Software-in-the-loop Modeling and Simulation Framework for Autonomous Vehicles

Mohamed Fasil Syed Ahamed
Mechanical Engineering
Kettering University
Flint, USA
syed0216@kettering.edu

Girma Tewolde
Electrical and Computer Engineering
Kettering University
Flint, USA
gtewolde@kettering.edu

Jaerock Kwon
Electrical and Computer Engineering
Kettering University
Flint, USA
jkwon@kettering.edu

***Abstract*—In the process of development of autonomous vehicles, software-in-the-loop (SIL) modeling and simulation has become an inevitable part of testing. In order to support the increased number of research on SIL modeling and simulation, in this paper we have created a vehicle framework by which anyone can build their own vehicle model with ease. This paper focuses on the SIL modeling and simulation which was developed in Gazebo using Robotic Operating System (ROS). The goal of the framework is to serve as a platform to create multiple vehicle models and help users to validate and compare the different algorithms for various models. This paper also explains in detail the methodology to create a vehicle model and a custom environment to show the validity of the proposed framework. The experimental results show the successful implementation of the framework in a custom environment.**

***Keywords*—Software-in-the-loop Simulation, ROS, Gazebo, Vehicle framework, Autonomous vehicles**

I. INTRODUCTION

A recent global automotive executive survey by KPMG, a professional service company, reveals that autonomous and self-driving vehicles have seen a steady increase among all the other key trends in the auto industry [1]. In accordance to the survey all the leading automotive global OEMs along with other big companies like Google and Uber have begun working on their own autonomous vehicles. The key prospect in the development of autonomous vehicles is testing and that is where software-in-the-loop (SIL) modeling and simulation comes into place as it supports safe and cost-effective testing. In order to test preliminary algorithms developed, simulation is used instead of testing the algorithms directly on the vehicle. Once the simulation results are positive, the algorithms are tested on vehicles. Development lead time also reduces significantly by the usage of SIL modeling and simulation as different environments required for validating the vehicle can be created easily in simulation rather than creating them in the actual world.

Robot Operating System (ROS) [2] is a collection of tools and libraries which helps simplifying the task of building a robot system. Gazebo [3] is a simulation tool which helps to simulate robots in various environments. The advantage of using Gazebo is that it has a topic-based Publish/Subscribe model of communication which goes hand in hand with ROS and makes it easy to create models [4]. The Gazebo Plugins help to build a direct communication interface with ROS thereby being competent simulation tool for developing and testing robots.

Having said the advantages of simulation, we designed and evaluated an SIL modeling and simulation framework, named MIR (Mobile Intelligent Robotics) vehicle that allows anyone to build a vehicle model in Gazebo using ROS. The evaluation proved that the proposed framework can be a platform of SIL modeling and simulation for autonomous vehicles. This paper also explains the methodology to build a vehicle model inside a test track environment in Gazebo and discusses the problems faced along with the solutions.

II. RELATED WORK

There are various simulation environments for vehicle models in Gazebo. Most of them are open source but some of them are commercial. This section talks in detail about the related SIL modeling and simulation done in Gazebo using ROS for vehicle models.

One of the promising efforts was the Cognitive and Autonomous Test (CAT) vehicle [5] that is an open source software-in-the-loop modeling and simulation done in Gazebo using ROS. The CAT vehicle has been used for several years for Research Experiences for Undergraduates (REU). It uses a Ford Explorer vehicle model with sensors such as cameras, LiDARs, and RADARs and multiple virtual environments where the vehicle can be tested using ROS along with Gazebo. The CAT vehicle was designed for working with a single particular vehicle, Ford Explorer. So, it is not easy for users to use for other vehicle models. The entire model focuses on how to launch the vehicle in different worlds and supports hardware in the loop (HIL) simulation [27].

The *dbw_mkz_simulation* is part of the ADAS development kit, *dbw_mkz*, created by Dataspeed Inc [6]. The dbw came from Drive By Wire (DBW) meaning electrical or electro-mechanical systems are used to perform vehicle functions that were achieved by mechanical linkages. The *dbw_mkz_simulation* can be considered as an example of commercial SIL modeling and simulation. It predominantly focuses on how to visualize and simulate vehicle models in virtual environments. The *dbw_mkz_simulation* package is to help users of *dbw_mkz* that is a commercial ADAS development kit in which algorithms can be tested in simulated environments. There are examples such as lane keeping in a simulated road and multi-cars are operating in an environment that would have required much resources in a real-world environment. Users cannot use their own car models to test algorithms in the simulated environment since some core modules are not provided as a form that users can modify.

We proposed a framework, *MIR vehicle* whose goal is to serve as a platform to create multiple vehicle models and help users to test and validate algorithms before deploying them to real vehicles. The *MIR vehicle* allows users to conduct autonomous vehicle related research such as stop sign detection, vehicle's longitudinal and lateral controls in a wide variety of car models and environments with ease.

In the following sections, we will explain in detail how to design and implement the *MIR vehicle* and how to use it. Then some examples will be shown to validate the proposed framework.

III. Framework For Vehicle Model

We designed a framework named as *MIR vehicle* for the SIL modeling and simulation. The *MIR vehicle* framework provides a complete set of tools in which one can add vehicle models and environments. Further topics below explain the framework design and the changes for different vehicle platforms.

A. Framework Design

A ROS workspace by the name of `mirvehicle_ws` was created and the packages were created in accordance with the latest ROS convention.

1. mirvehicle_description
   This package contains all the files required to create the vehicle model. It has the DAE [8], URDF [9] and launch files to visualize the vehicle in R-viz. The DAE files along with the textures are stored under `meshes` folder. Inside the `URDF` folder, folders are categorized according to the vehicle names which contain the XACRO [10] files of the vehicle. Finally, the launch file for visualizing the vehicle in R-viz is saved under the `launch` folder.

2. mirvehicle_gazebo
   This package contains the files required to create an environment in Gazebo. It has the DAE, SDF [11], WORLD [12], YAML [13] and launch files to launch the vehicle in the respective environment. The `Models` folder contains DAE, texture, SDF and config file for the world. The `world` folder contains the different custom worlds created like the test track, skidpad etc. The `config` folder contains the YAML file which includes ROS based controllers for drive and steering the vehicle in Gazebo [14]. Finally, all the launch files for getting the vehicle in different worlds are located in the `launch` folder.

3. mirvehicle
   This package predominantly contains further works that can be made to the vehicle model. The `scripts` folder contains ROS based python [15] scripts to convert the input from the joystick to *cmd_vel*, from *cmd_vel* to gazebo, etc. The `include` folder contains all the header files and `src` folder contains the C++ source codes. The `launch` folder has the ROS launch files to launch the additional work.

4. All other packages
   Any other specific package required for the vehicle like obstacle stopper [16] (to avoid obstacles) and `sicktoolbox_wrapper` [17] (ROS wrapper for the outstanding `sicktoolbox` library for interfacing with the SICK LMS2XX lasers) are also located under the `src` folder.

B. Changes in Framework for New Vehicle Model

In order to get the user vehicle model to work with this framework in a predefined world, the below steps need to be followed.

1. The user should create a folder by the name of `your vehicle` under `mirvehicle_description/meshes/` which has the DAE files of the body and wheel of the vehicle.
2. The user should also create the folder `your vehicle` under `mirvehicle_description/URDF/.` The folder should have the `properties`, `structure` and `sensor.urdf.xacro` files defined for vehicle model along with the generic `main.urdf.xacro` which can be created with a set of sample files given in the framework.
3. The framework also provides a YAML file by the name of `mirvehicle_config` under `mirvehicle_gazebo/config/` which has the option to select Front Wheel Drive or Rear Wheel Drive based on the vehicle design. The user should select the drive and save the file as `yourvehicle_control.yaml`.
4. The user should create a vehicle specific launch file named `yourvehicle_world.launch` under `mirvehicle_gazebo/launch/` to launch the vehicle in the desired world. The framework also provides a launch file `mirvehicle_empty.launch` which is used to launch the vehicle in an empty world. The robot name and other arguments can be modified according to the user vehicle model.

Here is an example of the changes above mentioned.

```
1. mirvehicle_ws/mirvehicle_description/src/me
   shes/<your_vehicle>
```

```
2.  mirvehicle_ws/mirvehicle_description/src/UR
    DF/<your_vehicle>/main.urdf.xacro
                      properties.urdf.xacro
                      sensors.urdf.xacro
                      structure.urdf.xacro
3.  mirvehicle_ws/mirvehicle_gazebo/config/<you
    r_vehicle_control.yaml>
4.  mirvehicle_ws/mirvehicle_gazebo/launch/<you
    r_vehicle_world.launch>
```

In the further sections, this paper also explains how to build a custom world and launch it along with the model using the framework.

IV. Creating The Vehicle Model in Gazebo

This section of the paper focuses on how to create the vehicle model

A. Design of 3-D Model

The 3-D model of the vehicle was designed in Solidworks [18] and the file was exported in DAE file format along with the texture file. The body of the vehicle and wheels were designed and exported separately in order to ease the joint creation in URDF.

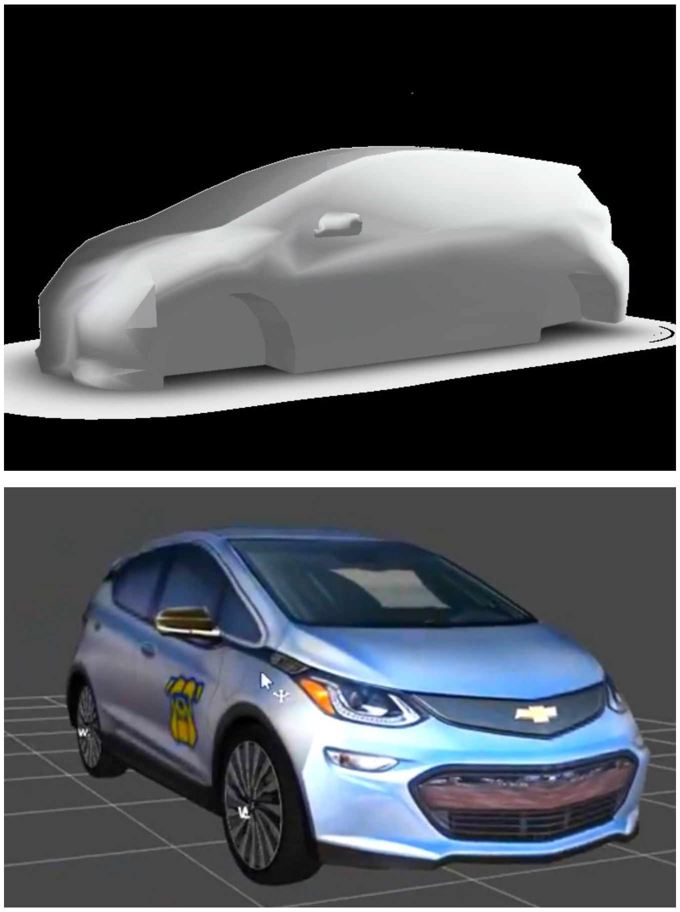

Fig 1. 3-D model of the body of vehicle.

Fig 1 shows the 3-D model of the body of vehicle without texture. Similarly, a 3-D model of the wheel of the vehicle was also designed and exported.

B. URDF Creation and Visualization

URDF (Universal Robotic Description Format) is used to create the models in Gazebo. The basic elements of URDF are links and joints [19]. Inside each link the three parts Inertial, Visual and Collision are defined. Our vehicle model will be having seven links, one for the body, one for each of the four wheels and two for steering. All the inertial and mass properties of the vehicle are given under Inertial part of the link. The DAE file is loaded in the Visual part of the link. The position and orientation of the model is also defined under the Visual part of the link. Under the Collision part, a simpler version of the vehicle model is used in order to reduce the computational time. The joints are followed by the definition of the links. There are six different types of joints namely revolute, fixed, continuous, prismatic, planar and floating. We are using continuous joints for the wheels to rotate and revolute joints for steering the vehicle. Important parameters like the type of joint, parent link, child link, axis of movement, locations of the joint are all defined inside the joint part of URDF.

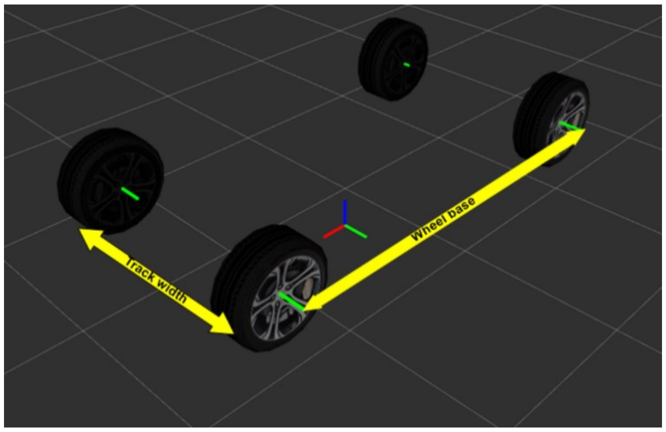

Fig 2. Trackwidth and wheelbase along with the ROS coordinate system.

In order to find out the exact position of joints and the right axis of rotation, the model of the vehicle was visualized in R-viz [7]. Fig 2 shows the co-ordinate system in R-viz. The blue line shows the positive z direction, the red line shows the positive x direction and the green line shows the positive y direction. Fig 2 also shows the basic properties like track width and wheelbase of the vehicle model along with the baselink position. Based on the visualization, the parameters like roll, pitch and yaw along the axis of rotation were corrected. For the steering angle a value of 0.6 rad/s was fed to the model which was calculated to be 34 degrees. The rotation of wheels and steering of front wheel was also visualized in R-viz with the help of Joint State Publisher.

Fig 3 shows an example of a graphical representation of the links and joints in the file `structure.urdf`. This graphical representation helps the user verify their vehicle model to the vehicle model created by the authors.

C. XACRO Conversion with Gazebo Plugins

Once a simple URDF is created, it is converted to a XACRO. XACRO is a XML macro language and is generally preferred over URDF as it can use variables by which all the properties like trackwidth, wheelbase, mass of the vehicle etc., can be defined. This is helpful as more than one link have the same properties and dimensions. Also, XACRO prevents repetitive lines in the code. All the required properties for the vehicle were defined in a XACRO file named properties. The URDF file created for the vehicle above was converted to a XACRO file by the name of structure. The sensors required for the vehicle was defined under the XACRO file named `sensor.urdf.xacro`.

In order to have our model working properly in Gazebo, the transmission for all the non-fixed joints need to be defined to let Gazebo know what to do with the joints [20]. All wheel joints were defined as velocity joint interface and the steering joints were defined as position joint interface. The above mentioned changes were done to the `structure.urdf.xacro`. Another XACRO file was created with gazebo elements [21] like Friction coefficients, maximum contact correction velocity, contact stiffness, etc. for each link of the vehicle model and the required gazebo plugins [14] like `gazebo_roscontrol`, `cmdvel_controller` and joint sate publisher were also added to the XACRO file. The XACRO file was named `mirvehicle_gazebo.urdf.xacro`. Finally a main XACRO file was created in order to link the four XACRO files, such that all the four files could be run if `main.urdf.xacro` was running.

V. Creating the Environment in Gazebo

This section focuses on creating a custom test track environment to test the vehicle model.

A. Track Design

A custom track was designed with a width of three meters and length of 965 meters. The track was also given a height of ten mm and meshes were created in NX [22]. The file was then exported to the DAE format. Further the DAE file was imported in Blender [23] along with the texture files and texture was applied to the track and the surroundings. Each of the mesh needed to be aligned properly with respect to the texture image in order to get the right texture for the track. The above mentioned process was time consuming as the track had huge number of meshes due to the high number of turns. Both the DAE and texture files were saved in the `models` folder.

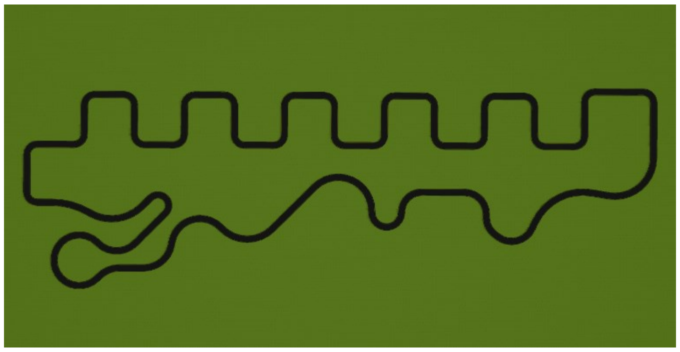
Fig 4. The track created in Gazebo along with the texture.

Fig 4 shows the test track layout and the environment in Gazebo. It can also be seen the track has been given asphalt texture and the remaining area has been given the grass texture.

B. SDF and World File Creation

The `model.config` file was created for the track which contains meta information about the model [24]. Following the config file, the SDF (Simulator Description Format) file was created for the test track. The SDF file is very much similar to

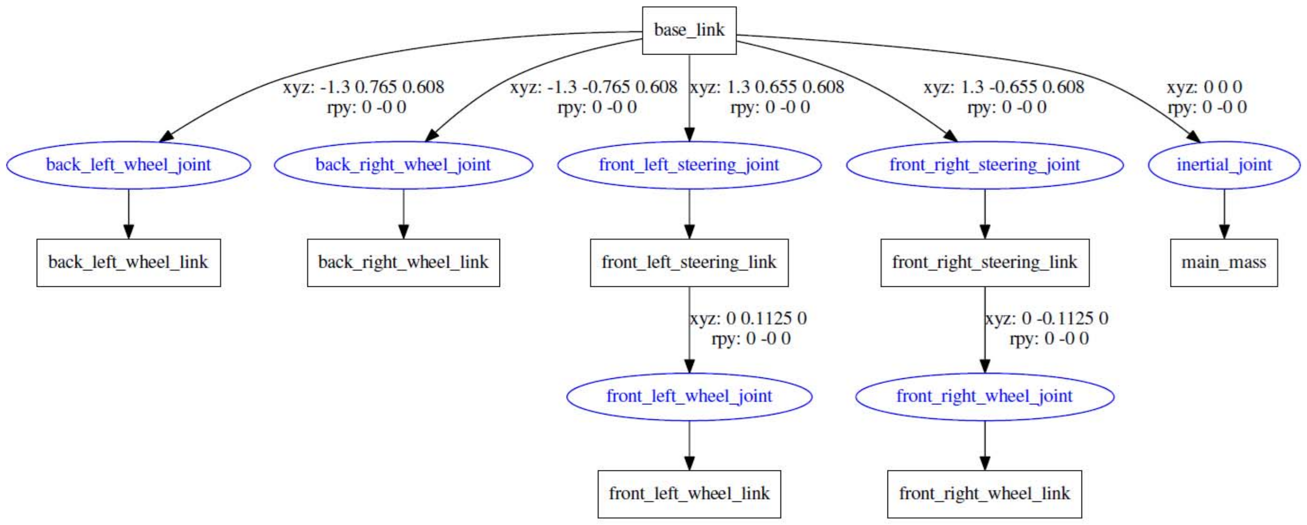

Fig 3. Graphical visualization of `structure.urdf`

the URDF file format. Hence similar to the vehicle model, the test track was also considered as a link and the inertial, collision and visual properties were defined. Once the SDF and config files are defined the test track model is available in the model database and ready to be used in any gazebo world. In order to create a custom world, a `.world` file must be created with all the models present in it. Also important properties like lighting, physics, and scene should also be set accordingly and the `.world` file should be saved. [25].

VI. Configuring Sensors and Launch Files

In this section, the paper explains how to use the sensors, configure the .yaml file, and create launch files to launch a custom environment along with a vehicle model.

A. Sensors

All the sensors necessary for the vehicle model like camera, laser, LiDAR etc., are available in the Gazebo Model Database [26]. The sensor models must be added to the XACRO file named sensors. The vehicle model created by the authors has a front camera, a Velodyne LiDAR and SICK LiDAR. Further, other packages like `obstacle_stopper` and `sicktooolbox_wrapper` uses the sensors to detect obstacles and stop the vehicle from crashing in to them.

Fig 5 shows the output of the front facing camera in the vehicle model during a left turn. The lane markings of the track can also be seen. The image was saved under the time it was recorded, hence the image name is `2018-02-11-23-29-13-000409`.

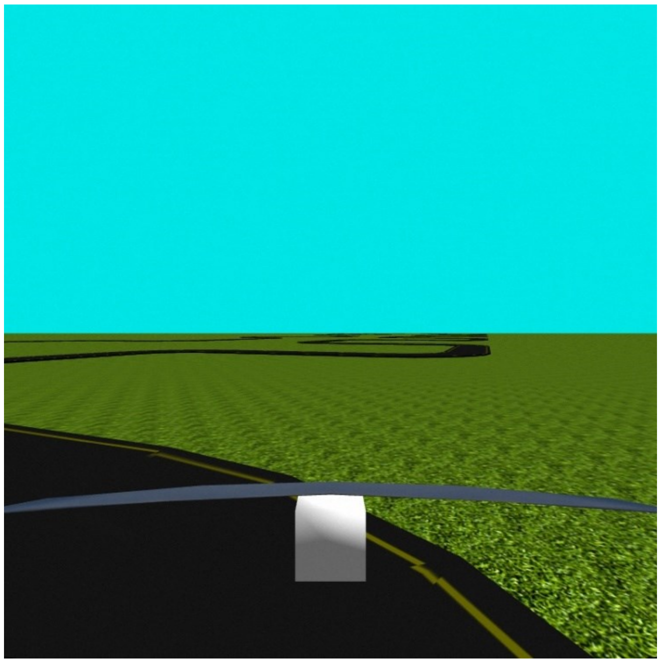
Fig 5. Output from the front facing camera during a turn.

B. Configuring the YAML File

The `.yaml` file contains the control for the joints of the vehicle. First the publish rate for the Joint state controller is set. Then based on the drive train the velocity controller is defined. Since our vehicle model is front wheel driven, the front left and right wheel joints were defined with the PID values of velocity controller. The position controller for steering joints will also need to be defined. Hence the front left and right steering joints were defined with the PID values of position controller.

C. Creating Launch File

The final step in the simulation of vehicle model is to create a launch file to launch all the required files simultaneously. First, the command for spawning the vehicle model into Gazebo is given followed by the command to load the joint controller configuration from the YAML file. The basic commands like to load controllers for driving the vehicle, to convert joint states into TF transforms for R-viz, to publish joint states that are not controlled, to run the script created for converting *cmd_vel* into gazebo and the script created for controlling the vehicle through joy stick were available with the framework by name of `mirvehicle_empty.launch`. A custom launch file was created using the `mirvehicle_empty.launch` and the vehicle model was loaded into the custom world created. Also, the command to control the vehicle model using joystick was included in the custom launch file. Hence when the launch file was launched all the above commands will be executed.

VII. Result and Discussion

In order to evaluate our framework design, a vehicle model of Chevrolet Bolt was used. Once the custom launch file was launched, all the controllers started along with the vehicle model loaded in the custom environment. The vehicle was controlled by a joystick to move around in the world. It was verified that the vehicle model resembles exactly like the real-world car.

As seen in Fig 6, the vehicle model and the custom test track were successfully built. The camera images, joint motions and the LiDAR scans were visualized. To evaluate the feasibility of the proposed framework, the vehicle model was run around the test track with a joystick and the steering and velocity data were collected along with the images. The images from the front camera were saved under the time it was recorded as explained earlier. The file name along with the steering data in rad/s and velocity data in m/s were saved in a text file.

Table 1 shows part of the collected data in the text file while running the vehicle model in the test track. The steering and velocity data for the image shown in Fig 5 is highlighted in Table 1. The steering and velocity data were collected for a time duration of 1 hour.

VIII. Conclusion and Future Scope

The purpose of this work is to build a dependable vehicle model and environment for software-in-the-loop modeling and simulation using ROS and Gazebo. The framework of the vehicle platform, creation of a vehicle model and creation of an environment are explained in detail. The details on how to

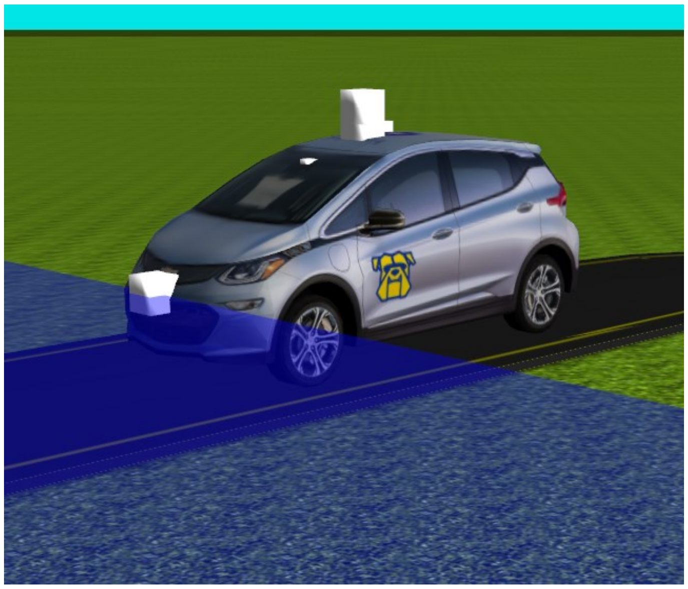

Fig 6. The test vehicle model in custom environment inside Gazebo.

Table 1. Part of the data collected while running the vehicle model in test track

Image Name	Steering (rad/s)	Velocity (m/s)
2018-02-11-23-29-12-825102	0.218694031239	1.68934880197
2018-02-11-23-29-12-865273	0.207835868001	1.68934880197
2018-02-11-23-29-12-893948	0.207835868001	1.68934880197
2018-02-11-23-29-12-926789	0.207835868001	1.68934880197
2018-02-11-23-29-12-964465	0.207835868001	1.68934880197
2018-02-11-23-29-13-000409	0.207835868001	1.68934880197
2018-02-11-23-29-13-031176	0.207835868001	1.68934880197
2018-02-11-23-29-13-065587	0.207835868001	1.68934880197

control the vehicle, and how to add further sensors are also discussed. We hope this would help anyone from the robotics community trying to conduct autonomous vehicle research with their own vehicle model in a simulated environment.

Further to the vehicle model and environment, different test scenarios and hardware in the loop (HIL) simulation for autonomous vehicles can be developed. Some of them are localization and mapping, path planning, obstacle avoidance and stop sign detection. The final goal of this research is to train a vehicle model to run autonomously in a custom test track by implementing end to end learning technique.

ACKNOWLEDGMENT

This research is supported in part by Kettering University Provost's Research Matching Fund, Development of Low-cost Autonomous Vehicle Experimental Platform, 2017. The authors would like to thank Shobit Sharma and Savan Adeshra for their continued support during this research.

REFERENCES

[1] KPMG Global Automotive Executive Survey [Online] Availble: https://assets.kpmg.com/content/dam/kpmg/nl/pdf/2018/sector/automotive/global-automotive-executive-survey-2018.pdf Accessed 2018

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS : an open-source Robot Operating Sysytem", ICRA Workshop on Open Source Software, 2009,.

[3] N. Koenig and A.Howard, "Design and Use Paradigms for Gazebo, An Open-source Multi-Robot Simulator", International Conference on Intelligent Robots and System, 2004, pp 2149-2154.

[4] K Takaya, T Asai, V Kroumov and F Smarandche, "Simulation Environment for Mobile Robots Testing using ROS and Gazebo", 20th International Conference on System Theory, Control and Computing, 2016, pp.96–101.

[5] CAT Vehicle Testbed [Online] Available :https://cps-vo.org/group/CATVehicleTestbed/about Accessed 2017

[6] Dataspeed ADAS kit [Online] Available: http://dataspeedinc.com/wp-content/uploads/2017/07/ADAS_FAQ.pdf Accessed 2017

[7] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, J. Zhang, "Manipulation Task Simulation using ROS and Gazebo", IEEE International Conference on Robotics and Biomimetics, 2014, pp 2594-2598.

[8] Digitial Asset Exchange file format [Online] Available: https://help.sketchup.com/en/article/3000168 Accessed 2017

[9] L. Kunze, T. Roehm, M. Beetz, "Towards Semantic Robot Description Languages", Robotics and Automation (ICRA), 2011 IEEE International Conference, 2011,pp 5589-5595.

[10] XACRO [Online] Available: http://wiki.ros.org/xacro Accessed 2017

[11] Simulator Description Format. [Online] Available: http://sdformat.org/spec Accessed 2017

[12] World file creation [Online] Available http://gazebosim.org/tutorials/?tut=import_mesh Accessed 2017

[13] V. Sinha, F. Doucet, C. Siska, R. Gupta, S. Liao, and A. Ghosh, "YAML: a tool for hardware design visualization and capture." Proceedings of the 13th international symposium on System synthesis. IEEE Computer Society, 2000.

[14] ROS control [Online] Available: http://gazebosim.org/tutorials/?tut=ros_control Accessed 2017

[15] Python programming [Online] Available: https://www.python.org/

[16] Obstacle Stopper [Online] Available: https://github.com/sprinkjm/obstaclestopper Accessed 2017

[17] SICK Toolbox drivers for SICK laser rangefinders [Online] Available : http://wiki.ros.org/sicktoolbox_wrapper Accessed 2017

[18] Solidworks [Online] Available: http://www.solidworks.com/ Accessed 2017

[19] Building a URDF model from scratch [Online] Available: http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch Accessed 2017

[20] Using A URDF in Gazebo. [Online] Available: http://wiki.ros.org/urdf/Tutorials/Using%20a%20URDF%20in%20Gazebo Accessed 2017

[21] URDF in Gazebo. [Online] Available: http://gazebosim.org/tutorials/?tut=ros_urdf Accessed 2017

[22] NX for Design. [Online] Available: https://www.plm.automation.siemens.com/en/products/nx/for-design/index.shtml Accessed 2017

[23] Blender – Free and open source 3D creation suite. [Online] Available: https://www.blender.org/ Accessed 2017

[24] Model Structure and Requirements. [Online] Available: http://gazebosim.org/tutorials?tut=model_structure&cat Accessed 2017

[25] Modyfying a World. [Online] Available: http://gazebosim.org/tutorials/?tut=modifying_world Accessed 2017

[26] Gazebo Model Database. [Online] Available: https://bitbucket.org/osrf/gazebo_models/src Accessed 2017

[27] J. Burbank, W. Kasch, and J. Ward, "Hardware-in-the-Loop Simulations," in An Introduction to Network Modeling and Simulation for the Practicing Engineer, 1, Wiley-IEEE Press, 2011, pp.114-142.