doi:10.3311/BMEZalaZONE2022-003

Development of 3D Visualization for Displaying Real-time Sensor Information from Multiple Sources

Mátyás Szalai[1a], Viktor Remeli[1], Viktor Tihanyi[1]

[1] *Department of Automotive Technologies, Faculty of Transportation Engineering and Vehicle Engineering, Budapest University of Technology and Economics*

[a] *Corresponding author: szalai.matyas@kjk.bme.hu*

Abstract

For modern scientific results, it is increasingly challenging to present them in a form that the general public can understand. This is particularly true in the field of collaborative intelligent transportation services and connected vehicles, where many different platforms and sensors are operating simultaneously. It is important for users to build up the necessary trust in the new technology, which is why the presentation of the emergence and interaction of collective, platform- and sensor-level environment models is an important area. In this paper, a visualization environment that can present the different locally detected and centrally fused objects in real time are presented through an example of multiple infrastructure sensors. By statically and dynamically representing the digital twin, we can create a world where human users can see their environment through the eyes of individual vehicles, infrastructure stations, or the whole system. This, combined with Mixed Reality technology, can produce a result that feels real to the human eye and facilitates intuitive understanding of and trust towards the system.

Keywords: *Real-time, Sensor Detections, Sensoris, Unity3D, Visualization*

1 Introduction

Developments surrounding road vehicles could bring big changes to the transportation systems. In addition to the driving assistance systems that are now in almost all new cars, there is increasing talk of self-driving vehicles as a soon-to-be available option. Self-driving cars are surrounded by many questions from the general public. There are still several directions in which different industrial players are trying to reach the final goal, with different solutions for sensing, data processing, decision making and final implementation. Legal issues remain to be worked out, who is responsible for decision in self-driving vehicles, especially in case of accidents. Taken together, these issues continue to create a lack of trust in the average user [1], which is particularly true for the area of vehicle sensing. Knowing human vision, it is difficult to imagine how a vehicle sees, which is why it is important not only to be able to use machine vision, but also to present its operation in an appropriate form.

In the case of self-driving vehicles, people always talk about the vehicle's own perception system, but in intelligent transportation systems, the infrastructure itself can also play an active role. Multiple sensors mounted on roadside pillars provide greater coverage. By collecting their data on a single central server, a more complex digital twin is created [2]. Vehicles plugged into this digital twin can see much more information, with only a proper communication system (V2X). Referring to human vision, it is difficult for us to imagine seeing something from more than one direction, and the benefits of such a system are therefore harder to bring to the public's attention.

In this paper, we present a visualization system that can display sensor data from independent multiple sources, all positioned appropriately on static HD maps, creating an online digital twin. The platform created ensures that different systems can be managed in a single environment. Through a central system and Sensoris-based communication, the visualization module operates independently of the sensing system. Thus, we have come up with a solution that allows people to see the detections of the sensor systems on a large display or through AR headsets, projecting them into a digital twin of the real environment. This system can be used to illustrate the greater reliability of multi-sensor systems, the greater avoidability of outliers, and thus the robustness of the sensing. In the following sections, the interface between the data collecting central server and the Unity 3D visualization tool is presented, and the logic of the visualization and the content of the incoming data also will be explained. Finally, the complete visualization system presented in operation, through a real-time application.

2 Communication Interface for reading and sorting Sensoris data

To create a solution that can be used universally, it is necessary that the information have to be available in some standardized form. For this purpose, we use the SENSORIS standard [3], which provides an appropriate descriptive language for objects already detected by sensors and for the senders of detections . Messages from different sources are available on a central server, which can be accessed by any actor through a communication client module. In our case, the visualization only reads from the server, but additional information also can be sent back via the client module. The following subsections describe the C# interface that communicates with the client module, and how Sensoris data is transformed into a format that can be handled by Unity.

2.1 Communication Interface

Unity provides the possibility to use so called Managed plug-ins [4], which require compiling dll files. By choosing this solution, we can create a reusable C# interface that can also manage the packages used by Sensoris. The Google Protobuf and Grpc packages are required to manage the Sensoris messages. These can be added to the solution in the form of so-called NuGets. While for Visual Studio the NuGet manager is a built-in module, for Unity we can add the NuGet manager using a custom-package called NuGetForUnity [5].

Through the interface built against netstandard 2.0, we can retrieve the real-time messages of the last fusion step. A fusion step contains all the incoming messages that have been used to determine the output of the fusion. In this case, only the most recent fusion step can be received. This guarantees that the visualization is running in real-time, the only delay is the delay of the entire system.

The IAsyncStreamReader class of Grpc.Core [6] is used to access the stream. The method call has two parameters, the name of the monitored topic and a bool parameter to allow or disallow receiving older messages. During real-time playback, the party using the interface receives a list of Sensoris datamessages. This list reflects the latest possible state after each call. However, this also means that, depending on the frequencies of the receiving and transmitting, it is possible that a step may be read more than once, or steps may be missed between two calls. To deal with this, the frequency of visualization recommended to be higher than the frequency of sensor fusion. The messages received through the interface therefore contain a list of elements of type Sensoris DataMessage. To manage their contents, additional operations are required, now within Unity.

2.2 Classification of Sensoris data into Unity classes

To simplify the handling of the information, the datamessages for a given fusion step are mapped one by one, according to a unique class for parsing Sensoris messages. This SensorisParser class is responsible for determining the sender of a given datamessage, and for sorting the detections associated by their type. The Sensoris specification contains several static and dynamic detection categories, the system can currently handle pedestrian and vehicle types. Objects are created for both the sender and the detection, which are populated with different parameters based on the Sensoris description. In the case of the sender, the Origin class contains the sender's ID, position, and rotation. In the case of detections, a Detection-type object is created, which contains the detection's own identifier, its position and rotation, as well as its 3D size, the probability of its existence and the identifier of its sender.

For the determination of the positions and rotations, it is important to find some reference point between the real and virtual worlds to create a digital twin. In our case this is a pre-selected null point, which we use as the origin of the whole system. The local position, interpreted in Unity's own coordinate system, is given by the UTM position relative to this null point. In addition, it was important to convert the incoming data expressed in millimeters to meter format and to handle the differences between the coordinate systems. Unity's own coordinate system differs from the coordinate system used in the automotive industry, while Unity's coordinate system is left-handed, and Y-Up type, the coordinate system of the automotive industry is right-handed, and Z-Up type [7]. This means that in addition to the offset and reverse rotation directions, the y and z axes are also swapped. As in the case of positions, in all other cases the conversion of the data to metric must be done so that the data has the correct type for Unity.

3 The logic of the visualization

The representation of the objects detected by the sensors should be interpreted as a function of time, where a packet of messages received from the server represents a specific fusion time snapshot. The received message packets, which are sorted into DataMessage type lists, can display different states. Since message reads are assigned to fusion steps, the last message packet is always associated with the fusion, and only one packet is guaranteed to be read. For the other sensor stations, the structure of the Sensoris topic, called "fusion-utm" guarantee that maximum one packet belongs to each sender. Depending on the frequency of the fusion and each sender, the Sensoris topic will always store only the most recent message for each sender, and if the frequency of a sender is lower, there may be no message part for that sender. The possible contents of messages as a function of time are illustrated in Fig. 1.

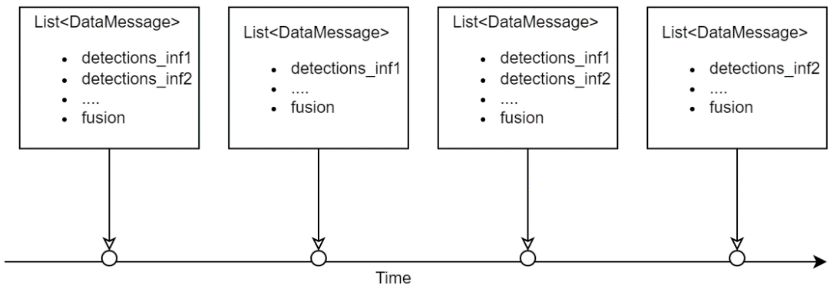

Fig. 1 *Content of the DataMessage Lists over time*

The visualization is always triggered by new incoming information, the visualized detections are moved when new information about their state is received. Each list is processed DataMessage by DataMessage, and based on their content, new detections may appear, old ones may disappear, or existing ones may be moved to another position. Fig. 2 illustrates how each data packet contains detections and the corresponding sender, called Origin.

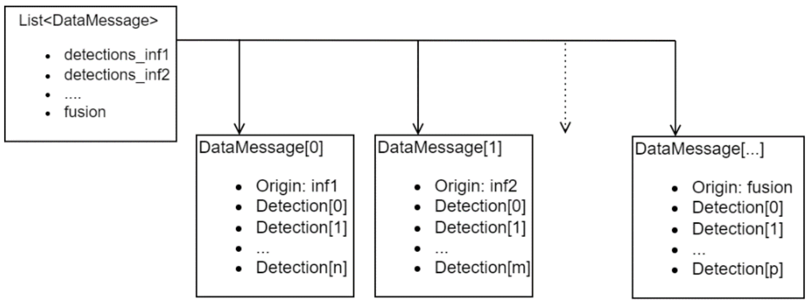

Fig. 2 *DataMessage list's elements and their content by sender*

A DataMessage can be divided into two main parts in terms of detections. The message always contains information about its sender and a list of objects detected by the sender, with their parameters. They are processed as described in chapter 2.2. Once the data is available in the Unity-defined classes, the next step is its visualization. The first visualization step is a simple task since each object must be represented in the virtual space according to its content. The next visualization step must act based on the objects that already exist. To ensure that only detected objects that exist at a given moment are displayed, all previous objects belonging to the sender of a given DataMessage are firstly deleted from the Scene, and then new detections are displayed according to the new message. This step also solves the problem if a particular sender can serve data at a lower frequency than the fusion. In this case, the detections associated with that sender will remain on the screen until a new message is received from that source. However, this solution raises the question of what happens to the detections if there are no more messages from that source, i.e., the sender has left the system for some reason. This has been solved by automatically deleting all detections 500 milliseconds after they have been displayed if they have not been deleted by another call.

An important area of visualization is to create virtual reality as a digital twin of reality. HD maps are becoming more and more familiar [8] and can easily be used to create a static digital twin of the real environment. These maps include their position in the Earth coordinate system, which we use in UTM format. The various sensor stations also know their own positions, and in turn the positions of all the objects they detect. The resulting UTM positions can then be used to place the detections exactly where the real objects are in the real world. However, we cannot store the whole world in Unity, so only a small part of it is implemented. On this map, we choose an origin, whose exact position are known, and then all the other objects are displayed relative to it. Unity provides a tool, using local position and rotation parameters, that always returns the position of an object relative to its parent. Therefore, both senders and detections are defined as children under the HD map, and their position is given by the local parameters.

To summarize, the visualization receives incoming data packets bound to fusion steps and then handles them separately per source. After deleting the previous messages for a given source, new detections are generated according to the new content. Meanwhile, the source, i.e., the Origin object, is also represented in its own position, represented by a sphere in the case of a sensor station, and by a 3D model of the source in the case of a vehicle.

4 Real-time application of the visualization system

The implementation of a system's logic is complete when it works in reality. The system developed has been demonstrated on several occasions. In this chapter, this system will be presented.

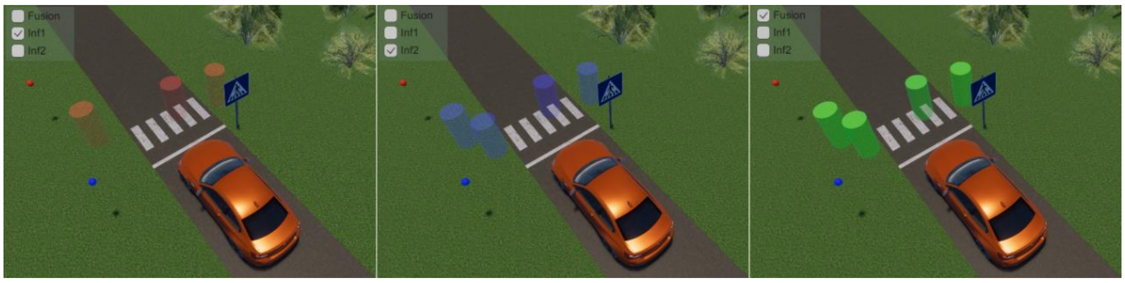

Fig. 3 *Visualized detections from the two stations and the fusion*

In the tested system, two real sensor stations sent their detections to the central system. These detections, as well as those generated by the fusion, are displayed in the visualization. In addition, a simulated vehicle was added to the scenario, which stopped when pedestrians entered the crosswalk. In Fig. 3, the different sensor stations are marked in different colors. Infrastructure sensor station 1 is marked in red, infrastructure sensor station 2 in blue, while detections from fusion are marked in green. It can also be seen that small spheres in the images indicate the exact location of the stations. It can also be observed that in the image for station 1 only 3 detection cylinders are visible, which is due to the occlusion. However, with the two stations, this object is also visible, it is included in the fusion detections that are considered to be most robust and reliable.

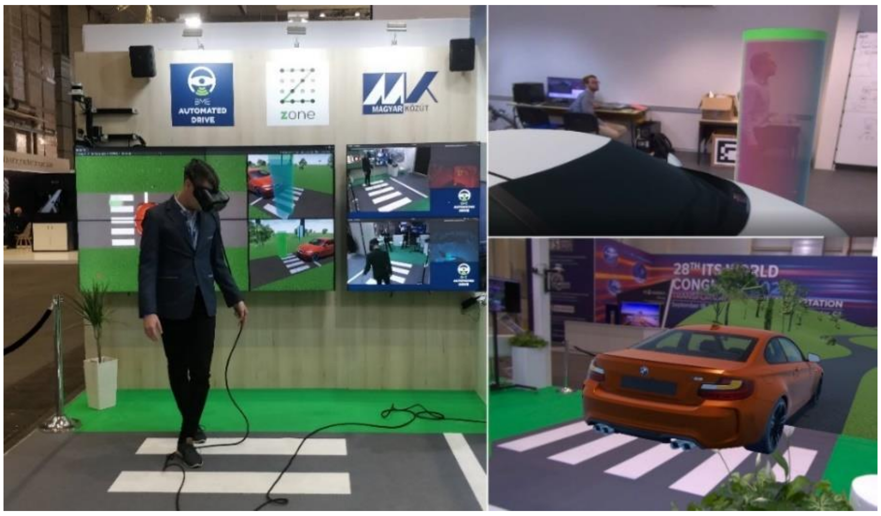

Fig. 4 *The demonstrated system and the AR headset with the mixed reality views*

The system presented was an indoor scenario with a single road and crossing. However, this small system is sufficient to demonstrate how the system works. Using Augmented Reality technology, the viewer entering the scenario can become a part of the virtual world. Using the Varjo XR-1 headset, the visitor can see everything just as in reality, combined with virtually generated and projected objects to create mixed reality. Fig. 4 shows a part of the demonstrated system, with one sensor station mounted on the left, a visualization of the virtual world on the large displays, and a tester with AR glasses. The images on the right try to show mixed reality, the image above shows a detection and a virtual vehicle during testing in a laboratory room, and the image below shows the virtual vehicle driving onto a virtual road, exiting the reality.

5 Conclusion

In summary, with the developed solution, it is possible to visualize the already processed data of sensors. The advantage of the system is that it can be easily used in conjunction with augmented reality technology and does not depend on any licensed software or drivers. The client module makes the visualization completely independent of the sensors. The solution used allows detections from multiple sources to be managed in a single environment. Just as the detection of pedestrians has been demonstrated, the visualization of detected vehicles is also implemented, with multiple tasks from the 3D size and orientation point of view. However, the biggest advantage of the system is that it helps the general public understand how collaborative environment perception systems

work, thanks to augmented reality. A possible future development is to extend the list of visualizable types of detections. An important task is to optimize the system and ensure that it can be used in a more compact way. We would like to develop a system that can be deployed easily and quickly at all the ZalaZONE proving ground sites [9], without the need for engineering work to start a demonstration.

Acknowledgement

The research reported in this paper and carried out at the Budapest University of Technology and Economics has been supported by the National Research Development and Innovation Fund (TKP2020 National Challenges Subprogram, Grant No. BME-NC) based on the charter of bolster issued by the National Research Development and Innovation Office under the auspices of the Ministry for Innovation and Technology.

References

[1] C. Olaverri-Monreal, “Promoting trust in self-driving vehicles,” Nat. Electron., vol. 3, no. 6, pp. 292–294, Jun. 2020, doi: 10.1038/s41928-020-0434-8.

[2] Zs. Szalay, V. Tihanyi, A. Rövid, V. Remeli, Zs. Vincze, M. Csonthó, Zs. Pethő, M. Szalai, B. Varga, A. Khalil, “Towards Cooperative Perception Services for ITS: Digital Twin in the Automotive Edge Cloud,” Energies, vol. 14, no. 18, p. 5930, 2021, doi: 10.3390/en14185930.

[3] “SENSORIS - SENSOR INTERFACE SPECIFICATION.” https://sensoris.org (accessed Oct. 15, 2021).

[4] “Managed plug-ins,” Unity Documentation. https://docs.unity3d.com/Manual/UsingDLL.html (accessed Sep. 20, 2021).

[5] “GlitchEnzo/NuGetForUnity,” GitHub. https://github.com/GlitchEnzo/NuGetForUnity (accessed Aug. 23, 2021).

[6] “Grpc.Core,” NuGet.org. https://www.nuget.org/packages/Grpc.Core (accessed Sep. 16, 2021).

[7] “A Guide to Unity’s Coordinate System (With Practical Examples),” techarthub. https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/ (accessed Jan. 28, 2022).

[8] L. Bao, Q. Wang, and Y. Jiang, “Review of Digital twin for intelligent transportation system,” in 2021 International Conference on Information Control, Electrical Engineering and Rail Transit (ICEERT), Lanzhou, China, Oct. 2021, pp. 309–315. doi: 10.1109/ICEERT53919.2021.00064.

[9] Zs. Szalay, Z. Hamar, and P. Simon, “A Multi-layer Autonomous Vehicle and Simulation Validation Ecosystem Axis: ZalaZONE,” in Intelligent Autonomous Systems 15, vol. 867, M. Strand, R. Dillmann, E. Menegatti, and S. Ghidoni, Eds. Cham: Springer International Publishing, 2019, pp. 954–963. doi: 10.1007/978-3-030-01370-7_74.