BUILDING OPENDRIVE MODEL FROM MOBILE MAPPING DATA

M. Barsi [1, *], A. Barsi [1]

[1] Dept. Photogrammetry and Geoinformatics, Budapest University of Technology and Economics, Hungary - barsimark@gmail.com, barsi.arpad@emk.bme.hu

Commission IV, WG IV/1

KEYWORDS: self-driving car, HD-map, OpenDRIVE standard, software development

ABSTRACT:

The development of self-driving vehicles has been efficiently accelerated by applying computer simulations. These environments can achieve the best result only if reality has been digitized: highly accurate and detailed maps are required. The high-definition (HD) maps aim to fulfill these requests. Companies and academia are seeking adequate technology, where OpenDRIVE standard seems to win this race.
A software was developed to create high-definition road models for self-driving car testing purposes, which can also be used in driving simulators, for example, for testing vehicle prototypes.
The data of a sample area was received located in northern Budapest, which included inconsistent information about the lanes of the road network. The top-priority task was to build a highly structured, consistent model satisfying this open standard, an increasingly popular modeling format to describe the environment for various simulating software. In order to do that, the relevant information had to be extracted from an AutoCAD file. The result was a collection of points that was added to a software module written in C++. The code has built a structure from those points, including determining connections between lanes, building roads from the lanes, determining connections between roads, adding road marks, and building the final OpenDRIVE model.
Two different sets of data have been used thus far in the software: a circa 8 km long road section and a separate junction — meaning about 20,000 points to process. Two independent models were created; both were tested in various automotive simulators – including IPG CarMaker, probably the most well-known simulating tool used in car manufacturing – which showed that both are indeed correct. The overall time required to build those models from simple text files to well-structured OpenDRIVE models is no more than two minutes, while before the software development – when all of those aforementioned processes were done by humans – it required several weeks, if not months.
The software is already capable of building basic road models but is nowhere near finished. Numerous other features can be added to it as well, for instance, traffic signs and signals and environmental elements such as roadside objects, trees, and buildings.

1. INTRODUCTION

The idea of having self-driving, autonomous vehicles on the roads has changed from a future possibility and dream to inevitability over the last couple of years. Due to the numerous technological breakthroughs and innovations, important pieces of both hardware and software were designed, such as various cruise systems, driving assistants, or even highway autopilots. These were created to help and support the driver who is still the one actively driving or supervising the car; technology only grants them valuable information and quality-of-life improvements. However, as the world is moving towards having fully capable, autonomous means of transportation, less and less human interaction is needed (KPMG 2020). At the end of this process, the computers inside the vehicle are going to be in charge at all times, meaning that steering wheels, pedals, or gear sticks are no longer required (RAND 2020, SAE 2020).
In order to develop the supporting technology for this humongous achievement in a safe manner, highly accurate computer simulations are an absolute necessity as they enable companies to test their products in almost any circumstances, weather condition, and area without the need to physically move there with all of their equipment. These environments can only reach their full potential if the simulated world can accurately represent reality and all the various, unique conditions of said area.

To fulfill these requirements, many description formats were designed by various organizations, out of which the OpenDRIVE standard seems to be the most successful as it was designed explicitly for driving simulators, capable of managing not only the road itself but its surrounding objects as well, and is available without the need of licensing in all the countries in the world. This format suits the needs of the most well-known auto manufacturer brands – such as Audi, BMW, or Daimler – and multiple universities are involved in the development process (Apollo 2020, OpenDRIVE 2021, USDOT 2020).
The remainder of this paper is as follows: Section 2 introduces the OpenDRIVE; Section 3 demonstrates the pilot sites from where the input data was taken; Section 4 details the backbone algorithms of the software; Finally, Section 5 and 6 summarize the results and propose a few future development ideas.

2. THE OPENDRIVE STANDARD

The OpenDRIVE standard is probably the most promising way to describe road networks with all connecting and surrounding objects. The fully detailed, official description exceeds the context of the current article; hence only a short insight is given here, which enables the reader to fully understand the latter chapters. This includes the structure, main components, and the basic concept of the standard (OpenDRIVE 2021).

* Corresponding author

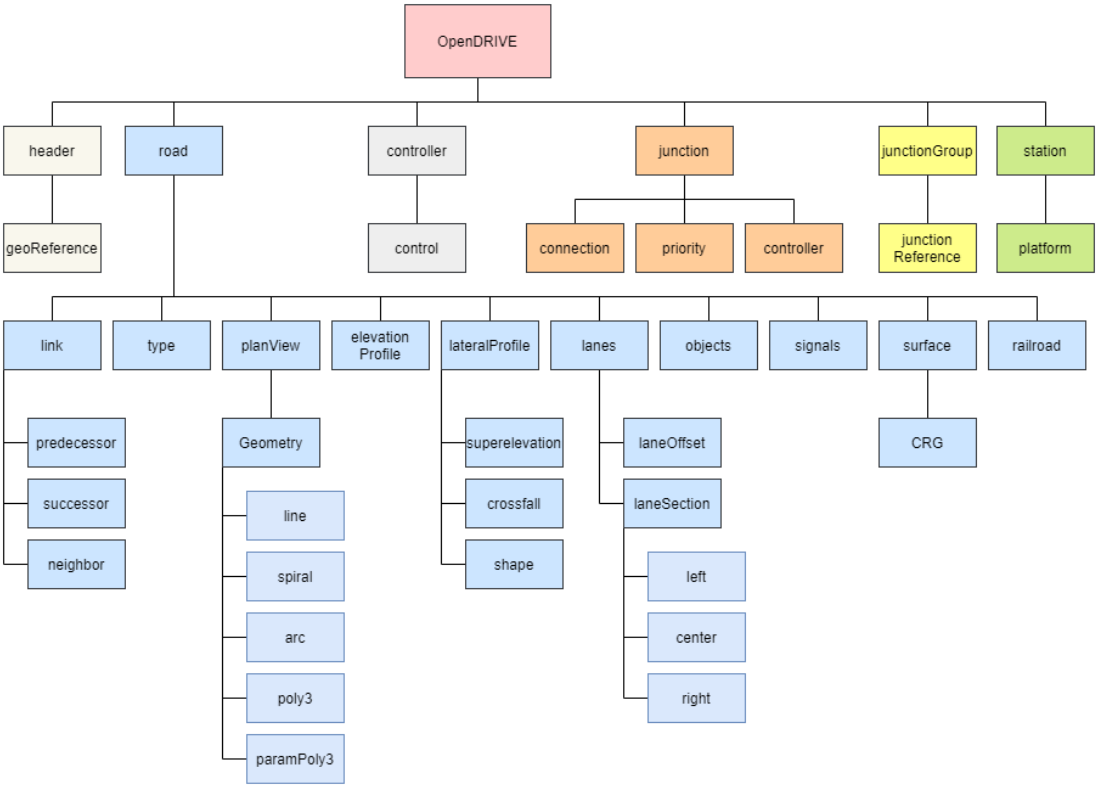


Figure 1. The bead structure of OpenDRIVE

The model consists of multiple building blocks which follow a predefined hierarchy. The six main components are header, road, controller, junction, junction group, and station. (Fig. 1)

The first element – the header – provides general details of the current model. These metadata can include the version of OpenDRIVE used, name of the database the data is from, name of the model, name of the company where the model was created, or geographic reference. These pieces of information are mainly for other developers or end-users as they have little effect on the simulator itself.

The vast majority of the valuable information can be found under the road tag, which is the section describing the individual roads, resulting in a fairly complex structure. It contains the reference line of the current road segment creating a local system in which the rest of the road – including the lanes, elevation, roadside objects – must be described. When it comes to the lanes, an ID number is assigned to all of them, which are unique within the road section. The ones on the right get negative, the ones on the left positive values. The closer a lane is to the reference line, the smaller the absolute value of said ID is.

OpenDRIVE is capable of handling a vast majority of different road types, which all have an effect on the simulation result. The most basic and most commonly used one is driving, which represents the pavement. Other types can include border, parking, curb, biking, sidewalk, etc.

In order to create a proper structure and connect the individual roads and lanes, links must be established between them. It enables the vehicle to travel from one section to another. To define the connections, each road and lane can have predecessors (previous item) and successors (following item).

The issue with such connections is the limitation of only being able to create them between two roads; therefore, intersections cannot be established without another building block, called a junction. It can replace the road tag as either predecessor or successor of any road. A separate block has to be created in the model for them, where the problem of overlapping lanes, called virtual lanes must be detailed. More complex junctions (roundabouts, for example) can be divided into multiple smaller ones and later organized into a junction group.

The remaining two units, controllers and stations, are not essential. The former can help to ensure the correct flow of traffic in junctions with various light settings and phases, while the latter marks the area where pedestrians get on and off public means of transportation.

The formal description of an OpenDRIVE model follows the same hierarchy as the model itself. Therefore, a suitable, hierarchic format was chosen in a form of a modified version of XML version, called XODR, short for XML OpenDRIVE.

3. FIELD SURVEYS AND PILOT SITES

In order to receive data for my software, BKK (Center for Budapest Transportation) was contacted as they are using various mobile mapping systems to gain up-to-date information about the city's road network. The most recent version of which is called KARESZ, short for Public Road Data Collection System (Fekete 2015). The georeferencing of this set of data happens afterwards, resulting in a colored point cloud with geographic information available. Various mapping systems – for example, RapidLasso, TerraSolid, and TopoDot – are used to process these few-billion-point data sets. The final step requires human operators not only to check the result of these automated processes, but they also help creating the suitable structures for later use. It involves on-screen visualization and lane tracking.

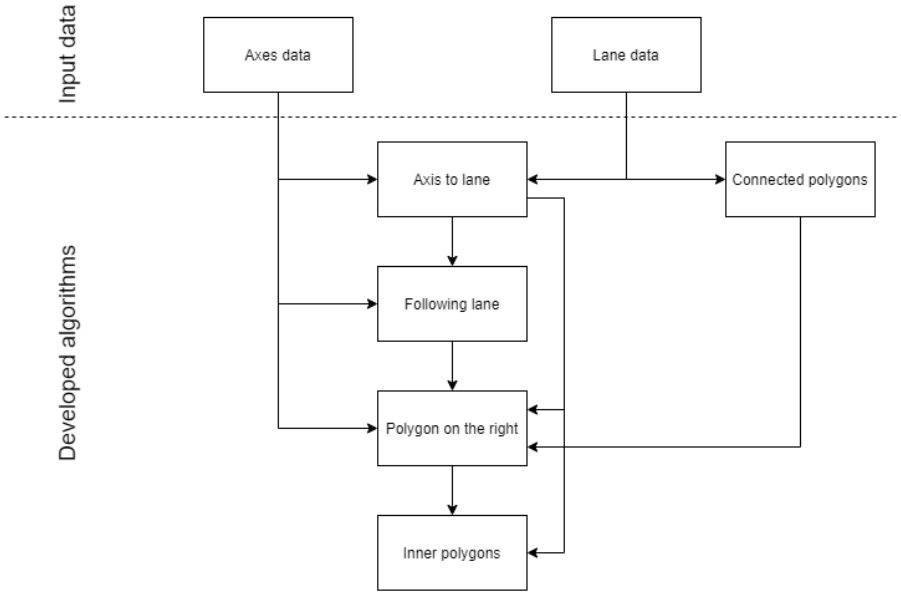


Figure 2. The developed workflow

The final product is in the form of a database, which can be easily updated when a new field survey is conducted in the same area. It can also be the beginning of another long process where digital maps are drawn using the processed data. As a result of these processes using human operators, numerous errors can be found in these files, meaning further software development involving them is complicated by the unknown nature of these mistakes.

Two pilot sites are particularly interesting as these were the ones my software was tested on. The first one is an urban area in the northern part of Budapest, along the main road No. 11. This circa 8-kilometer-long section poses many challenges. It has many lanes in both directions, with new ones opening and closing both on the inner and outer side. There are multiple intersections and a couple of bus stops as well. As it was detailed in Section 2, the format is capable of managing changes in the vertical component. To properly test this feature, a bridge over a suburban railway gave an excellent challenge to the overall planer region.

The other test site is a section of the M86 highway. The road is not only newly built, but the plans specifically required it to be suitable for self-driving cars. Therefore, it is frequently closed off to conduct such surveys and test drives for local and international companies and organizations.

Both data sets were created and provided by BKK, using the methods and techniques briefly described earlier.

4. SOFTWARE DEVELOPMENT

The first data set was received in the form of an AutoCAD file; therefore, a parser program was required to properly extract the valuable, relevant information from the exported plain text format of the drawing. It was written in C++ and is able to convert said file to another one without all the irrelevant pieces added automatically by AutoCAD. As it was mentioned previously, these input files were all created by human operators, which as a result contained numerous errors, which made it impossible to use the parser program effectively. These included the use of various drawing commands both with and without z coordinates, making later extension to the third dimension impossible.

After the problem being reported to BKK, a new set of data was provided in the form of three text files: one including information of all the polygons describing the lanes of the roads within the given pilot site, another one containing the axes of them point by point, and the third one was the reference line coordinates. All of these files were exported directly from the company's geodatabase, meaning that human intervention is even further reduced.

Our comprehensive analysis of the three sets of data resulted in the structure and outline of the required algorithms. The content of all of them follows the same pattern: ID of the object (polygon or line), ID of the point within that shape, and the coordinates (x, y, and z) of the point. Numerous details were found in the text files using multiple Matlab scripts, for instance: the polygons are closed, meaning that in the point-by-point description of the lane's polygons, the last point is the same as the first one; the axes of said polygons are almost perfectly connected, meaning a less than 3 centimeter difference between the last point of one and the first of the next axis.

The idea of using the provided reference line data was thrown out after the realization was made that the left side of the innermost lane can replace it perfectly. In fact, due to this line creating a local reference system, this way the road's description is a lot simpler. A dependency on human operators was eliminated, which further increased the accuracy of the final model.

The main algorithms were developed according to the following flowchart, using C++ for efficiency purposes. (Fig. 2)

The first function implemented was the one connecting axes and lanes. It was found by our analysis that both the first and last point of all the axes are exactly on an edge of a polygon, therefore if all lanes are checked for all the ends of all axes, the center lines can be found. Due to the model containing not only rectangle-shaped polygons but triangle-shaped ones (for example, bus stops or when a new lane opens) as well, the method needed a few minor corrections in a form of allowing either the first or the last – but not both – point of the axes to have the same coordinates as a point of a lane.

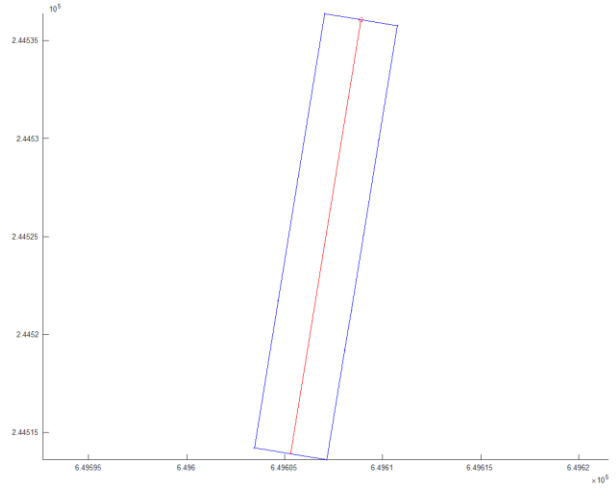


Figure 3. Lane-axis matching

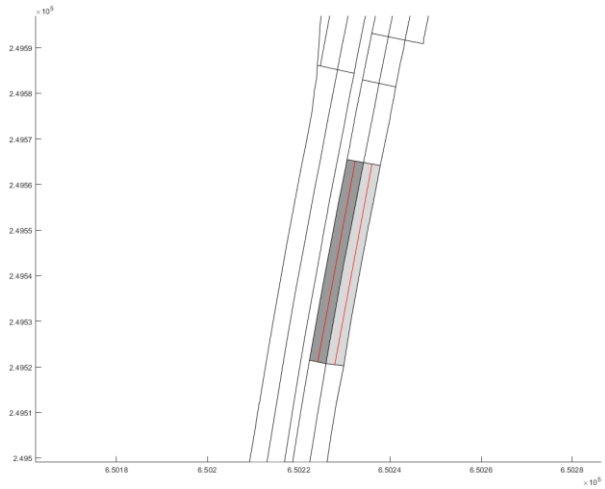


Figure 4. Innermost polygon and its right neighbor

In order to optimize the algorithm, a common strategy of using Minimal Closing Rectangles (MCR) was decided, meaning that only those axes are checked for a certain lane, which are inside the smallest possible rectangle fitting the polygon. This condition is faster to calculate; hence it greatly reduces the execution time of the code despite its time-complexity remaining the same. Furthermore, one axis cannot be used multiple times; therefore, checking it more than once is redundant and can be eliminated via noting which of them is already used. (Fig. 3)

The second algorithm is used for determining which polygons are connected. Despite it being a straightforward function, it is crucial in later methods. Given the fact that lanes are also perfectly connected without any gap, it can safely be said that two polygons are connected only if they share at least two common vertexes. For optimization purposes, the exact same MCR method was used.

The third main function is for finding predecessor and successor polygons for each lane if there are any. The basic concept is built on the prior realization that axes are perfectly connected to each other; hence if the last point of an axis is the same as the first one of another, the lane matching the former is continuing in the one matching the latter. It is possible that polygons have multiple predecessors and successors in junctions, for instance, however, the algorithm is capable of finding all of them.

As it was mentioned earlier, the left side of the innermost polygons is used as a reference line for the given road section; therefore, these lanes must be found. In order to do that, a similar method was implemented as in the previous function, namely the use of axes. It is significantly easier and much more effective to make various calculations with them. At first, a structure was created for each road segment, which where the right-side neighboring polygon was connected to each lane by utilizing basic third-dimensional distances and directions for the axes. From this point, determining the innermost ones were straight-forward as they are the lanes that are not on the right side of any of the polygons. (Fig. 4)

This function was a bit complicated by the fact that the input data contained triangle-shaped polygons, and in intersections, a lane can have more than one right-hand neighbor. To overcome this, a constant epsilon value was introduced, which served as a minimum distance between the axes first and last point, as well as allowing having many right-hand connected polygons. Due to the lanes being directed, this method works even when there is no median area between the two sides of the road.

After these crucial functions were implemented, the OpenDRIVE model was written using all the algorithms detailed above. To follow the guidelines and the hierarchy of the standard, a couple of extra calculations, data manipulation, and sorting was still required. These include measuring the width of the lanes at all the polygons' vertexes or finding the distance between these points and the reference line.

The models were built according to the 1.4 version of OpenDRIVE (OpenDRIVE 2021).

5. RESULTS

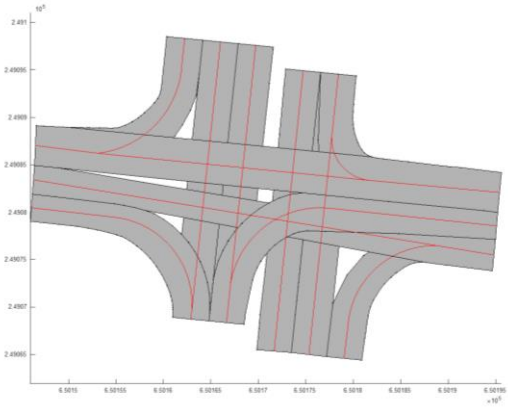
As a result of having two different sets of input data, two independent OpenDRIVE models were built. Both of them went through multiple simulators, which included the scenario-based IPG CarMaker (IPG CarMaker 2019) and the VTD Vires (Vires Virtual Test Drive 2019), where the latter was developed by the creators of the standard itself. (Fig. 6)

The first pilot site's (main road No. 11) input contained 757 polygons and 967 axes, meaning that a large portion of center lines was not used. The reason for this significant difference can be explained with the fact that these are stored in different database relations, and were exported by different human operators. The length of this particular road section is circa 8 kilometers. The total runtime of my software is under 2 minutes.

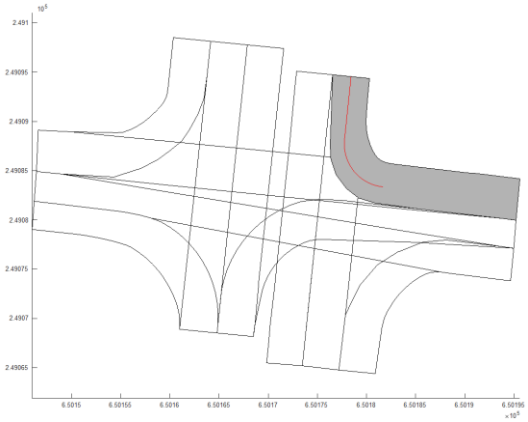
The second pilot site (M86 highway) consisted of 119 polygons and 121 axes. This set of data was created after our feedback based on the previous one was received; therefore, the number of errors was almost eliminated. Although the code was originally developed for the first set, it ran flawlessly on this 3.5-kilometer-long section as well, with an overall runtime of less than 20 seconds.

	No. 11 main road	**M86 highway**
Length of section	8 km	3.5 km
Number of polygons	757	119
Number of axes	967	121
Model building time	< 2 minutes	< 20 seconds

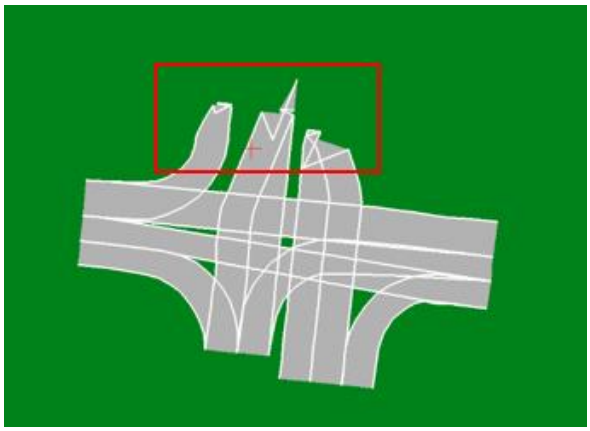
Table 1. Efficiency of the software



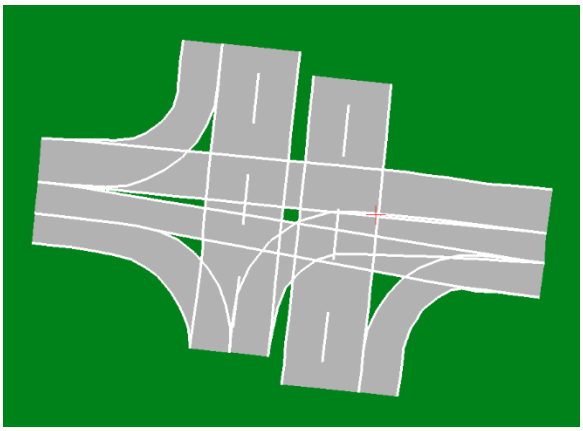
a) all data is seemingly good



b) missing complete lane axis for the turning lane



c) model created with all unnecessary points at the edges of the lane border



d) automatically corrected topology

Figure 5. Automatic topology check and correction

The developed software presented in the previous section imports the surveyed and preprocessed data: lane borders and axis lines. The algorithm performs the matchings but also checks the topology. Human errors were mentioned already: not only missing elements occur, but further hardly detectable errors burden the datasets: too dense placed border points, which come from digitizing or handling lane segments. Due to cutting, joining, and other lane segment manipulations, disturbing very close points enter the lane description. These unintended points could be avoided by fully human geometry construction tools, for example, by snapping. Because the preprocessing workflow had minimal human interactions, we can't rely on this beneficial tool and had implemented a basic topological correction by removing these disturbances. Fig. 5 illustrates a junction example, where diverse topological anomalies were detected, and our development has managed them.

Ignoring these anomalies, but following the prescription of the OpenDRIVE standard, severe structural and visual errors will be the output (Fig. 5c); therefore, they need to be corrected – possibly – automatically. The tolerances of the method are adjustable, so multiple levels of error correction are within the possibilities of the user. These values are flexible and can also be stored for future use, meaning that when the correct parameters of a sample area are found, these can be used for larger maps as well. As it can be seen, the designed and implemented correcting algorithm tremendously improves the overall quality of the final model in junction areas (Fig. 5d).

6. CONCLUSION

It is safe to say that such an algorithm was developed, which can have the potential to make a significant impact on creating a realistic simulation environment for self-driving cars by building high-definition road network models of reality. This was achieved via writing software in C++ purely due to its efficiency. The finished product follows the strict rules and hierarchy of the OpenDRIVE standard, however, it is far from being perfect.

Numerous extensions can be made to improve either the performance or the number of supported roadside elements; many of them are already planned. These include the cross or lateral profile of the road (for instance, superelevation) or traffic lights, signs, and signals, which only enhance the visual experience but don't have any impact on the simulation itself.

ACKNOWLEDGEMENT

I want to express my special thanks to BKK for providing the data sets for my research.

Figure 6. The finished model during simulation

REFERENCES

Apollo. (2020, Oktober 18). Retrieved from https://apollo.auto/index.html

Barsi, Á. (2019). Map production. Localization and mapping lecture notes. Budapest

Barsi, Á., Csepinszky, A., Gábor, Á., Krausz, N., Lógó, J. M., Potó, V., Varga, B. (2020). Evaluate and test physical map data formats. Budapest: Mobility Platform.

dSPACE (2019, September 26). Retrieved from https://www.dspace.com/en/pub/home.cfm

Fekete, G. (2015). Introduction to KARESZ based on 3D laser point cloud [in Hungarian]. KARESZ Conference. Budapest

IPG CarMaker (2019, September 26). Retrieved from https://ipg-automotive.com/products-services/simulation-software/carmaker/

KPMG. (2020, September 1). Retrieved from https://assets.kpmg/content/dam/kpmg/images/2015/05/connected-and-autonomous-vehicles.pdf

Mapping Ignorance. (2020, September 20). Retrieved from https://mappingignorance.org/2014/04/07/one-way-googles-cars-localize/

OpenDRIVE (2021, April 18) Retrieved from https://www.asam.net/standards/detail/opendrive/

PreScan | TASS International (2019, September 26). Retrieved from https://tass.plm.automation.siemens.com/prescan

RAND. (2020, September 2). Retrieved from https://www.rand.org/content/dam/rand/pubs/research_reports/RR400/RR443-2/RAND_RR443-2.pdf

SAE. (2020, September 2). Retrieved from https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles

Simulation Technologies – AVL (2019, September 26). Retrieved from https://www.avl.com/hu/web/guest/simulation

The Ohio State University. (2020, September 1). Retrieved from https://onlinemasters.ohio.edu/blog/the-future-of-driving/

USDOT. (2020, September 1). Retrieved from https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

Vires odrViewer (2019, September 26). https://vires.com/

Vires Virtual Test Drive (2019, September 26). Retrieved from https://vires.com/vtd-vires-virtual-test-drive/