Detection and Localization of micro-mobility vehicles in a simulated urban environment in the context of Autonomous Driving

Deepak Talwar and Seung Won Lee
Department of Computer Engineering, San José State University
San José, CA
deepak.talwar@sjsu.edu, seungwon.lee@sjsu.edu

***Abstract*— Urban transportation has been rapidly evolving over the past few years; with the introduction of new technology and needs for new use-cases, a new mobility trend has emerged – Micro-Mobility. Specifically, micro-mobility refers to transportation solutions for first-mile and last-mile needs. For self-driving cars to be successful in urban environments, they will need to be able to safely and efficiently navigate and plan paths around these new vehicles that come in numerous shapes, sizes and with varied motion dynamics. This paper presents how perception of these vehicles can be achieved in a simulated environment. Five micro-mobility vehicles are modeled in 3D computer-aided design software and placed in LG SVL Automotive Simulator [1]. The simulator is then configured to allow for data collection from common perception sensors simulated on the ego-car. The collected camera images are annotated and used for training the YOLOv3 [2] object detection model for 2D bounding boxes. The trained model is demonstrated to work well with out-of-sample data and in real-time. For 3D localization of obstacles, a new algorithm based on YOLO3D [3] is implemented in Keras [4] and TensorFlow [5]. YOLO3D [3] uses LiDAR point clouds to detect and create 3D bounding boxes for micro-mobility vehicles. Baidu's Apollo [6] platform for autonomous vehicles is tested with a micro-mobility vehicle inside the simulator and is shown to fail at detecting, localizing and planning a path around the vehicle. Integration of the custom built perception system with Apollo [6] and improvements to prediction and path-planning are left as future extension.**

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 [7], 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 [8]. These shifts are not only changing how people transport themselves, but also show that the future of urban transportation is multi-modal [9], [10], [11] 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 [12]. 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 [13], 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™[14] and Bird™[15], skateboard-esque vehicles from Boosted Board™[16] to One-wheel™[17] 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 [18], whereas, max speed of an electric skateboard is 24 miles per hour [19]. A false detection of a skateboard rider as a pedestrian will lead to 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

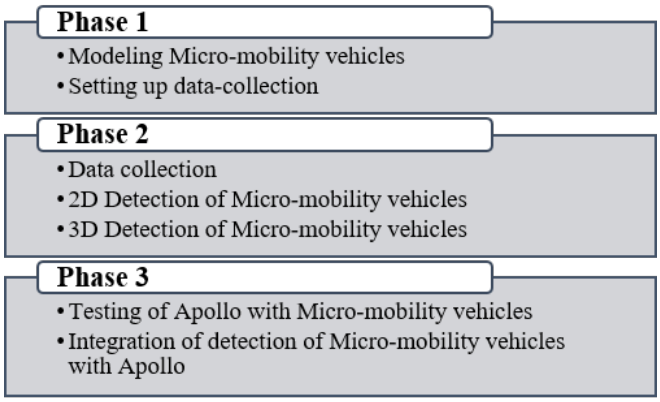

Fig. 1. Project Phases

Fig. 2. LG SVL Automotive Simulator [1]

Simulator [1]. 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 [6] 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 [1] 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 (camera frame) and 3D (LiDAR frame), (iv) fully trained 2D Bounding Box detection model in V based on YOLOv3 [2], (v) implementation of YOLO3D 3D Bounding Box detection model [3] in Keras [4] with TensorFlow [5] backend, and (vi) performance evaluation of Baidu Apollo [6] with micro-mobility vehicles in simulation.

II. PROJECT PHASES

We split this project into three phases. Figure 1 describes the phases and the goals for each phase. Please note that this report describes the procedures and results from phases 1 and 2. We include results from our preliminary testing of the e-scooter model with Baidu Apollo [6] 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) [1]. This is a Unity [20] based simulator that provides out-of-box integration with Baidu's Apollo - open autonomous driving platform [6]. 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™[17]
- Segway Personal Transporter™[21]

Figure 3 shows the final versions (with rider) of all the vehicles. These vehicles were chosen because for their varied appearance and motion models, and their popularity in urban environments. Figure 4 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.
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

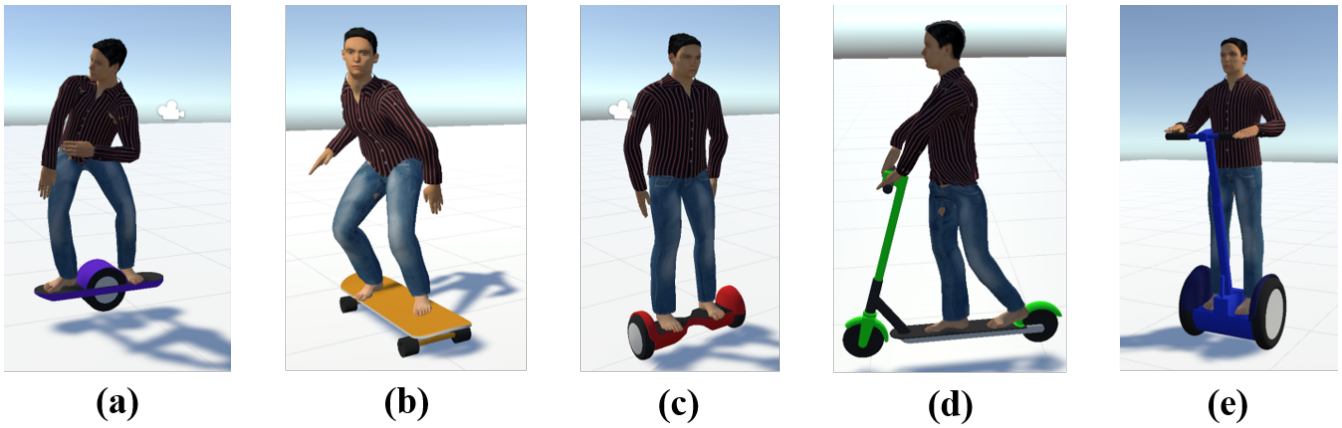

Fig. 3. Final versions of all vehicles. (a) One-wheel™[17], (b) Skateboard, (c) Hoverboard, (d) Electric Scooter and (e) Segway™[21]

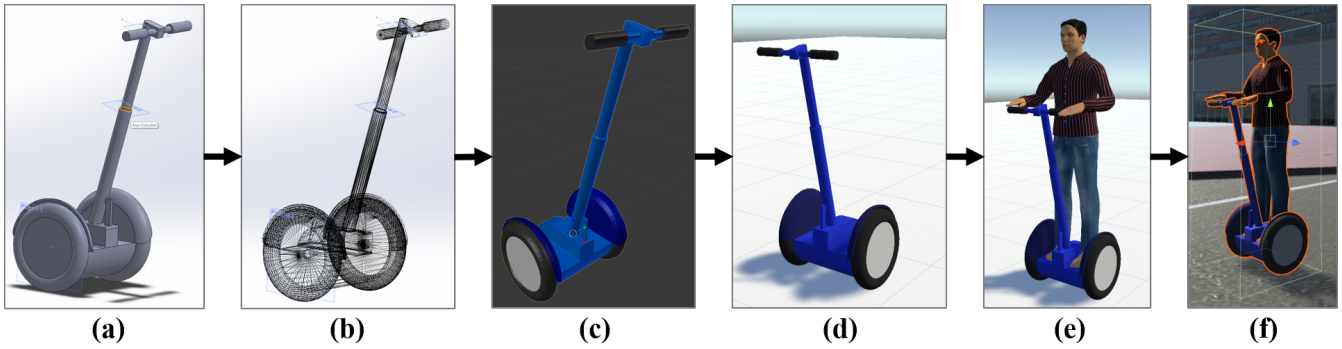

Fig. 4. This figure shows the design process for micro-mobility vehicles created for the project. The first step is to model the vehicle as a solid model in a solid-modeling CAD software. This step is performed in SolidWorks [22] and is shown in (a). The solid model is then converted into a mesh and saved in stereolithography (STL)[23] format as shown in (b). This file is then imported into Blender[24] 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 [25]. 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.

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 Figure 4 (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 Downtown San Francisco 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 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 evident in IX-H.

C. Modifications to Ego-car

LG SVL Automotive Simulator [1] provides an ego-car configured with sensors to work with Apollo [6]. 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`
   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`

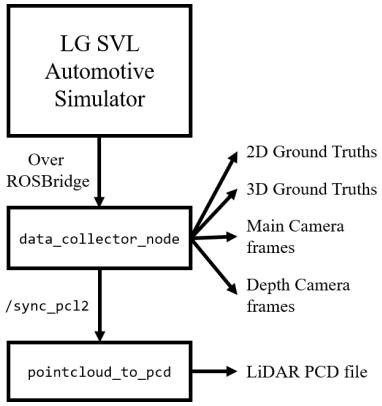

Fig. 5. Using `lgsvl_data_collector` for collecting data from LG SVL simulator over ROSBridge

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 5 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 task in autonomous driving as it provides informs the car of its environment. A 2D detection algorithm using camera images is the 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[2], a real-time object detection architecture based on images.

A. YOLOv3 Architecture

YOLOv3[2] is an improved version of its predecessor, YOLOv2[26]. Although we are not going to cover extensively how this CNN architecture works, YOLOv3[2] is known for its realtively high accuracy and fast inference speed. This architecture uses Darknet-53[27], a neural network framework written by the author of the YOLO series. Fortunately, we were able to find an open source repository[28] that translated the YOLOv3[2] in Keras[4] with TensorFlow[5] backend. Utilizing this repository for our 2D detection was the first step towards training.

B. Training

1) Training 1: After setting up the CNN architecture, we went straight ahead with training. We initially trained with the small dataset that we collected in order to see whether our network could learn as we intended. Our model was able to locate different objects but it failed to classify them correctly; majority of the objects were classified as e-scooters. This was because of class imbalance in the small dataset. There were many more instances of the e-scooter compared to other vehicles. Also, we used the default anchor boxes that were meant to be sized for KITTI[29] dataset like cars and trucks which made the learning ineffective.

2) Training 2: After the first training session, we used Large dataset 1 as introduced in IV-B.2 that was roughly 10 times larger than the first dataset. This time, the anchor box sizes were changed to favor portrait orientations, as all the vehicles we aimed to detect have tall orientations. In our dataset, we also included different weather conditions to have diversity in training images for better generalization. As a result, it showed a promising result that our training was effective. It was able to not only detect objects but also classify them correctly. Video IX-I demonstrates our training results.

C. Results

After having achieved good qualitative results from Training 2, we wanted to evaluate real-time performance of the trained model while driving the ego-car manually around the San Francisco Downtown map. To output the inference results in real-time, we tried two different approaches.

1) Outputting detections with bounding boxes using OpenCV: The video demonstrating this approach is linked in IX-H. We created a ROS Node to subscribe to topic in IV-A.1a and collect main camera frames. This node also initialized the TensorFlow [5] graph and loaded the trained weights for inference. Since the trained model expects the image input to be in `Pillow.Image` [30] format, We used `CvBridge` to convert the `CompressedImage` message first to OpenCV image array, and then to `Pillow.Image` [30]format to send through the TensorFlow model. The inference result was then converted from `Pillow` [30] format back to OpenCV [31] format and displayed in a window continuously. These conversions added lag to the entire process and we saw frame-rates of 15 FPS which is less than ideal. We tried increasing the publish frequency rate of the simulator, however, this resulted in a larger lag between the input and outputs as inferences couldn't keep up.

2) Publishing detections back to the simulator: In an attempt to improve real-time performance, we decided to try a second approach. The video demonstrating this approach is linked in IX-I. LG SVL Simulator [1] provides a template of the ego-car that is configured to work with Autoware [32]. This ego-car subscribes to two additional topics that listen to 2D and 3D detections respectively. Publishing inference results to these topics would allow us to render the predicted bounding boxes directly into the simulator, thus saving us time spent in unnecessary image format conversions. To get this to work, we implemented all of the modifications described in III-C to the Autoware [32] template and imported this vehicle into the scene. We then modified the ROS node described in V-C.1 to publish 2D inferences to topic `/detection/vision_objects` in `DetectedObjectArray` message format. The message definition was also manually created and is linked in IX-E. Although this approach eliminated the various image format conversions, which allowed us to speed it up, most of the gains were lost during the rendering of the bounding boxes inside the simulator. We were able to achieve higher publish frame-rates, however, it also added additional lag in rendering of those frames which resulted in similar performance to the first approach.

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. While selecting the architecture to use for 3D object detection, we needed to make sure that the model could run in real-time. We found out that Baidu's Apollo uses YOLO3D[3] 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. We used [33], an implementation of YOLOv2 [26] in Keras [4] and TensorFlow [5] as reference.

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 Figure 6.

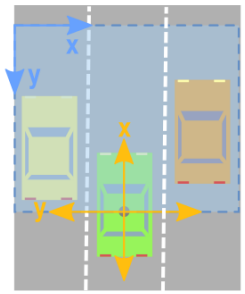

Fig. 6. Frame rotation and translation required for bird's eye view. Yellow axis show the LiDAR frame (Z axis points up). Blue axis show the image frame. Image taken from [34]

The original paper uses feature maps of 608×608 in dimension. 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 $0.1m \times 0.1m$ area. For the height range, the original YOLO3D [3] 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[29] dataset .

In our 3D object detection model, we set the forward range of 15.2 meters while reducing the side ranges to 7.6 meters on each side; therefore, each grid represents $0.025m \times 0.025m$ area. Since our model only detects the five vehicles we are interested in, we also modified the height range to cover from -2 meters to 0 meter. 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 from one another.

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/slice 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 and map it to a value between 0 and 255. Then we store these maximum values to a new $(608, 608)$ map which creates a height map.

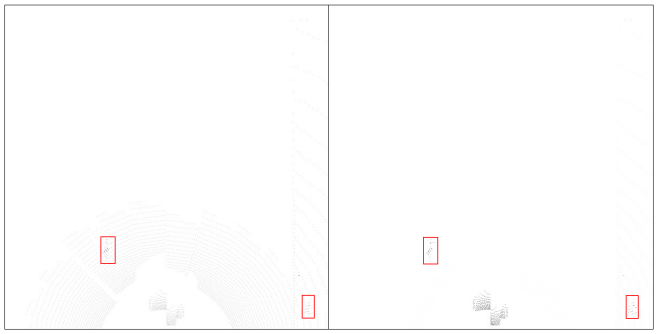

Fig. 7. Bird-eye view maps generated from the point cloud data collected with the LG SVL simulator. height map (left) and density map (right) where both figures are in size of $(608, 608)$. The red rectangles show the locations of the objects' ground truth. The images are intentionally color-inverted for a better display. Note that the actual images used for training are not color-inverted.

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 counting all non-empty slices in those 64 height slices. This value is then mapped to a value between 0 and 255. 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 7) 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 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 written in the Figure 8. The first term in the loss function calculates an error between the predicted $(\hat{x}, \hat{y}, \hat{z})$ and the ground truth (x, y, z) location. The second term calculates an error between the predicted $(\hat{w}, \hat{l}, \hat{h})$ and the ground truth (w, l, h) – width, length, and height. The third term represents an error between the predicted $(\hat{\phi})$ and the ground truth (ϕ) – yaw angle. The following two terms represent an error between

$$
\begin{aligned}
L = \lambda_{coor} & \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2 + (z_i - \hat{z_i})^2] \\
& + \lambda_{coor} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w_i}})^2 + (\sqrt{l_i} - \sqrt{\hat{l_i}})^2 \\
& + (\sqrt{h_i} - \sqrt{\hat{h_i}})^2] \\
& + \lambda_{yaw} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} (\phi_i - \hat{\phi_i})^2 \\
& + \lambda_{conf} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} (C_i - \hat{C_i})^2 \\
& + \lambda_{conf} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{noobj} (C_i - \hat{C_i})^2 \\
& + \lambda_{classes} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p_i}(c))^2
\end{aligned}
$$

Fig. 8. Loss function for YOLO3D[3]

the predicted $(\hat{C})$ and the ground truth (C) – confidence that an object is present at that location (also known as objectness). Finally the last term represent an error between the predicted $(\hat{p}(c))$ and the ground truth $(p(c))$ – class probabilities. λ values are the weights for each loss and L^{obj} and L^{noobj} are the variables that take either 0 or 1 depending on the presence of an object.

The YOLO3D[3] 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). Table I shows the modified architecture of YOLO3D[3].

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

In order to evaluate how well Apollo 3.0 [6] deals with micro-mobility vehicles out of the box, we connected the modified LG SVL Automotive Simulator [1] with Apollo 3.0 [6] over ROS Bridge. The video of this test is shown in IX-J. An electric scooter is placed in the simulator and is manually controlled with the keyboard by the user. The ego-car is configured to be controlled by Apollo [6]. We then provide the car with a routing request to go straight and move the e-scooter to interfere in the ego-car's path.

As evident from the video, in the first attempt, Apollo's [6] perception module fails to correctly perceive the e-scooter and collides with it. It is only able to perceive it in a fraction of the frames. It first detects the vehicle as "Unknown" (denoted by magenta colored bounding box) and as it gets closer to the vehicle, it detects the vehicle as a moving pedestrian with the path predicted changing rapidly.

In the second attempt, Apollo's [6] perception module does detect the vehicle as "Unknown" object and slows the car down, however, it loses the bounding box detection as it gets closer to the vehicle and inevitably collides with it and tips it over.

This test was performed on a single computer running Intel Core i9-9900K [35], 32 GB RAM and NVIDIA GTX 1080 Ti [36].

TABLE I
CNN ARCHITECTURE FOR YOLO3D[3]

layer	filters	size	feature maps
conv2d	32	$(3, 3)$	$(608, 608, 2)$
maxpooling	-	(size 2, stride 2)	
conv2d	64	$(3, 3)$	
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	-	-	$(38, 38, 5, 13)$

VIII. FUTURE IMPROVEMENTS AND EXTENSIONS

This project only accomplishes the first-step of teaching self-driving cars to interact with micro-mobility vehicles – perception. There are many improvements and extensions that can be added.

A. Improvements

- Improving frame-rates of 2D detections by optimizing image data conversions.
- Better generalization of detections by collecting more data in cities other than San Francisco.
- Better generalization by using more realistic materials/finishes on the models.
- Adding different kinds of human riders on vehicles.
- Collect pedestrian data and add them into training as well.
- Using reflectance information from LiDAR point clouds.

B. Extensions

- Automating movement of micro-mobility vehicles to collect trajectory information.
- Training recurrent neural networks to predict path of these vehicles after successful detection.
- Incorporating perception and prediction modules into Apollo [6] to improve path planning.

IX. DELIVERABLES

A. Modified LG SVL Simulator source code (GitHub repository)

https://github.com/deepaktalwardt/simulator

B. Modified LG SVL Simulator binaries

https://www.dropbox.com/sh/xvqvayy46ehagmn/AAC16z0FEG7Vtl2YylLQ83VAa?dl=0

C. 2D Detection using YOLOv3 (GitHub repository)

https://github.com/deepaktalwardt/keras-yolo3

D. 3D Detection using YOLO3D (GitHub repository)

https://github.com/swdev1202/keras-yolo3d

E. ROS Packages (GitHub repository)

https://github.com/deepaktalwardt/cmpe297_ros_pkgs

F. Large Dataset 1

https://www.dropbox.com/s/9cvsmraio6q6v0d/large_dataset_1.zip?dl=0

G. Large Dataset 2

https://www.dropbox.com/s/kt6hwfsa95v4hck/large_dataset_2.zip?dl=0

H. 2D Obstacle inference with OpenCV (YouTube Video)

https://youtu.be/DwWY89dVGEw

I. 2D Obstacle inference in LG SVL Simulator (YouTube Video)

https://youtu.be/72CPQL3bGWQ

J. Testing Apollo with E-scooter (YouTube Video)

https://youtu.be/TVreirGAxmI

K. Project presentation slides

https://docs.google.com/presentation/d/1NzCOh9w1M_gmNOr4F7BRGH3UlDinqTB6wfFwwc2YfJY/edit?usp=sharing

X. CONCLUSION

We have introduced micro-mobility vehicles in a simulated world using LG SVL Automative Simulator [1] to study autonomous vehicles' interactions with them. At this point, the self-driving car is only able to perceive where the objects are located using its camera. However, we will continue to explore by implementing future extensions. We ultimately want our findings and studies to be applied in real world applications where all autonomous cars could not only perceive micro-mobility vehicles but also predict their paths to navigate safely in those surroundings.

XI. ACKNOWLEDGMENTS

We would like to thank Prof. Wencen Wu and Prof. Kaikai Liu for 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 self-driving technology.

We would also like to extend special thanks to Martins Mozeiko and Brian Shin from LG Silicon Valley Lab for inviting us to their office and demonstrating all the features of their amazing simulator. We wouldn't have been able to get this far with the project without Martins' detailed email responses to our very specific questions. We wish to continue working with LG SVL and their simulator as we extend this project.

REFERENCES

[1] lgsvl/simulator: A ROS/ROS2 Multi-robot Simulator for Autonomous Vehicles. [Online]. Available: https://github.com/lgsvl/simulator

[2] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[3] 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.

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

[5] 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/

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

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

[8] M. R. Dickey. The electric scooter wars of 2018. [Online]. Available: https://techcrunch.com/2018/12/23/the-electric-scooter-wars-of-2018/

[9] 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/

[10] 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/

[11] Urban Transportation's Multimodal Future. [Online]. Available: https://www.govtech.com/fs/perspectives/Urban-Transportations-Multimodal-Future.html

[12] Traffic Congestion: Why It's Increasing and How To Reduce It — LivableStreets Alliance. [Online]. Available: https://www.livablestreets.info/traffic-congestion-why-its-increasing-and-how-to-reduce-it

[13] 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

[14] Lime - Electric Scooter Rentals, Micro Mobility Made Simple. [Online]. Available: https://www.li.me/

[15] Bird - Enjoy the Ride. [Online]. Available: https://www.bird.co/

[16] Boosted Boards: The Best Electric Skateboards, Longboards Scooters. [Online]. Available: https://boostedboards.com/

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

[18] 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

[19] 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

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

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

[22] 3D CAD Design Software. [Online]. Available: https://www.solidworks.com/

[23] STL (file format) - Wikipedia. [Online]. Available: [https://en.wikipedia.org/wiki/STL_(file_format)]

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

[25] RCP - Caucaisan Character Models - Asset Store. [Online]. Available: https://assetstore.unity.com/packages/3d/characters/humanoids/rcp-caucaisan-character-models-81402#reviews

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

[27] pjreddie/darknet: Convolutional neural networks. [Online]. Available: https://github.com/pjreddie/darknet

[28] qqwweee/keras-yolo3: A keras implementation of yolov3 (tensorflow backend). [Online]. Available: https://github.com/qqwweee/keras-yolo3

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

[30] Image Module — Pillow (PIL Fork) 6.0.0 documentation. [Online]. Available: https://pillow.readthedocs.io/en/stable/reference/Image.html

[31] Opencv. [Online]. Available: https://opencv.org/

[32] autowarefoundation/autoware: Open-source software for self-driving vehicles. [Online]. Available: https://github.com/autowarefoundation/autoware

[33] experiencor/keras-yolo2: Easy training on custom dataset. various backends (mobilenet and squeezenet) supported. a yolo demo to detect raccoon run entirely in brower is accessible at https://git.io/vf7vi (not on windows). [Online]. Available: https://github.com/experiencor/keras-yolo2

[34] Ronny Restrepo. [Online]. Available: http://ronny.rest/tutorials/module/pointclouds_01/point_cloud_birdseye/

[35] Intel® Core™ i9-9900K Processor (16M Cache, up to 5.00 GHz) 186605. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/core/i9-processors/i9-9900k.html

[36] GeForce GTX 1080 Ti Graphics Cards — NVIDIA GeForce. [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/