**Setup and installation of LG and API**

24-Feb-2021

This document contains the information of Setup and Installation process for running the LG Simulator binary in Linux/Unix and using the Application Programming Interface (API) for Lidar.

With the completion of these steps the user shall be able to use the API in Python user scripts to modify the Point cloud data frame by frame.

**Note:** Since we are using LG Simulator it will require a high-end server with Graphics card to run it smoothly. The Simulator performance will be very slow if run on a lower PC configuration.

Below is the PC configuration required:

**PC Configuration:**

* CPU: Intel i7 10700
* GPU: Nvidia RTX 2070 (8GB memory)
* OS: Ubuntu 64-bit

**Package Includes:**

* LG\_Simulator Binary
* PythonAPI’s Package
* TestScript.py
* Document for reference

**Prerequisite:**

* **Ubuntu**
* **Nodejs 12.16**
* **Python3.5 or later**

**Setup:**

* Copy Simulator binary in local path of the Ubuntu
  + ex : #cd pts/lg\_simulator\_binary
* Copy the Python API in the local path of Ubuntu
  + ex : #cd pts/PythonAPI
* Copy the user script in the same path parallel to the Python API
  + ex : #cd pts/TestScript/TestScript.py

**Run and Test Simulator:**

* Open terminal and go to the simulator package path
* Ubuntu - Install Vulkan user space library
  + **#sudo apt-get install libvulkan1**
* Run Simulator using the below command
  + **#sudo ./pts/lg\_simultor\_binary/simulator**
* Click Open Browser or Open any browser, enter localhost:8080 in address bar

Note: If it asked for register then register it and login with the same credential.

* Once it is logged in successfully, you will find the 4-tab in the left corner which are Maps, Vehicles, Clusters, Simulations.
* In the Maps tab, **Add new** map with the URL to an environment asset bundle or use below string to add the Cube Town Map

**Name - CubeTown**

**Asset Bundle String-**

**https://assets.lgsvlsimulator.com/085da734088f2b584075fce2d1d478b98ca076eb/environment\_CubeTown**

* In the Vehicles tab, **Add new** vehicle with the URL to a vehicle asset bundle or use the below string to add the Lexus

**Name -** **Lexus2016**

**Asset Bundle String- https://assets.lgsvlsimulator.com/ea5e32fe566065c6d1bbf1f0728d6654c94e375d/vehicle\_AWFLexus2016RXHybrid**

* In the Vehicle tab after adding the vehicle, Click the wrench icon next to the vehicle name
  + Add the below config-

**Bridge: ROS2**

**Sensor Json Config:**

**[**

**{"type": "Transform","name": "base\_link", "transform": { "x": -0.015, "y": 0.369, "z": -1.37, "pitch": 0, "yaw": 0, "roll": 0}},**

**{"type": "CAN-Bus","name": "CAN Bus", "params": { "Frequency": 10, "Topic": "/lgsvl/state\_report" }, "transform": {"x": 0,"y": 0, "z": 0,"pitch": 0, "yaw": 0, "roll": 0}},**

**{"type": "GPS Device", "name": "GPS", "params": { "Frequency": 12.5,"Topic": "/gnss/fix", "Frame": "gnss", "IgnoreMapOrigin": true },"parent": "base\_link", "transform": {"x": 0, "y": 0, "z": 0, "pitch": 0, "yaw": 0, "roll": 0}},**

**{"type": "GPS Odometry","name": "GPS Odometry", "params": {"Frequency": 30.0, "Topic": "/lgsvl/gnss\_odom","Frame": "odom", "ChildFrame": "base\_link","IgnoreMapOrigin": true},"parent": "base\_link", "transform": {"x": 0, "y": 0, "z": 0, "pitch": 0, "yaw": 0, "roll": 0}},**

**{"type": "IMU","name": "IMU","params": {"Topic": "/imu/imu\_raw","Frame": "imu"}, "parent": "base\_link", "transform": {"x": 0, "y": 0, "z": 0, "pitch": 0, "yaw": 0, "roll":0 }},**

**{"type": "Lidar","name": "LidarFront","params": { "LaserCount": 16, "MinDistance": 2.0, "MaxDistance": 100, "RotationFrequency": 10, "MeasurementsPerRotation": 360, "FieldOfView": 20, "CenterAngle": 0, "Compensated": true, "PointColor": "#ff000000","Topic": "/lidar\_front/points\_raw","Frame": "lidar\_front" },"parent": "base\_link", "transform": {"x": 0.022, "y": 1.49, "z": 1.498, "pitch": 0, "yaw": 0, "roll": 0}},**

**{"type": "Vehicle Control","name": "Autoware Car Control","params": {"Topic": "/lgsvl/vehicle\_control\_cmd"}},**

**{"type": "Vehicle State","name": "Autoware Auto Vehicle State","params": {"Topic": "/lgsvl/vehicle\_state\_cmd"}},**

**{"type": "Keyboard Control","name": "Keyboard Car Control"},**

**{"type": "Clock","name": "Simulation Clock","params": { "Topic": "/lgsvl/clock"}}**

**]**

* Click on Simulations tab and create a new Simulation. Give it a name and check the **API Only** option. Click **Submit**
* Select the API\_Only simulation from the simulations tab.
* Press the Play Button
* The Unity Application window should now show API\_Only on the screen.

**PythonAPI:**

* Open terminal and go to the Python API folder and enter the below command to install the Python files and necessary dependencies.
  + **#cd pts/PythonAPI/PythonAPI/**
  + **#pip3 install *--user -e .***
* Run the following example to see the API in action:
  + **#python3 ./quickstart/TestScript.py**

**Python Script for end user:**

* We have added the functionality to fetch and modify the lidar points cloud data of LG Simulator. For this we have provided an **API interface** to modify the lidar point cloud data frame by frame in Python.
* Below is the control flow in the code:

Lg\_Simulatror\_LidarPointsData  Output\_PythonAPI  ModifyThePoints  AgainSetTotheLGSimulator  UpdatedPointsInSimulator  PublishToROS2Bridge  AD Stack

* API’s introduced to use and modify the lidar points cloud data
* Custom call back which will helps to fetch the customize data of the LG Simulator
  + **on\_custom()**
* Class to interact with simulator
  + **LidarContext**
    - **var Points**
* Function to apply the created object to the simulator
  + **apply\_lidar\_context()**
* Below is the example of how to use the API’s to modify lidar data in python script

**def UpdateLidarData(agent, kind, context): # In context it will return the data**

**if kind == "mylidar": # Check if the context is kind of Lidar**

**lidarPoint = context["Points"] # Fetch the Points from the context jsonarray**

**y = json.loads(lidarPoint) # Get json data to use and modify**

**print("json data: ", y["array"]) # Print the complete data**

**#modify y[“array”] which have lidar point cloud data**

**s = lgsvl.LidarContext() # Create the LidarContext object**

**s.Points = y["array"] # Set modified lidar Points to the object**

**a.apply\_lidar\_context(s) # Apply updated object to the simulator**

**else:**

**pass**

**a.on\_custom(UpdateLidarData) # Call the API’s on custom call**

**Note: TestScript.py is the sample script to run and test the above API’s**