subscribing to them. By default, `MMGroundTruth2D`, `MMGroundTruth3D` and the depth camera are not added to `NeedsBridge` list. The script components of these sensors are added to `NeedsBridge` list in order to send messages over ROSBridge.

IV. DATA COLLECTION

A. `lgsvl_data_collector` ROS package

To perform detection of micro-mobility vehicles, we need to collect annotated ground truth data from the various perception sensors available on the ego-car. The data streams that we collected are the following:

1) Main Camera frames
   a) Topic: `/apollo/sensor/camera/traffic/image_short/compressed`
   b) Message type: `CompressedImage`
   c) Image dimensions: 1920×1080
2) Depth Camera frames
   a) Topic: `/simulator/sensor/depth_camera/compressed`

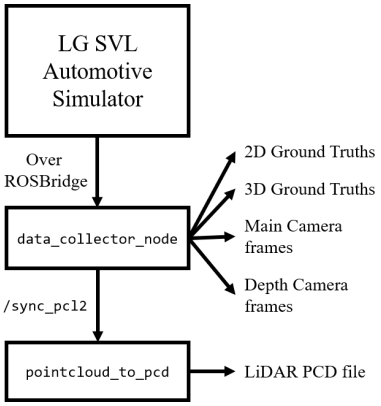

Fig. 4. Using `lgsvl_data_collector` for collecting data from LG SVL simulator over ROSBridge

b) Message type: `CompressedImage`
c) Image dimensions: 1920×1080

3) LiDAR point clouds
   a) Topic: `/apollo/sensor/velodyne64/compensator/PointCloud2`
   b) Message type: `PointCloud2`
   c) Channels: 64
4) Micro-mobility 2D Ground Truth
   a) Topic: `/simulator/ground_truth/mm_2d_detections`
   b) Message type: `Detection2DArray`
5) Micro-mobility 3D Ground Truth
   a) Topic: `/simulator/ground_truth/mm_3d_detections`
   b) Message type: `Detection3DArray`

To maintain acceptable frame-rates, we set the publish rate of these topics at 8 Hz. While the camera images and ground truth messages were saved captured and saved directly by `data_collector_node`, LiDAR point clouds were republished on `sync_pcl2` topic. The `pointcloud_to_pcd` node provided by `pcl_ros` package was run to subscribe to `sync_pcl2` topic and was used to convert the published point clouds to PCD files. Figure 4 shows the graph for this process.

B. Datasets collected

Using the process defined in IV-A, we collected multiple datasets.

1) Small Dataset: We first collected a small dataset to ensure that data collection was working correctly. This dataset contains 1035 data points and was collected with time of day frozen at 11 AM. Collecting this dataset required driving the ego-car manually in the San Francisco Downtown map for less than 10 minutes. Since we did not drive the car through the entire map, the number of e-scooters appearing in this dataset is a lot larger than other vehicles.

2) Large Dataset 1: After ensuring that data collection is working correctly, we drove the ego-car manually for about 1 hour and collected this dataset. This dataset contains 10,042 data points. Time of day was allowed to vary and we added rain, fog and road wetness characteristics in this dataset as well.

3) Large Dataset 2: The process of collecting the large dataset 1 was repeated to collect a second large dataset. This dataset was collected as we needed more LiDAR PCD files for training 3D bounding box detection as mentioned in VI. This dataset contains 10,255 datapoints in different weather conditions and time of day.

V. 2D Detection using YOLOv3

Perception is an important module in autonomous driving because it provides clear visions of one's surroundings. A 2D detection using camera images is a first step to the perception. The 2D detection model should not only be accurate in detecting objects but also be fast enough to run in real time. For this task, we used YOLOv3[16], a real-time object detection architecture based on images.

A. Data Preprocessing

VI. 3D Detection using YOLO3D

The 2D detection using camera images has a great advantage to classify objects due to its utilization of RGB channels. However, it lacks the depth information which is critical for self-driving cars to tell exactly where other objects are located from the rider's perspective. Therefore, LiDAR is used to perceive exact locations of other obstacles. When our group was selecting which architecture to use for 3D object detection, we first had to make sure the model could run in real time. We found out that Baidu's Apollo uses YOLO3D[17] which is based on YOLOv2[26] for their 3D object detection. Also, YOLO3D is solely dependent on point cloud data from the LiDAR; therefore, we decided to use the YOLO3D architecture. For this task, we followed instructions from the paper and built the architecture from scratch since there were no publicly available source codes.

A. Data Preprocessing

YOLO3D utilizes bird-eye-view maps which are the translation of point cloud data to 2-dimensional feature maps where it crops forward and side ranges like the Figure5.

The original paper uses feature maps of 608x608 in dimensions. When they translate the point cloud data to bird-eye-view maps, they only take the side range of 30.4 meters for each side with a forward range of 60.8 meters where the distances are measure from the LiDAR's perspective. Each pixel in the map represents 0.1 meters. For the height range, the original YOLO3D considered from -2 meters to +2 meters from the LiDAR's perspective that will be roughly from the ground to the top of the truck because they were trained on the KITTI dataset [27].

In our 3D object detection model, we set the forward range of 15.2 meters while having the side range of 7.6 meters for each side; therefore, each grid represents 0.025 meters. Since our model only detects those five objects we are interested, we set the height range to cover from -2 meters to 0 meter.

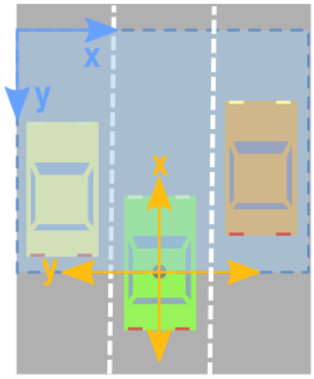

Fig. 5. Bird-eye-view frame rotation and translation

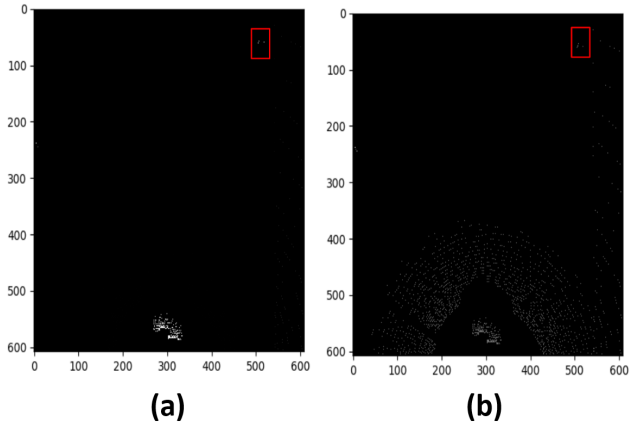

Fig. 6. bird-eye-view maps generated from the point cloud data collected with the LGSVL simulator. (a) height map (b) density map where both figures are in size of $(608, 608)$. The red rectangles are intentionally drawn to show the location of the object's ground truth

The primary reason for this zoomed-in bird-eye-view map is that our targets are much smaller than those cars and trucks. If we kept the same resolution as the paper did, our targets would only be shown as 1 to 2 pixels nearly impossible to differentiate each other.

We have only talked about the grid sizes that are determined by the forward and the side ranges. We also slice height range with a height resolution. We kept this resolution to be 0.03125 meter in order to slice our heights into 64 slices. As a result, our final bird-eye-view conversion would produce maps with dimensions of $(608, 608, 64)$ where the first two dimensions represent (x, y) location from the LiDAR and the last channel represents the height value at the given (x, y) location.

B. Height Map

After converting the point cloud data to bird-eye-view maps, we extract a height map which takes the maximum height at a given (x, y) location. This is simply done by getting the maximum height value out of those 64 slices at a given pixel. Then we store these maximum values to a new $(608, 608)$ map which creates a height map.

C. Density Map

From those $(608, 608, 64)$ bird-eye-view maps, we also need to extract a density map which is based on Eq 1. In the density map, each pixel represents how dense each pixel location is by calculating the sum of all non-empty points in those 64 slices. This also produces another $(608, 608)$ map.

$$min(1.0, \frac{\log(N+1)}{\log(64)}) \tag{1}$$

We then generated a height map and a density map to produce $(608, 608, 2)$, a 2-channel image (Figure 6) that will be used as an input to our YOLO3D architecture.

D. YOLO3D Architecture

As mentioned previously, YOLO3D is based on YOLOv2, a CNN architecture for real-time object detections. However, there needs some modifications on CNN layers, parameters, and a loss function to detect objects in 3D. The predictions from the YOLO3D include 3 more regression outputs.

- yaw angle to tell the orientation of the object
- z center coordinate of the object
- height of the object

These new regression outputs lead us to modify the original loss function from YOLOv2 in order to train the model correctly. A new loss function is the following [include loss function equation]. Note that we included those 3 regression outputs in a new loss function. The YOLO3D paper has a table of a CNN model summary. We initially followed those layers but we were not able to obtain the same output shapes unless we modify some layers. We changed the third max-pooling layer to have a stride of 2 instead of original 1, and removed the last max-pooling layer to prevent the output shape to become too small. As a result, we were able to obtain our output layer to become in shape of $(38, 38, 5, 13)$; the first two dimensions represent number of grid on our input image, the third dimension is number of boxes in each grid, and the last dimension represents regression outputs for each box that has 8 terms $(x, y, z, \text{yaw}, \text{width}, \text{length}, \text{height}, \text{objectness})$ plus the number of classes (5 classes in our project). [include our layers model]

E. Training

Within the given forward and side ranges, we were able to obtain 5503 images. We then divided them - 90% to be the training set and the rest 10% to be the validation set. Currently, our training model is under development due to some issues with batch generations.

VII. TESTING WITH APOLLO

VIII. FUTURE EXTENSIONS

IX. CONCLUSION

X. DELIVERABLES

1) Modified LG SVL Simulator source code:

TABLE I

CNN LAYERS

layer	filters	size	feature maps
conv2d	32	$(3, 3)$	$(608, 608, 2)$
maxpooling	-	(size 2, stride 2)	
conv2d	64	$(3, 3)$	
conv2d	32	$(3, 3)$	
maxpooling	-	(size 2, stride 2)	
conv2d	128	$(3, 3)$	
conv2d	64	$(3, 3)$	
conv2d	128	$(3, 3)$	
maxpooling	-	(size 2, stride 2)	
conv2d	256	$(3, 3)$	
conv2d	128	$(3, 3)$	
conv2d	256	$(3, 3)$	
maxpooling	-	(size 2, stride 2)	
conv2d	512	$(3, 3)$	
conv2d	256	$(1, 1)$	
conv2d	512	$(3, 3)$	
conv2d	256	$(1, 1)$	
conv2d	512	$(3, 3)$	
conv2d	1024	$(3, 3)$	
conv2d	512	$(1, 1)$	
conv2d	1024	$(3, 3)$	
conv2d	512	$(1, 1)$	
conv2d	1024	$(3, 3)$	
conv2d	1024	$(3, 3)$	
conv2d	1024	$(3, 3)$	
conv2d	1024	$(3, 3)$	
conv2d	1024	$(1, 1)$	$(38, 38, 65)$
reshape	-	-	$(38, 38, 5, 13)$

2) Modified LG SVL Simulator binaries:

3) 2D Detection using YOLOv3:

4) 3D Detection using YOLO3D:

5) ROS Packages:

6) Large Dataset 1:

7) Large Dataset 2:

XI. ACKNOWLEDGMENT

We would like to thank Prof. Wencen Wu for her guidance and support throughout the semester and for providing us with an opportunity to work on a project like this through which we were able to gain ample amount of industry knowledge by learning and implementing concepts in deep learning technology.

REFERENCES

[1] [potential impact of self-driving vehicles on household vehicle demand and usage]. [Online]. Available: https://deepblue.lib.umich.edu/handle/2027.42/110789

[2] TechCrunch. [the electric scooter wars of 2018]. [Online]. Available: https://techcrunch.com/2018/12/23/the-electric-scooter-wars-of-2018/

[3] The future of urban transport: Multi-modal system - — ge look ahead — the economist. [Online]. Available: http://gelookahead.economist.com/the-future-of-urban-transport-multi-modal-system/

[4] Here's why multi-modal transport is the future of travel — dds wireless. [Online]. Available: https://ddswireless.com/blog/heres-why-multi-modal-transport-is-the-future-of-travel/

[5] Urban transportations multimodal future. [Online]. Available: https://www.govtech.com/fs/perspectives/Urban-Transportations-Multimodal-Future.html

[6] Traffic congestion: Why its increasing and how to reduce it - livablestreets alliance. [Online]. Available: https://www.livablestreets.info/traffic-congestion-why-its-increasing-and-how-to-reduce-it

[7] The robots have arrived on campus. they come bearing food. — edsurge news. [Online]. Available: https://www.edsurge.com/news/2019-04-19-the-robots-have-arrived-on-campus-they-come-bearing-food

[8] Lime. Lime — electric scooter rentals, micro mobility made simple. [Online]. Available: https://www.li.me/

[9] Bird - enjoy the ride. [Online]. Available: https://www.bird.co/

[10] Boosted boards: The best electric skateboards, longboards scooters. [Online]. Available: https://boostedboards.com/

[11] Onewheel // future motion. [Online]. Available: https://onewheel.com/

[12] R. L. Knoblauch, M. T. Pietrucha, and M. Nitzburg, "Field studies of pedestrian walking speed and start-up time," *Transportation Research Record*, vol. 1538, no. 1, pp. 27–38, 1996. [Online]. Available: https://doi.org/10.1177/0361198196153800104

[13] Boosted board stealth review: speed racer - the verge. [Online]. Available: https://www.theverge.com/2018/7/10/17532568/boosted-board-stealth-electric-skateboard-review-price-specs

[14] [lgsvl/simulator: A ros/ros2 multi-robot simulator for autonomous vehicles]. [Online]. Available: https://github.com/lgsvl/simulator

[15] Apolloauto/apollo: An open autonomous driving platform. [Online]. Available: https://github.com/ApolloAuto/apollo

[16] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[17] W. Ali, S. Abdelkarim, M. Zahran, M. Zidan, and A. El Sallab, "YOLO3D: End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud," *arXiv e-prints*, p. arXiv:1808.02350, Aug 2018.

[18] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[20] U. Technologies. [unity]. [Online]. Available: https://unity.com/

[21] [3d cad design software]. [Online]. Available: https://www.solidworks.com/

[22] [stl (file format) - wikipedia]. [Online]. Available: [https://en.wikipedia.org/wiki/STL_(file_format)]

[23] B. Foundation. [blender.org - home of the blender project - free and open 3d creation software]. [Online]. Available: https://www.blender.org/

[24] [rcp - caucaisan character models - asset store]. [Online]. Available: [https://assetstore.unity.com/packages/3d/characters/humanoids/rcp-caucaisan-character-models-81402reviews]

[25] Personal transportation that simply moves you — segway. [Online]. Available: http://www.segway.com/

[26] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arXiv e-prints*, p. arXiv:1612.08242, Dec 2016.

[27] The kitti vision benchmark suite. [Online]. Available: http://www.cvlibs.net/datasets/kitti/