GEORGE BROWN PRESS
JOURNALS, BOOKS & PROCEEDINGS

Journal of Computer, Signal, and System Research

Article

Research on the Simulation Test of Intelligent Connected Automobile Sensor Based on Virtual Reality Technology

Xiaoyu Xu [1,*] and Joan Lazaro [1]

[1] University of the East, Manila, Philippines
* Correspondence: Xiaoyu Xu, University of the East, Manila, Philippines

Abstract: With the continuous development of virtual reality technology and intelligent and connected vehicles, Sensor test data based on actual vehicles are also increasing, The time required for road testing has increasing, Testing is also becoming more difficult and expensive, In order to improve the efficiency of measurement collection and autonomous driving algorithm development for large and laser sensors of intelligent connected vehicle cameras, Reduce the development costs, This study is based on the virtual reality technology, Using panosim software to build a simulation test platform, By creating a road test map in real time, selecting a vehicle model, selecting and installing the sensor type, Using python to read the camera image data and lidar point cloud information, Further through unity3D synchronization simulation screen for road analysis, Receive the image data transmitted through the TCP / IP connection on the server side, And display the received images using the OpenCV, To verify the accuracy of the algorithm, Using virtual simulation technology shortens the sensor development and test cycle to some extent, Reduced costs, It has practical significance.

Keywords: intelligent connected vehicle; virtual reality technology; panosim; python

1. Introduction

With the continuous penetration of artificial intelligence and the Internet of Things technology in various fields, Intelligent and connected vehicles have also ushered in new opportunities for development, Intelligent connected vehicles perceive the surrounding environment in real time through cameras, lidar and other sensors, The data collected by the sensor is processed through an integrated algorithm developed within the autonomous driving processor, Then control the vehicle by real-time chassis technology, To achieve the effect of autonomous driving, therefore, Of the identification of objects and environment around the vehicle and the acquisition of data, Well-known industries and enterprises at home and abroad have invested a lot of research, Its core problem is the research of sensor-related algorithms of intelligent connected vehicles, Algorithms cannot be studied without a lot of sensor data, So how to collect the data from the sensor more effectively, It is an urgent problem to be solved.

With the continuous advancement of network technology, virtual reality technology has been widely applied in fields such as medicine, automobiles, and intelligent manufacturing. By integrating digital twin and V2X technology, a digital twin-based autonomous driving testing framework has been proposed, providing non-line-of-sight perception information, enabling efficient vehicle-to-data connectivity, and achieving the integration of physical and digital spaces [1]. A simulation testing approach that combines the real world with autonomous driving was introduced, utilizing digital twin technology to reconstruct vehicle models and test roads in a three-dimensional space. This approach also employs a collision detection algorithm to predict the surrounding environment, successfully enabling interaction between simulated vehicles and other traffic scenarios. Based on the concept of digital twins, the application status and development prospects

Received: 10 January 2025
Revised: 15 January 2025
Accepted: 24 January 2025
Published: 25 January 2025

of virtual testing and urban transportation in autonomous driving were analyzed. The review highlights that virtual reality technology can be effectively applied to the simulation testing of intelligent connected vehicles, significantly reducing development costs [2].

To improve the efficiency of data collection from sensors such as smart-connected vehicle cameras and lidar, Reduce the development cycle and development cost of intelligent connected vehicle algorithm, This paper studies the simulation test of intelligent connected vehicle sensor based on virtual reality technology, Road maps and scene creation through the panosim, panosim The software integrates complex road scenes, pedestrians, traffic signs and other information, And offers a rich selection of vehicle models, Can realize the installation of mainstream sensors such as cameras and lidar on any model, Data from the sensor was obtained via the pycharm, Convenient to modify the sensor logic from the bottom layer, Finally implementation algorithm test, Finally, to be tested through the unity3D, To obtain the test results, At the same time, the collected data is converted into PNG format pictures for local preservation, As a dataset used for training, It has theoretical research value and practical significance for the development of intelligent and connected vehicles [3].

2. The Construction of the Simulation Platform Architecture

Reference bidirectional link table structure of intelligent connected vehicle sensor simulation platform, Connecting nvidia, panosim simulation platform and pycharm platform through TCP / IP communication protocol service, The structure diagram is shown in Figure 1, The platform has cross-software, cross-language, cross-platform, The communication settings among the three groups is crucial, If the parameters set parameters will cause the data of the entire simulation platform to be communicate communicated, Configuration parameters mainly refer to the communication IP and communication port, IP needs to be set to computer IP "192.168.3.102" and port number "14311", In the configuration interface of the panosim, configure Agent, Then, in the right column, find the functional script CameraTcpOutput.py to obtain the camera raw data and the lidar raw data script LidarTcpOutput.py, Drag it into the main car model in the middle, Through this functional script, panosim, the camera image data and lidar point cloud data can be output to the outside in a TCP / IP protocol format, Open the camera engineering file "camera_panosim.py" and the lidar engineering file "lidar_panosim.py" in pycharm, Find " server_address = ('192.168.3.102', 14311)", IP consistent for the IP and panosim functional scripts in the modified code, Both are "192.168.3.102", Port number is "14311". Complete pycharm communication configuration, the NVIDIA host, panosim and pycharm under the same ip and port data communication, in addition to the NVIDIA host environment configuration, mainly based on deep learning application CUDA kit and deep neural network GPU acceleration library, the kit can realize the subsequent sensor test in pedestrian and vehicle detection task of YOLOv3 target detection model, simulation platform architecture.

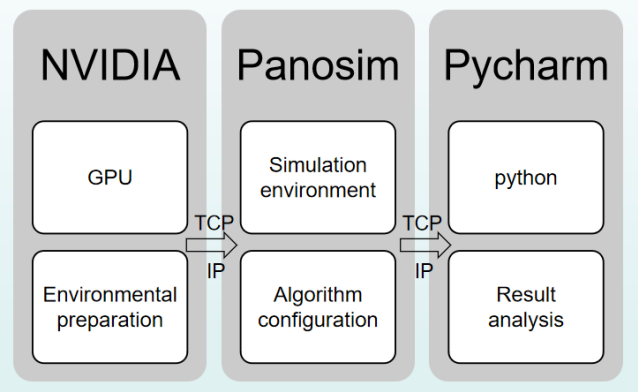

Figure 1. Architecture Diagram of the Simulation Platform.

3. Design and Implementation of the Simulation Platform

3.1. Create a Map

In PanoSim software, FieldBuilder tool is used to map, and appropriate tools and materials are selected to draw roads, highways, buildings, etc., and adjust their parameters and attributes. Maps are created according to the research requirements as shown in Figure 2.

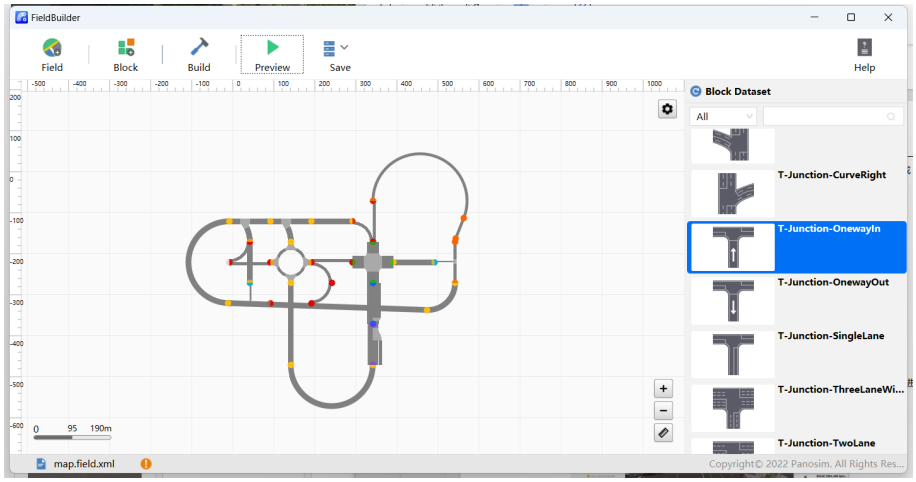

Figure 2. Map Drawing.

After the map is drawn, save the map and open the Unity software through the Preview function. Import and load the saved map file in the Unity. Preview the map effect through the perspective, and check whether the road, traffic signs and scene elements meet the expectations. If a problem is found, return to FieldBuilder to modify and save it again, and preview it in Unity again. The effect of Unity simulation map preview is shown in Figure 3 until the map effect is correct, and the map is finally saved and the map is created.

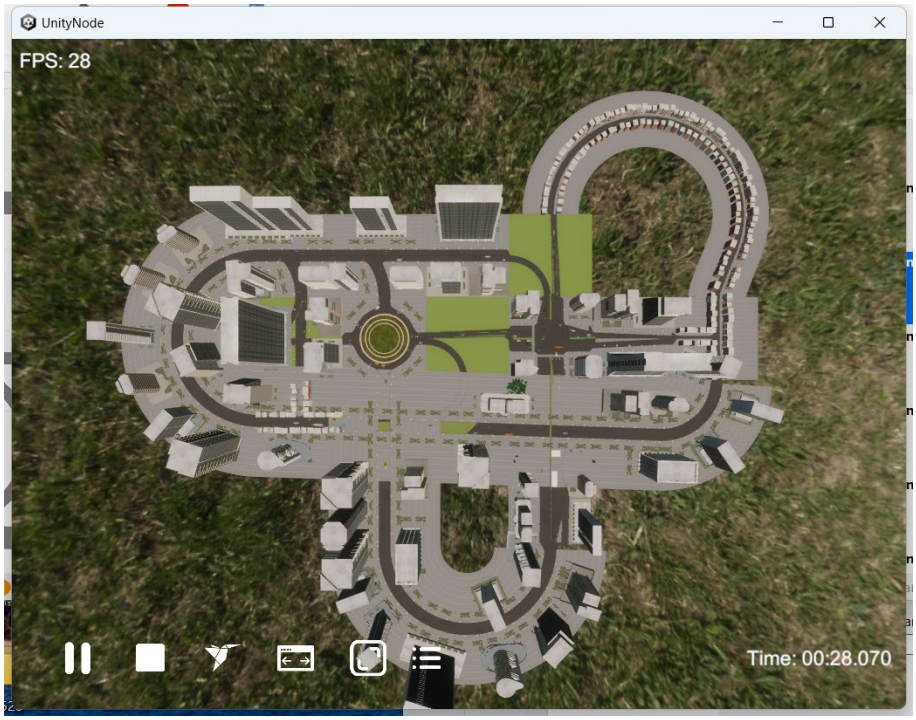

Figure 3. Effect Diagram of the Unity Simulation Map Preview.

3.2. Simulation Scene Construction

Virtual simulation scene construction is mainly based on the panosim software simulated under the actual road environment construction, Add traffic sign information TrafficSign and traffic facility Facility to the panosim software, Including traffic lights, pedestrian crossings, parking Spaces, Traffic signs include speed limit signs, no-traffic signs, warning signs, In a virtual simulation environment, Different scene types have their own parameters and meanings, X, Y, and Z represent the position coordinates of the traffic signal lights in the road coordinate system, Yaw indicates the yaw angle, Pitch indicates the pitch angle, Roll indicates the roll angle, CycleTime Indicates the cycle time of the traffic signal light. For the crosswalk, X and Y indicate the positional coordinates of the crosswalk in the road coordinate system, Yaw indicates the yaw angle, Length represents the length of the crosswalk, and Width indicates the width of the crosswalk. For parking space, Name refers to the name or label of the parking space, Open said whether open parking space, Arrow says parking space arrow direction, Bar said parking limit setting, Opacity represents transparency, X and Y represents the location in the road coordinate coordinates, Yaw said pendulum Angle, in the simulation scene need to set up each parameter, scene building can be completed by loading the map by World Dataset to view the effect.

3.3. Install the Sensor

Before installing the sensor, select the required model and name it, and place it in the built simulation environment [4]. After selecting a good model, you need to install the simulation test sensor on the car, First, install the camera, In Edit Sensor, select the Mono Camera Sensor under the Camera menu, Install the camera in the front windshield position, Next, install the lidar system, Under the Edit Sensor select Lidar menu, Surround Lidar Point Cloud Sensor, Install the lidar on the roof before the test, The camera and lidar are installed in the panosim interface, The top and side view of the sensor installation are shown in Figure 4.

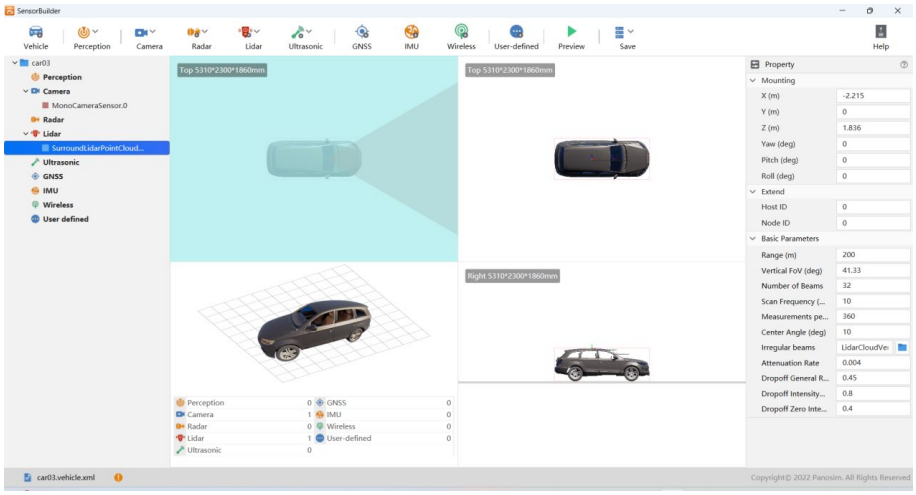

Figure 4. Sensor Installation Diagram.

4. Algorithm Design

After the sensor is built, it needs to be connected to intelligent driving algorithm for path planning and navigation, and use global planning A * algorithm and python language.

The A * algorithm divides the node states into three different preparation stages through search traversal, which are not checked, is checked and has been checked. By introducing the heuristic function, it can effectively measure the actual consumption of

path nodes, and thus effectively improve the efficiency and accuracy of path planning. Compared with the traditional traversal search method, A * algorithm is more flexible, more effective, and more can meet the actual needs. The probability of the best path is measured by evaluating the global information with the heuristic function $f(n)$. To better achieve this goal, two key list lists are built that store already and unexperienced grid nodes, and combined with the heuristic function, can effectively estimate the distance between any grid node on the map and the target node. The closer the valuation is to the actual consumption, the better the search direction in four or eight neighboring cases. The main expression of the algorithm is shown in Equation (1).

$$f(n) = h(n) + g(n) \quad (1)$$

The $f(n)$ is used to estimate the cost of reaching the target state from the initial state, the $g(n)$ is used to estimate the actual movement consumption, and the $h(n)$ is used to estimate the lowest target state.

Algorithm design

The traditional A * algorithm can implement the search tasks more effectively by setting the open and closed list. The Open list contains the current nodes and their neighboring nodes as the starting point for the next search, while the Closed list contains the pending nodes that meet the search requirements of the A * algorithm to complete the search task more efficiently. The search process of the traditional A * algorithm is as follows:

Step 1: Set up two lists, namely, the open list (Open List) and the closed list (Closed List). Add the start node S to the Open list.

Step 2: The algorithm searches for all possible extended nodes in the open list. If not, the algorithm exits and will not find a feasible path.

Step 3: In the Open list, all the nodes to need to be searched are counted to determine which nodes n have the smallest f (n) value, n is regarded as the next node of the algorithm path and put in the Closed list for further screening, and then deleted in the Open list.

Step 4: Determine whether the expansion node searched in the second step coincides with the target point G. If the overlap means that the search is successful, otherwise, perform the next step;

Step 5: Based on the third step, search and calculate all the child nodes m in its neighborhoodand so on f(m) Values, by comparison to obtain the f (m) The minimum value, taking the corresponding child node as the next node selected by the algorithm;

Step 6: Check whether there is a node m in the open list and the closed list.

1) If the open list or the closed list does not contain the node m, then add it to the open list, and add a pointer to point the child node m to the parent node n. When there is a formal path between two nodes, you can output paths according to the pointer.
2) In the open list, the valuation function value f of node m_0(m) Used to interact with the f (m) Compare the size if its valueless-than f(m), Which indicates that a better path has been found. Meanwhile, the parent pointer of node m is modified to point to node n in the step on the current node, and node m is added to the closed list.
3) In the closed list, node m is designated to be on the optimal path, so continue to expand other child nodes in comparison to achieve the optimal results.

Step 7: cycle step 2 to step 6, when the target node G or Open list is empty, the algorithm will terminate and output the optimal solution, otherwise the solution will not be output.

The procedure flow of the A star algorithm is shown in Figure 5.

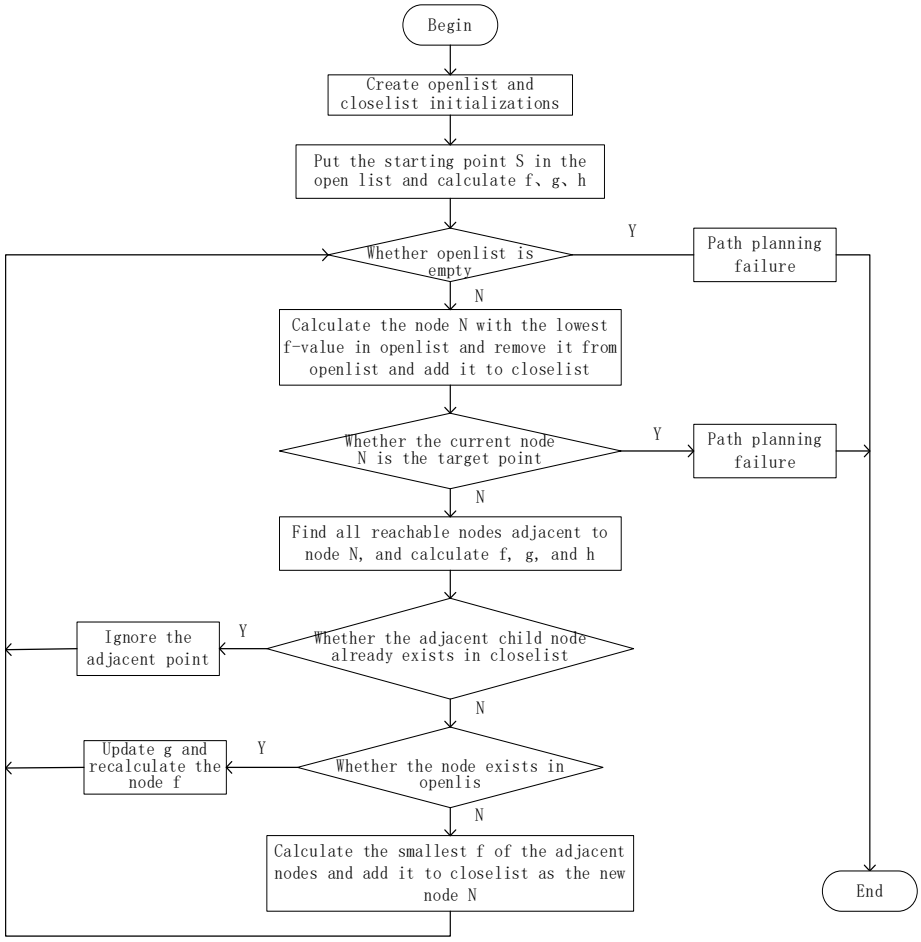

Figure 5. A Star Algorithm Flowchart.

Python Code implementation part of the code is shown in Figure 6, first, By importing the required modules such as ` numpy `, `Rotation` and `minimize` in ` scipy `, Several key functions are constructed: the `transform _ points` function rotates and shifts the point set according to the input transformation parameters, Use `Rotation` class to convert Euler angle into rotation matrix and perform matrix multiplication and vector addition; The `compute _ normal _ distributions` function calculates the normally distributed weights between the target point set and the input point set, The weight is obtained by calculating the sum of the difference and taking the exponential; The `compute_score ` function calls `transform _ points` to transform the source point set, Scores were then calculated using `compute _ normal _ distributions`, Turn the score; The ` icp ` function uses the `minimize` function to optimize the `compute_score ` with the ` BFGS ` method to find the optimal transformation between the source point set and the target point set, Take the source point set, the target point set as parameters, And output the optimization results; Finally, random source and target point sets are generated in the ` man ` function, Set the initial transformation parameter as a zero vector, Call the ` icp ` function to calculate and print the final transformation parameters, The original code was also optimized, Add the exception handling in the `transform _ points` function, Vectorization operations for distance calculation using ` cdist ` in the `compute _ normal _ distributions` function improve efficiency [5].

```python
import numpy as np
from scipy.spatial.transform import Rotation as R
from scipy.optimize import minimize
1 usage
def transform_points(points, transform):
    rotation_matrix = R.from_euler('xyz', transform[:3]).as_matrix()
    transformed_points = np.dot(points, rotation_matrix.T) + transform[3:]
    return transformed_points
1 usage
def compute_normal_distributions(points, target_points):
    distributions = []
    for point in target_points:
        diff = points - point
        dist_sq = np.sum(diff**2, axis=1)
        weight = np.exp(-dist_sq)
        distributions.append(weight)
    return np.array(distributions)
1 usage
def compute_score(transform, source_points, target_points):
    transformed_points = transform_points(source_points, transform)
    distributions = compute_normal_distributions(transformed_points, target_points)
    score = np.sum(distributions)
    return -score
1 usage
```

Figure 6. Program Python Implements the Partial Code of the A * Algorithm.

5. Simulation Test and Analysis

Cameras and lidar act as very important sensors in intelligent and connected vehicles. Represented by tesla car companies commonly used cameras as intelligent sensors, called figure intelligent driving, mainly used to identify traffic lights, lane lines, pedestrians, etc., through the camera can clearly observe the color of the object, motion and logo, the current research is generally believed that camera is influenced by light, has certain limitations, so auxiliary laser radar as intelligent made car sensor has certain necessity. Huawei Cyrus uses lidar as the main sensor of intelligent driving, which is called intelligent driving without map. With strong anti-interference ability and long detection distance, lidar realizes the detection of vehicles, pedestrians and obstacles in high-speed driving mode, and is displayed through point cloud data. Therefore, this study is based on the current mainstream map driverless and map driverless technology, combined with the characteristics of camera and lidar for simulation test and analysis.

5.1. Camera Simulation Test and Analysis

In the process of intelligent connected vehicle image recognition, the task of pedestrian and vehicle detection based on the front-facing camera is a key link. In this study, the python program implemented vehicle and pedestrian detection in the simulation environment according to the created environment of panosim simulation. The python program uses the deep learning model YOLOV3 as the detection algorithm for vehicles and pedestrians. First, the weight files, configuration files and tag files of YOLOv3 are loaded in the python program, and the YOLOv3 model is accelerated using CUDA through the GPU of NVIDIA. Create a TCP server in the program to wait for the client connection, and receive image data after the client connection, the received image data is decoded into OpenCV format, and detected using the YOLOv3 model, including borders, confidence and category, these results are filtered by non-maximum suppression (NMS), and draw the detection border and labels on the image, visualized the identification results, ensure that the algorithm can correctly identify the target in the simulation environment, and visually display the detection results [6].

When the camera simulation test can be replaced according to the requirements of different scenarios, Set the scene parameters in the panosim, Can choose sunny, rainy,

snowy, daytime, night and other different environments for algorithm test, Can also adjust the camera position and parameters, Test the recognition effect in different scenarios, Record and compare the identification results, Analyzing the performance of the algorithm in different scenarios, Further adjust the sensor and the algorithm parameters according to the test data, Thus improving the identification accuracy, Optimize the identification effect, Finally, complete the camera simulation test and analysis work, Based on the camera simulation test, as shown in Figure 7, The parameters such as the type of the obstacle can be judged by the simulation test.

Figure 7. Pedestrian and Vehicle Detection Diagram.

5.2. Simulation Test and Analysis of Lidar

Lidar is one of the commonly used sensors in intelligent connected vehicles. It can obtain point cloud data from the environment around the vehicle through laser scanning, providing accurate positioning and perception information for autonomous driving. In the panosim simulation environment, the lidar scanning process can be simulated, and the point cloud data can be processed and analyzed.

First, the parameters of lidar are set in panosim, lidar is generally installed on the top of the vehicle, you can select the lidar with 32 lines or higher beam, set the correct name, IP and port of the lidar sensor, the TCP data transmission script of the sensor is configured, obtain the sensor data using Pycharm, design and run the lidar data acquisition python program, by creating a TCP server, the program receives the point cloud data sent from the client and decodes it as the three-dimensional coordinates of the point. After receiving the data, the point cloud data is visualized using the Open3D library, colored as the point cloud data according to the Z-axis value, and displayed in the window. The program runs continuously and updates the point cloud data in real time until the window closes. After running the simulation scene with panosim, the lidar is allowed to scan the surrounding environment and obtain the point cloud data. The point cloud processing algorithm is used to process the obtained point cloud data, such as filtering, segmentation, clustering and other operations, so as to extract target objects such as vehicles and pedestrians.

In the PanoSim software, first enter the project to be edited, click the Agent in the left column in the element editing interface, enter lidar or LidarTcpOutput.py in the search box on the right bar for retrieval, select the LidarTcpOutput.py function script and drag it into the middle road model, change" Remote IP" to "192.168.1.100" is the same as the python program, " Port" is changed to "14322", the lidar point cloud visualization interface is shown in Figure 8. Through testing and analysis of lidar simulation, we can evaluate the performance of lidar in autonomous driving, including its scanning accuracy, stability, reliability and other aspects. At the same

time, we can also optimize and improve the parameters and algorithms of lidar according to the test results, so as to improve its application effect in autonomous driving.

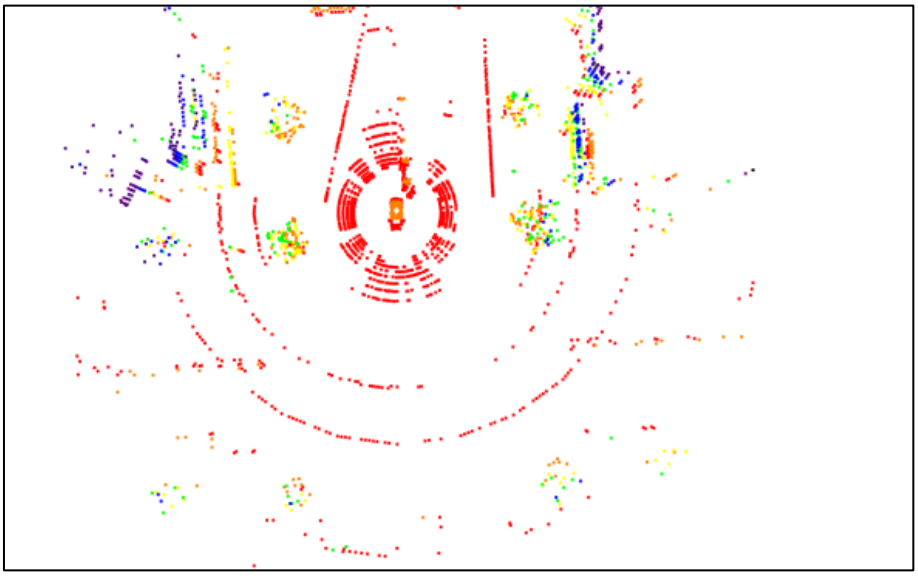

Figure 8. Lidar point cloud map.

6. Conclusion

Based on the development idea of panosim software combining virtual and real, Through the cross-platform implementation of intelligent connected vehicle sensor simulation test, Get the camera and lidar data via Pycharm, Convenient tools such as map creation and environment building, vehicle and main parameter selection, sensor layout and parameter selection, Complex real vehicle test scenarios can be presented through simulation resources, Greatly reduced the development costs, Improved the efficiency of intelligent and connected vehicle sensor testing, Using the python program to connect to the panosim software, Finally, the Unity3D platform is used to display the test scenarios in real time, This study is applicable to multiple AI fields, Aable to perform complex simulation studies.

With the rapid development of artificial intelligence technology and the continuous expansion of the autonomous vehicle market, the importance of sensor simulation testing in the development of autonomous driving has become increasingly prominent. In the future, we will continue to deepen the research on sensor simulation test technology, explore more efficient and accurate test methods, and provide more reliable technical support for the research and development and application of self-driving vehicles.

References

1. R. Sethuraman, K. Venkataramani, M. M. Y. Devi, and S. Subbaraj, "Enhancing autonomous vehicle performance with ensemble weighted support vector-based optimization in cloud," *Cluster Comput.*, vol. 3, pp. 192-192, 2025, doi: 10.1007/s10586-024-04706-x.
2. E. Landolfi and C. Natale, "An adaptive cascade predictive control strategy for connected and automated vehicles," *Int. J. Adapt. Control Signal Process.*, vol. 10, pp. 2725-2751, 2023, doi: 10.1002/acs.3658.
3. P. Juntao, N. A. Tu, W. Sujun, D. H. Huifan, and Z. Hui, "Fuzzy unknown input observer for estimating sensor and actuator cyber-attacks in intelligent connected vehicles," *Automotive Innov.*, vol. 2, pp. 164-175, 2023, doi: 10.1007/s42154-023-00228-1.
4. E. Biagioni, S. Giordano and C. Dobre, "Ad Hoc and Sensor Networks," in *IEEE Communications Magazine*, vol. 55, no. 7, pp. 172-172, July 2017, doi: 10.1109/MCOM.2017.7981546.
5. Z. Peng, "Research on path planning algorithm of substation inspection robot based on improved A-star algorithm," (in Chinese), M.S. thesis, Hunan Univ. of Technol., 2024, doi: 10.27730/d.cnki.ghngy.2024.000593.
6. F. Zhao, M. Zhang, and W. Sun, "Vehicle terminal system of intelligent logistics based on STM32F103 and BDS/GPS technologies," in *Advances in Energy Science and Equipment Engineering II Volume 1: Proceedings of the 2nd International Conference

on Energy Equipment Science and Engineering (ICEESE 2016)*, S. Zhou, A. Patty, and S. Chen, Eds. Boca Raton, FL, USA: CRC Press, 2017, pp. 1871–1876, doi: 10.1201/9781315116167.