Realtime map building, vehicle localization and track following, and object detection based on LGSVL, Autoware and LogiTech G29

Zhi Fang Tan, Lingyi Song, Kexuan Zhai
Department of Automotive Engineering, Clemson University
International Center for Automotive Research (CU-ICAR)
2019 Fall Course Report: AuE 8930 HPC for Autonomy, Instructor: Dr. Bing Li

Abstract

To ensure safety, autonomous system can be tested in simulated environment before being deployed into real world condition. In this project, we implement simple autonomous system on Autoware platform using LGSVL simulator with LogiTech G29 Steering Wheel. Sensor data such as LIDAR point cloud, camera image, etc. are acquired from LGSVL simulator and input into Autoware to achieve mapping, detection, localization, and motion planning.

Introduction

1.1 Overview

In this project, the mapping is first performed to produce a point cloud map. Next, localization is then done using NDT matching to acquire the pose of the vehicle through matching point clouds from LIDAR sensor. With localization done, a preset path is created by driving the car manually using G29 steering wheel. Following that, the car will follow the path using pure pursuit algorithm.

1.2 Experimental Facilities

LGSVL and Autoware were utilized as the software part of realtime autonomous system. LGSVL is being run on Windows operating system to simulate real-drive circumstances while Autoware runs on the Linux operating system to achieve the function of data reading, data processing and control. LGSVL and Autoware communicate and exchange data through LAN. The figure below shows the communication between LGSVL and Autoware.

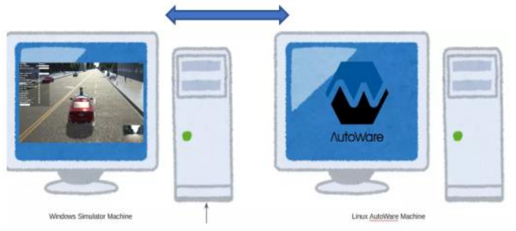


Figure 1: The Communication Between LGSVL and Autoware.

Figure 2: The demo in LGSVL

The packages installation includes CUDA 10, cuDNN, CUDA toolkit. CUDA is obtained for Autoware installation and Deep leaning. cuDNN and CUDA toolkit are obtained for YOLO.

Logitech G29 is obtained as a steering wheel controller. GTX 1060 is installed on the desktop with Windows 10 operating system. GTX 1050 is installed on the Laptop with Linux Ubuntu operating system.

1.3 Project Objective

- Implementation of a virtual environment for autonomous drive testing in LGSVL
- Implementation of the communication between Logitech G29 and LGSVL
- Communication between LGSVL and Autoware
- Point cloud mapping for a virtual map in LGSVL
- Automatic path following simulation
- Real-time object detection using YOLO3 in a virtual environment

Autoware

2.1 Autoware.AI

Autoware.AI is the world's first "All-in-One" open-source software for autonomous driving technology. It is based on ROS 1 and available under the Apache 2.0 license. It contains the following modules:

- **Localization** is achieved by 3D maps and SLAM algorithms in combination with GNSS and IMU sensors.
- **Detection** uses cameras and LiDARs with sensor fusion algorithms and deep neural networks.
- **Prediction** and **Planning** are based on probabilistic robotics and rule-based systems, partly using deep neural networks as well.
- The output of Autoware to the vehicle is a twist of velocity and angular velocity. This is a part of **Control**, though the major part of the Control stack commonly resides in the by-wire controller of the vehicle.

The above four modules cover almost the entire current technical process of autonomous driving (sensing layer --- prediction and planning layer --- control layer) and are of great reference value for the study of technical routes. Most importantly: its code is completely open-sourced.

2.2 Autoware.Auto

Autoware.Auto, another project from the Autoware Foundation, is a clean slate rewrite of Autoware.AI based on ROS 2. Compared to Autoware.AI, Autoware.The auto has the best possible software engineering practices which include PR reviews, PR builds through CI, 100% documentation, 100% code coverage, style guide, development, and release process.
Autoware.The auto has two other major differentiators when it's compared to Autoware.AI:

- Crisply defined interfaces for different modules (messages and APIs)
- An architecture designed for determinism, such that it is possible to reproduce behaviors on live machines and development machines

The figure below shows the function of LGSVL and Autoware that used in the project. LGSVL was used as a simulator, which runs on the Windows operating system. All the algorithm was applied in Autoware for vehicle autonomy. For instance, computing, data analyzing, and vehicle controlling.

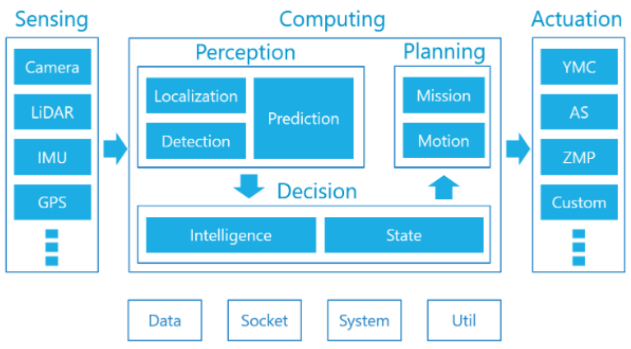


Figure 3: The Algorithm Applied in Autoware

2.3 Use Cases

Autoware.Auto initially targets the following 2 use cases:

- Autonomous Valet Parking
- Autonomous Depot Maneuvering

2.4 Roadmap

Functional features in Autoware. The auto was developed according to the roadmap below.

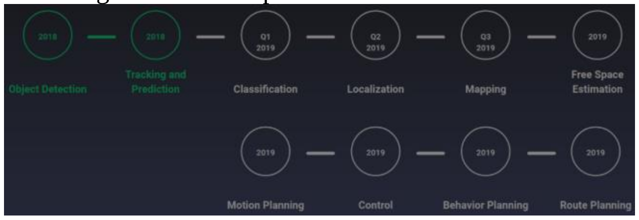


Figure 4: The Autoware Roadmap

LGSVL Simulator

3.1 LGSVL Introduction

The LGSVL Simulator is a simulator that facilitates the testing and development of autonomous driving software systems. It enables developers to simulate billions of miles and arbitrary edge case scenarios to speed up algorithm development and system integration.

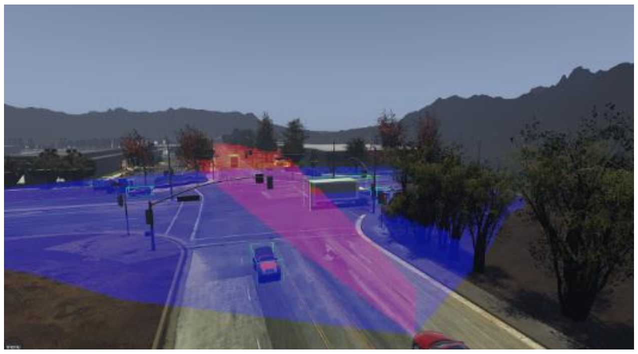


Figure 5: LGSVL Simulator

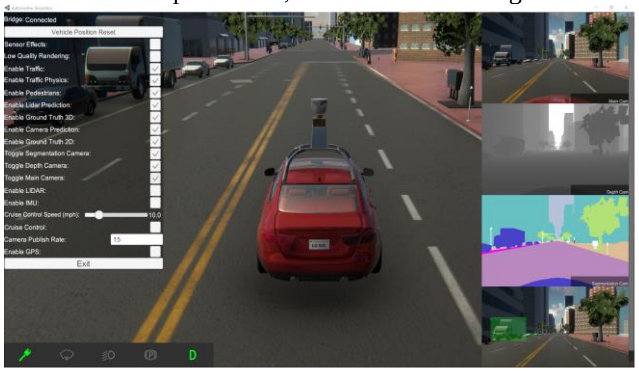
The simulator is fully integrated with the popular open-source platforms Apollo (from Baidu) and Autoware (Autoware Foundation). This means that a developer working with either of these platforms can simply download our latest binaries and run the simulator to test their algorithms, whether they are for localization, perception, or path planning.

By running an autonomous vehicle, or modules of the autonomous stack, in a safe and reproducible yet realistic 3D environment, developers are able to run tests and iterate much more quickly on their algorithms. Our simulator provides real-time outputs from sensors including camera, LiDAR, RADAR, GPS, and IMU. Environmental parameters can also be changed, including a map, weather, traffic, and pedestrians.

The LGSVL simulator is developed by the Advanced Platform Lab at the LG Electronics America R&D Center, formerly the LG Silicon Valley Lab.

3.2 LGSVL Implementation

The virtual environment of a for autonomous testing in LGSVL was implemented, as is shown in the figure below.



Bridge: Connected
Vehicle Position Reset
Sensor Effects: ☐
Low Quality Rendering: ☐
Enable Traffic: ✓
Enable Traffic Physics: ✓
Enable Pedestrians: ✓
Enable Lidar Prediction: ✓
Enable Ground Truth 3D: ✓
Enable Camera Prediction: ✓
Enable Ground Truth 2D: ✓
Toggle Segmentation Camera: ✓
Toggle Depth Camera: ✓
Toggle Main Camera: ✓
Enable LIDAR: ☐
Enable IMU: ☐
Cruise Control Speed (mph): 10.0
Cruise Control: ☐
Camera Publish Rate: 15
Enable GPS: ☐
Exit

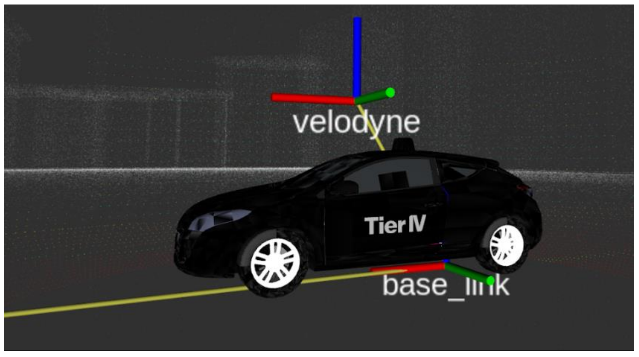
Figure 6: LGSVL Implementation

Mapping

4.1 Coordinate Systems

Four coordinate systems were obtained in Autoware. Including World Coordinate System, Map Coordinate System, Base Link (vehicle coordinate system) and Velodyne (sensor coordinate system).

Different coordinates are connected through coordinate transformation. The coordinate system transformation from World to map and the coordinate system transformation from base_link to Velodyne are fixed. This function can be achieved using ROS 'TF'. The mapping from Map to base_link requires a scan-to-map algorithm. Autoware uses NDT (Normal Distribution Transform) matching in the construction.



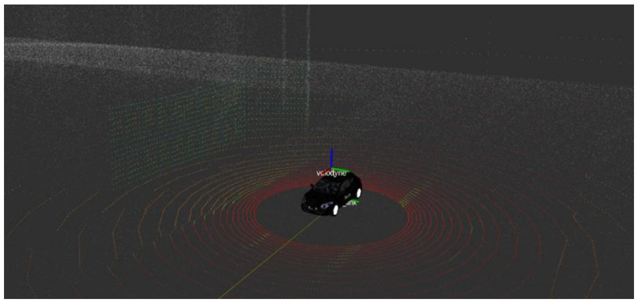


Figure 7: Coordinate Systems

The map that was used for the experiment is shown in the figure below. The picture on the left shows the vector map. The picture on the right shows the screenshot taken from LGSVL.

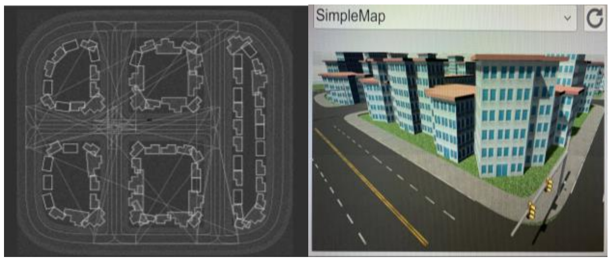


Figure 8: Experimental Map

4.2 Normal Distributions Transform

The Normal Distribution Transform (NDT) algorithm is a registration algorithm that is applied to a statistical model of three-dimensional points. It uses standard optimization techniques to determine the best match between two-point clouds because it does not make use of the registration process. Corresponding point features are calculated and matched, so time is faster than other methods.

The basic idea of the NDT algorithm is to first construct a normal distribution of multi-dimensional variables based on reference data. If the transformation parameters can make the two-laser data match well, the probability density of the transformation points in the reference system will be very high and Big. Therefore, it can be considered to use an optimized method to obtain the transformation parameter that maximizes the sum of the probability densities. At this time, the two laser point cloud data will match best.

The basic steps of the algorithm are as follows:

- Divide the space occupied by the reference point cloud into a grid or voxel of a specified size (CellSize), and calculate the multidimensional normal distribution parameters of each grid:

Mean: $q = \frac{1}{n}\sum_i x_i$

Covariance matrix: $\sum = \frac{1}{n}\sum_i (x_i - q)(x_i - q)^T$

- Initialize transformation parameters $p = (t_x, t_y, \phi)^T$
- For the point cloud to be registered (second scan), transform it into the grid of the reference point cloud by transforming *T*.

$$T : \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

- Calculate the probability density of each transition point based on the normal distribution parameters.

$$p(x) \sim \exp\left(\frac{(x-q)^T \sum^{-1} (x-q)}{2} \right)$$

- NDT registration score (score) is obtained by adding the probability density calculated for each grid

$$score(p) = \sum_i \exp\left(\frac{(x_i' - q_i)^T}{2} \right)$$

- Score the objective function according to Newton's optimization algorithm. Perform optimization, that is, find the transformation parameter p to maximize the value of the score. The key steps of the optimization are to calculate the gradient and Hessian matrix of the objective function:

$$s = -\exp\left(\frac{-q^T \sum^{-1} q}{2} \right)$$

- According to the chain derivation rule and the formula for vector and matrix differentiation, the direction of s gradient is

$$\frac{\partial s}{\partial pi} = \frac{\partial s}{\partial q}\frac{\partial q}{\partial pi} = q^T \sum^{-1} \frac{\partial q}{\partial pi} \exp\left(\frac{-q^T \sum^{-1} q}{} \right)$$

- Jacobian matrix：

$$\frac{\partial q}{\partial pi} = J_T = \begin{pmatrix} 1 & 0 & -x\sin\phi & -y\cos\phi \\ 0 & 1 & x\cos\phi & -y\sin\phi \end{pmatrix}$$

- According to the transformation equation, the vector of the second derivative of the vector q to the transformation parameter p is:

$$\frac{\partial^2 q}{\partial p_i \partial p_j} = \begin{Bmatrix} \begin{pmatrix} -x\cos\phi & +y\sin\phi \\ -x\sin\phi & -y\cos\phi \end{pmatrix} & i = j = 3 \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & otheruise \end{Bmatrix}$$

- Jump to step 3 and continue execution until the convergence condition is reached.

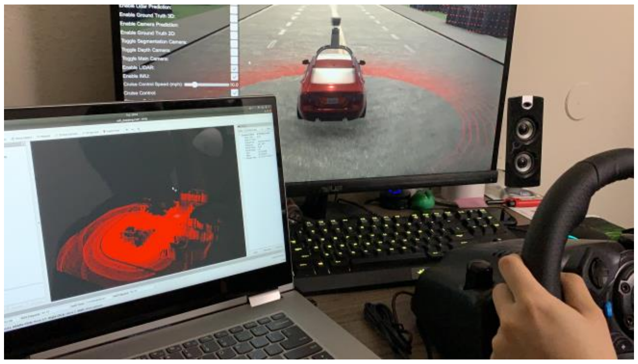


Figure 9: Point Cloud Map Through Lidar Data Acquired From a Virtual Map

Localization

. Using lidar data to matching the points cloud map.

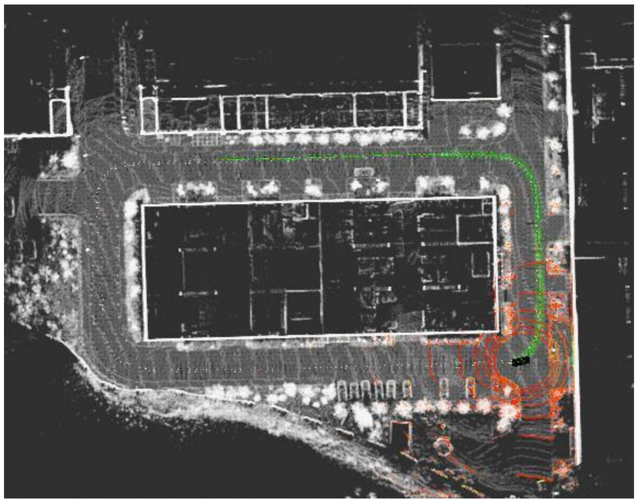


Figure 10: Localization in the Points-cloud Map Though Lidar.

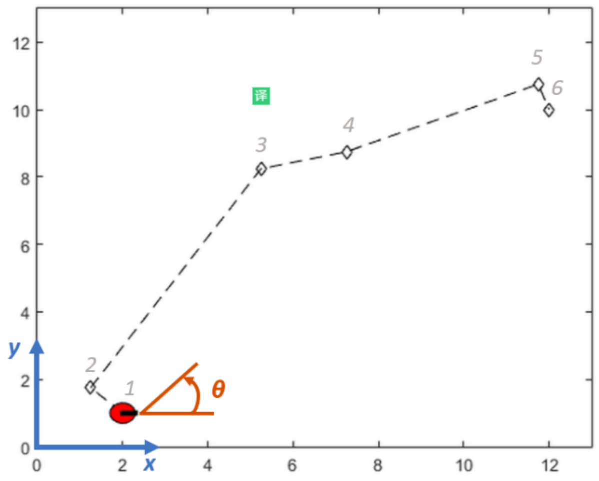


Figure 11: Pure Pursuit Algorithm

Path Following

5.1 Pure Pursuit algorithm

Pure pursuit is a path tracking algorithm. It computes the angular velocity command that moves the robot from its current position to reach some look-ahead point in front of the robot. The linear velocity is assumed constant, hence you can change the linear velocity of the robot at any point. The algorithm then moves the look-ahead point on the path based on the current position of the robot until the last point of the path. You can think of this as the robot constantly chasing a point in front of it. The property LookAheadDistance decides how far the look-ahead point is placed.

The controllerPurePursuit object is not a traditional controller but acts as a tracking algorithm for path following purposes. Your controller is unique to a specified list of waypoints. The desired linear and maximum angular velocities can be specified. These properties are determined based on vehicle specifications. Given the pose (position and orientation) of the vehicle as an input, the object can be used to calculate the linear and angular velocities command for the robot. How the robot uses these commands is dependent on the system you are using, so consider how robots can execute a motion given these commands. The final important property is the LookAheadDistance, which tells the robot how far along on the path to track towards.

5.1.1 Look Ahead Distance

The LookAheadDistance property is the main tuning property for the controller. The look-ahead distance is how far along the path the robot should look from the current location to compute the angular velocity commands. The figure below shows the robot and the look-ahead point. As displayed in this image, note that the actual path does not match the direct line between waypoints.

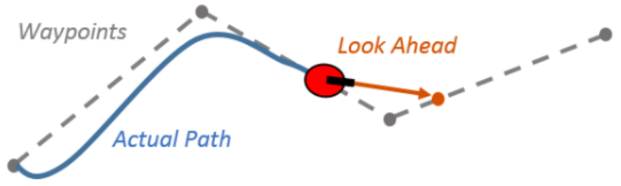


Figure 12: Pure Pursuit Algorithm

The effect of changing this parameter can change how your robot tracks the path and there are two major goals: regaining the path and maintaining the path. In order to quickly regain the path between waypoints, a small LookAheadDistance will cause your robot to move quickly towards the path. However, as can be seen in the figure below, the robot overshoots the path and oscillates along the desired path. In order to reduce the oscillations along the path, a larger look-ahead distance can be chosen, however, it might result in larger curvatures near the corners.

5.1.2 Pure Pursuit path following

The vehicle will automatically follow the path we recorded before. The path is recorded as .cvs file, with the x, y, z coordinates, the yaw rate and the speed of the car. The limitation of lateral acceleration can be set as demanded. The GPS is added as the supplement for localization. The procedure of pure pursuit path is shown below:

- Set coordinate systems transform
- Load the map
- Activate localization
- Set filtering and limitations
- Run the simulation data
- Monitoring the driving process

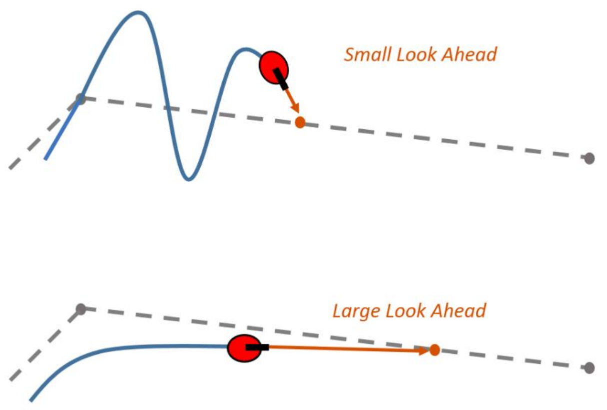


Figure 13: Pure Pursuit Algorithm

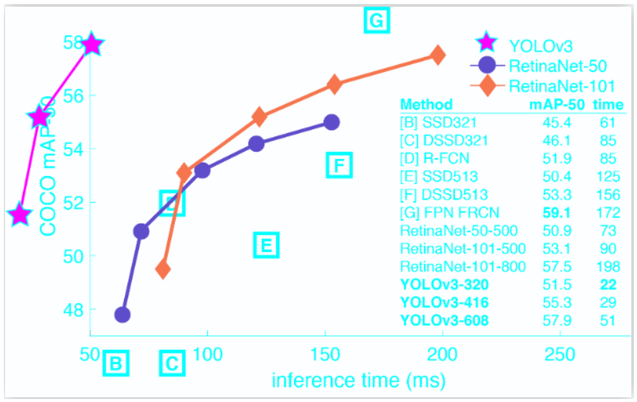
5.2 Pure Pursuit Implementation

Pure pursuit algorithm is used for path following
The bicycle model is considered in the algorithm as the dynamic model for vehicle control, and through calculation, will give the planning for following the path, and command the vehicle to follow the path.

Figure 14: Pure Pursuit Path Following

Object Detection

To implement objective detection, YOLO is utilized. YOLO is a deep learning detection algorithm based on CNN. YOLO stands for You Only Look Once: Unified, Real-Time Object Detection. It is a target detection algorithm proposed in CVPR2016. Her core is to transform target detection into regression problem-solving. And based on a separate end-to-end network, it completes the input from the original image to the output of the object position and category. Camera image was sent to Autoware, where Autoware was running YOLO. Therefore, the detection results can be got from the output.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 15: Comparation of Different Algorithm

There were two demonstrations displayed. The first demonstration is for the traffic lights, as shown in the figure below. The second demonstration is for object detection. Because of the computation capability of our hardware is limited, we can just set the publish rate of the camera as 10. The implementation of object detection will be better if it can run with a more powerful platform.

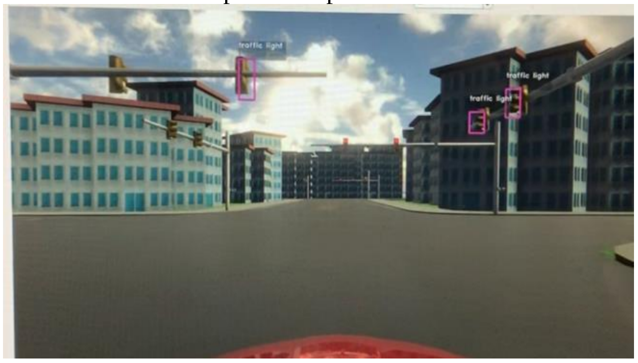


Figure 16: The Demonstration for Traffic Light

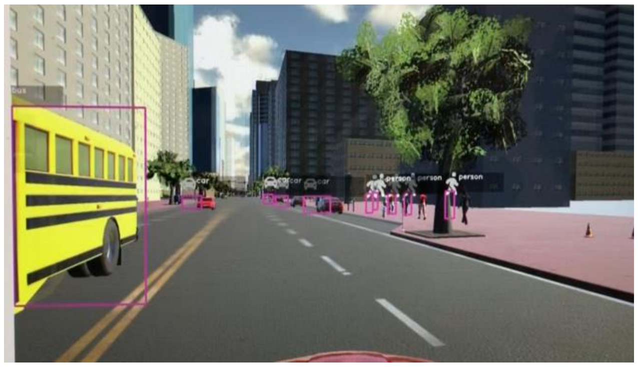


Figure 17: The Demonstration for Object Detection

Conclusion

Through this project, it is clear that Autoware has great potential as an autonomous platform. For future work, we would like to implement obstacle avoidance and motion planning which is able to obey the traffic rules. We would also like to implement actual sensor and do a real world mapping with Autoware in future if chance is given.