AILiveSim: An Extensible Virtual Environment for Training Autonomous Vehicles

Jérôme Leudet
AILiveSim Oy
Helsinki, Finland
jerome.leudet@ailivesim.com

François Christophe
Department of Computer Science
University of Helsinki
Helsinki, Finland
francois.christophe@helsinki.fi

Tommi Mikkonen
Department of Computer Science
University of Helsinki
Helsinki, Finland
tommi.mikkonen@helsinki.fi

Tomi Männistö
Department of Computer Science
University of Helsinki
Helsinki, Finland
tomi.mannisto@helsinki.fi

***Abstract*—Virtualization technologies have become commonplace both in software development as well as engineering in a more general sense. Using virtualization offers other benefits than simulation and testing as a virtual environment can often be more liberally configured than the corresponding physical environment. This, in turn, introduces new possibilities for education and training, including both for humans and artificial intelligence (AI). To this end, we are developing a simulation platform AILiveSim. The platform is built on top of the Unreal Engine[1] game development system, and it is dedicated to training and testing autonomous systems, their sensors and their algorithms in a simulated environment. In this paper, we describe the elements that we have built on top of the engine to realize a Virtual Environment (VE) useful for the design, implementation, application and analysis of autonomous systems. We present the architecture that we have put in place to transform our simulation platform from automotive specific to be domain agnostic and support two new domains of applications: autonomous ships and autonomous mining machines. We describe the important specificity of each domain in regard to simulation. In addition, we also report the challenges encountered when simulating those applications, and the decisions taken to overcome these challenges.**

***Index Terms*—Virtual Environment, Autonomous Systems, Autonomous Vehicles, Simulation, Training, Validation.**

I. Introduction

Developing software for autonomous systems can rely on fully integrated end-to-end solutions [5], [28] or more a engineered software stack [3], [4]. However, in both approaches the algorithms controlling these systems rely on sensors for perception and controllers for actuation. The autonomous functions of systems are often based on machine learning algorithms. Training and testing them takes a huge amount of data. The exact amounts varies of course[2] and popular datasets: 15k images [9], appoloscape 140k image [13], bdd100K 1.2M images [29], can be used as a starting point, but most manufacturer will want to customize them to their own sensors, ground truth and requirements. Collecting and maintaining this data is difficult and time consuming. It takes up to 79% of data scientists time, and 99.9% of the data collected in real life is redundant[3]. Testing autonomous systems thoroughly is no easy task either, partly because a good enough test case is impossible to define. Even with real-size mock-ups – such as Google's fake city using professional pedestrians [2] – there are yet many real-life cases that are simply impossible to reproduce during training. There are interesting deployments of agents in virtual environments used for automating testing [27]. On the domain of driving, similar environments are also used for motion planning and training cars for autonomous driving [19].

In this paper, we present AILiveSim, a simulation platform intended for development, training, testing, and validating of autonomous systems. In the advent of disruption of autonomous mobility and AI in various industries, AILiveSim provides a tool for the developers that makes the journey to autonomy faster and more reliable. AILiveSim provides a parametric simulation environment that lets developers gather data virtually through simulations, together with annotations or metadata that can be used for supervised learning, or to measure the performances of the systems. AILiveSim helps deep learning, reinforcement learning and testing of autonomous systems. AILiveSim enables (i) building prototypes and verify concepts; (ii) creating data sets to train AI systems; (iii) debugging and optimizing algorithms; and (iv) testing and validating products.

Using virtualization also offers other benefits, as a virtual environment can often be more liberally configured than the corresponding physical environment. This, in turn, introduces new possibilities for education and training, including both humans and artificial intelligence (AI). While the results may not be enough for completely bypassing training in real world,

[1]https://www.unrealengine.com

[2]https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/

[3]https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#3f58ed7f6f63

virtual reality can offer learning experiences configured to match exact learning needs and real-world corner cases that almost never happen. Furthermore, the environment has been architected so that it can be easily enriched with new features and subsystems that can be plugged into the system in a straightforward fashion.

The rest of this paper is structured as follows. In Section 2 we provide the background and the motivation for this paper. In Section 3, we introduce our prototype design and implementation, including its key interfaces and design decisions. In addition, we provide sample applications, created for a diversity of contexts such as maritime, automotive or ground mining. In Section 4, we discuss some design challenges with the system's architecture that we had to solve when applying our software to new domains. In Section 5, we provide an extended discussion regarding lessons learned during the development of our environment. In Section 6, we present some directions for future work. Finally, in Section 7, we draw some final conclusions.

II. Background

Developing a complete and good enough test set for autonomous system is no easy task. Standards like ISO 26262 [17] present some of the core principles, and establish that tests should be integrated at all stages of the development. To this end, developers of autonomous vehicles have started to integrate the principles of continuous software engineering [8] in their development cycles. Tesla, for example, delivers software updates over-the-air[4]. However in many cases, the integration has turned out to be much more complex than originally envisioned. For instance, in [22], the authors conducted a study in which they interviewed developers to understand how continuous integration of their changes could be improved. This study shows that the main factors include, in order of importance:

- speed, including in particular tools and processes that are fast and easy to use;
- availability of the test environment;
- confidence through test activities;
- reliability of the test environment.

In our previous study [18], we provided an overview of how VEs are used and how they could be beneficial for different machine learning (ML) tasks. We also showed how having access to the precise state of the world – in simulation – gives a significant advantage over conducting the same tests in real life, when considering measuring or scoring the performance of algorithms or systems.

Testing and pre-training autonomous vehicles in a Virtual Environment (VE) as such is not a very novel idea. The concept was originally introduced in [9]. Today, multiple other VEs exist [6], [10], [23] that cover several applications, including for instance control, reinforcement learning, and situation testing. However, these environments are still very focused on the automotive context. Our claim in this paper is that VEs can be applied in the development of any kind of Autonomous Systems. To this end, our platform has been in used in various domains, including maritime, mining, and automotive. This versatile background has helped us to integrate the necessary components to the AILiveSim platform and to interface it with various maritime applications already in use or in current development, including MonaLisa project [20], AIS [25], and STM validation [21], to name some examples.

Each of these domains have their special characteristics – developing autonomous ships, for instance, presents somewhat different challenges than those of an autonomous car. In the context of ships, considering deep sea navigation, the relative scarcity of fixed points in the environment, and because of the distances being large, the types of sensors and the problems to solve in general are different from the automotive context. Also, when considering harbour conditions, where traffic can be busy with a high number of ships of different sizes, the development of an autonomous ship poses very different issues than with cars due to parameters such as vessel's own inertia and awareness of every other actors position, speed and direction within the ships area of reach [24]. Therefore, sensors dedicated to marine environments [26] have different specifications and requirements than those for automotive [17]. In the development of autonomous ships, various radars will be used to detect objects at different ranges, positioning will rely on the Global Positioning System (GPS) coupled with Geographic Information System (GIS), or satellite imaging. LiDARs are fairly new sensors and they are used for close range navigation [12]. Cameras are also making their way to the autonomous ships [14]. Measurements from these sensors need to be compensated for roll and pitch of the boat when at sea, which is not necessarily an easy task as these parameters change according to weather conditions. Weather conditions such as fog, rain, or frost, also have other impacts on sensors. Precise sensor fusion is necessary to make sense of constant orientation of the ship but also get a clear description of objects in the environment regardless of weather conditions. For this purpose, a common practice is to use redundant sensors that have different strengths and weaknesses in order to crosscheck relevant information from incorrect information caused by these changing conditions. In contrast to roads where things can happen in a fraction of a second, in the maritime environment things are much slower, ships are relatively slow and with long braking distances due to their inertia. Each situation covers large areas and for this reason we might need to accelerate the simulation speed and concentrate only on the essential moments.

In mining environments, the fundamental application-specific needs and the set of sensors available are again different. The GPS is not available underground, and the very low light conditions will have an impact on the sensors that can be equipped to the mining machines. The machines can be loaded with heavy loads affecting their dynamic models. The tunnels can be narrow allowing for very little imprecision in control commands. Also at a planning level, the problems are very different than for self driving cars. In the mines,

[4]https://www.tesla.com/support/software-updates

there are custom traffic rules, and priorities can traditionally be agreed upon and communicated through radio, or even a central recommendation can be used for planning.

III. Prototype Implementation

The purpose of our prototype, named AILiveSim, is to act as a demonstrator of how virtual reality can be used as an environment where the training autonomous vehicles can take place. The first use case – training artificial intelligence for self-driving cars – is reported in [18], together with some preliminary lessons learned. Focusing on two new use cases that were developed using AILiveSim for customers from the maritime and the mining industrial domains, this section summarizes the key learnings and design principles that were implemented to generalize the platform to support the two new new application domains.

A. Requirements

Based on our learnings in [18], we extracted the following architecturally significant requirements for our design:

- to be able to simulate any amount of sensors in real time;
- to produce simulated sensor data that is good enough to be used to augment existing datasets;
- to be able to accelerate simulations;
- to have the ability to collect ground truth annotations together with data;
- to use parameters to artificially alter the quality of data in a realistic way (e.g. simulating stains, lens distortions grain, or bad pixels);
- to support multiple domains of applications, covering vehicles in a general sense, and covering cars, trucks, cranes, boats, and drones;
- to enable the creation of use cases for different algorithms, such as machine vision, control, and reinforcement learning.

Furthermore, we wanted to design AILiveSim as an open platform, so that it can be used for training various other types of vehicles as well in the future. Such openness was one of the key architectural design drivers when composing this prototype. To this end, additional components that introduce specialized functions can be added to the virtual world as plugin extensions. Therefore, it is also possible to plug in not only sensors that detect virtual items in the virtual reality, similarly to their real-world counterparts, but also application specific components, like controllers or trackers. These can even interface with external hardware or proprietary customer architectures, which allows the inclusion of drones, ships, and other autonomous vehicles in the animation. As an example, Fig. 1 presents three sample applications, where a car following a lane in traffic (top), a maritime vessel is sailing under configurable conditions (middle), and two mining machines in a collision avoidance scenario in a cramped mining environment (bottom). Obviously, there could be numerous 'real' vehicles included in the same virtual environment, if there are enough computational resources, or the other vehicles could be fully simulated by the platform.

Fig. 1. AILiveSim in action. Top: A car autonomously following a line in traffic. Middle: A maritime scenario with different sea conditions and ships. Bottom: Avoiding a collision in a mine.

B. Design

In the technical sense, the AILiveSim platform is built on top of the Unreal Engine [7], a suite of integrated tools for game developers to design and build games, simulations, and visualizations. This way the design efficiently leverages pre-existing technology and benefits from updates that will keep the performance level high as the technology evolves. Fig. 2 presents the AILiveSim architecture with the specific components that had to be developed, in addition to those that we could directly use from the Unreal Engine.

More specificially, we created blueprints, use modes, environments and sensors (marked in blue in Fig. 2) that are built within the Unreal platform. These internal components allow us to control the elements of the scene, the logic, the properties of the simulations etc. AILiveSim exposes a simple API that allows the users to control the simulation, send commands to the vehicles, extract sensor feedback or other information about the state of the world. The AILiveSim controller is handling sessions, caches, and configurations related to client applications. AILiveSim was developed to interface with a wide range of applications such as autonomous vehicle training, intelligent harbour and autonomous ship simulation, and a framework for automated tests of a maritime vessel sailing in configurable conditions near a coastline. In the mining application, the open architecture allowed us to easily create two different versions of the vehicles dynamics and two sets of sensors that allowed us to easily support two very different use cases with the same assets.

As AILiveSim is an open and extendable platform by design, custom features specific to a new domain can be integrated in the system as plugins, including also extending key parts of the system itself. We have developed components inside the Unreal Engine to support its specific features and we also have developed content in specific ways to maximize the amount of changes and parameters that can be exposed. AILiveSim also offers a library of common sensors that are parametric, allowing users to tweak them to obtain data that is similar to what they are getting with their real sensors. The platform also simulates cameras, LiDARs, GPS, and other sensors at a reasonable level of fidelity so that they can be used together with data collected in real conditions. Furthermore, custom sensor code, or different application-specific logic and behaviors can be integrated in the system as well using the plugin mechanism.

C. Developing Custom Applications

Due to the openness of the design, the system can be easily adjusted to different application domains, requiring different functions. In the following, we discuss how AILiveSim was gradually extended through two customer projects from maritime and mining industry domains.

Extending AILiveSim for Autonomous ships. We have created a maritime environment in AILiveSim. We created a model for the sea, waves and coast line – we automatically dress a natural landscape based on height maps of real locations – and parametric harbours that allow for many situations. We already mentioned some of the specificity of the maritime environment in Section II. The main challenge in this project was to realize simulation acceleration. Indeed, ship movements and manoeuvres happen relatively slowly and, to be able to test rapidly the validity of actions within a long process, we needed to implement a fast-forward capacity.

Extending AILiveSim for the mining industry. We have created a mining environment and a few different mining machines. In our case, the vehicles are mostly relying on LiDARs and laser based proximity sensors. In this case, Lidar data is used by an awareness system to establish a map of navigable areas. In turn this map is used by other systems like path planning or other higher level systems. Those two can be isolated and developed separately. We can generate LiDar data and test the awareness system, and we can generate the map of navigable areas based on the meta data we have in the simulation to test higher level systems. In Fig. 3, we display both the simulation of the real data and the simulation of the abstracted data that a subsystem would otherwise generate. That data takes the form of a grid that indicates navigable areas. In turns this data can be used to improve the performance of the awareness system. This information can also be fed to the other systems. As a matter of fact, we found that simulating the Lidar data is much more expensive than simulating the more abstract map, and moreover that the map data is much smaller (1kb vs 122kb per frame for Lidar) so it is easier to manage and to transfer.

IV. Experiences and Design Challenges

In this paper, we report the real life issues that we encountered when we wanted our existing platform, AILiveSim, to support new domains of application. We grouped the major ones into two categories: (i) challenges related to the integration into the virtual world and the Unreal engine underlying simulation; and (ii) execution challenges related to performance of the simulation.

A. Integration challenges

There are several types of integration challenges that we faced when composing AILiveSim. These include integrating sensors in the virtual environment as well as integrating our design with available libraries and open source components. In the following, these are discussed in more detail.

1) Integration of sensors in the environment: The implementation of the simulation of sensors varies with each sensors. So far, we have been working with three different sensor categories – cameras, and LiDAR and radar sensors.

Cameras. Images that are generated by default were not close to the images we got out of real sensors, so we had to add custom parametric elements to allow more realism, and less perfect images. Luckily in Unreal Engine, the image generation and render pipeline is rather good. There are plenty of parameters, and you can use a stack of post-process shaders that can access various buffers to create custom effects, distortions to the images or ground truth measurements. In our platform, we have developed an interface to configure a stack

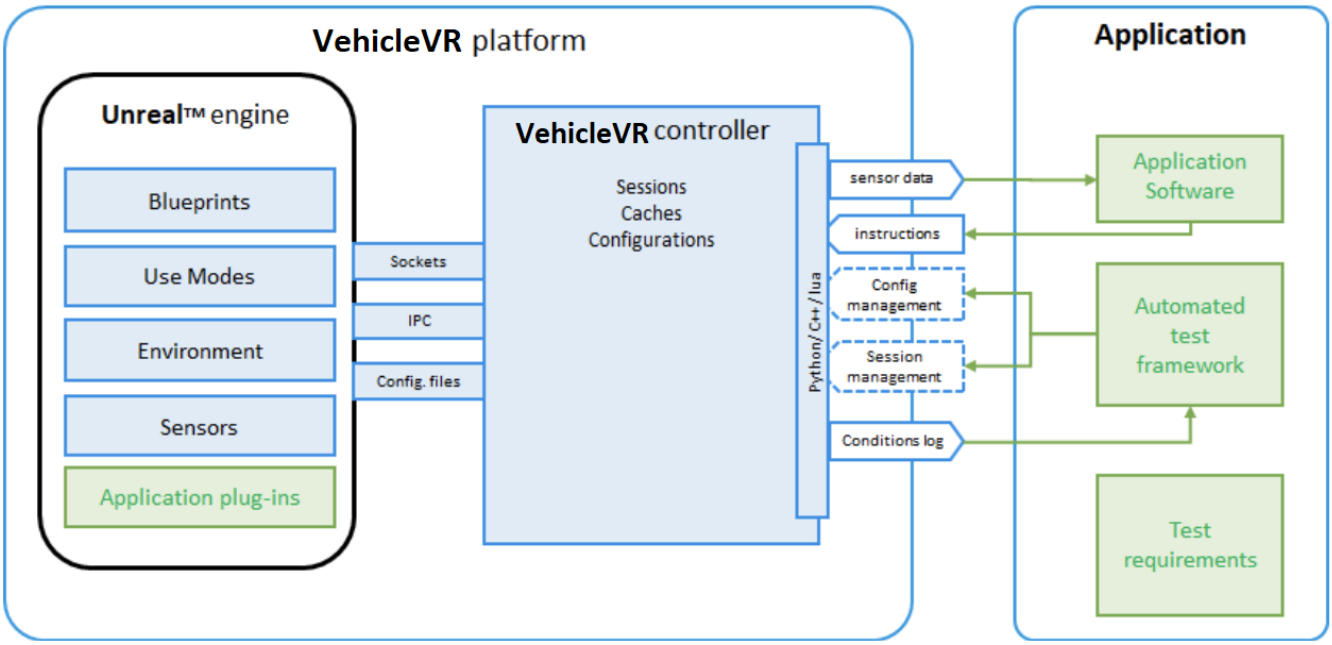

Fig. 2. A simplified view of the parts provided by AILiveSim (presented in blue), combined with the components of an integrated example application to perform automated testing (represented in green).

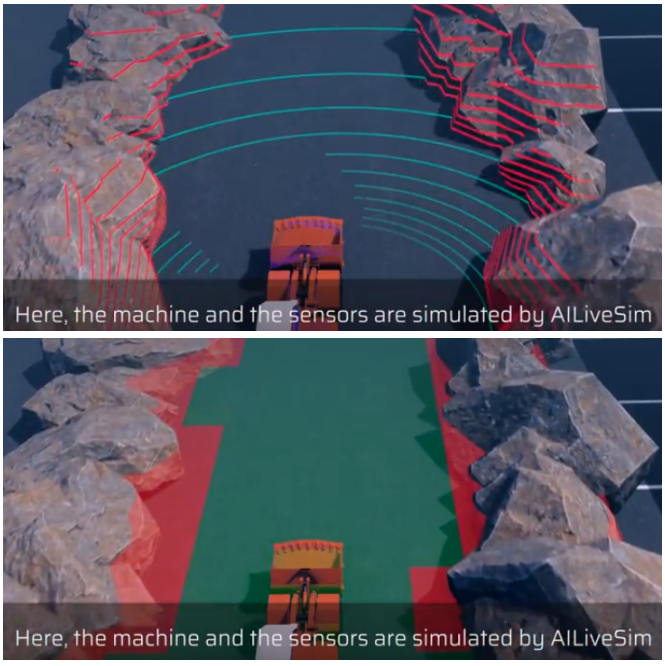

Fig. 3. Simulation of a VLP16 LiDar, with segmentation of the data using custom rules (top), and the simulation of abstracted data that another system would otherwise generate based on LiDar data (below).

of pre-made post-process effects that can be applied on top of each other to compose different effects and their strength. That allows users to control the amount and the nature of the noise that they want their systems to support. Fig. 4 presents the range of effects that can be applied directly from our platform.

LiDAR Sensors. The LiDAR sensors need to be simulated based on reflected light. There are two different options: (i) Simulate the LiDAR on the rendering using shaders [1]; or (ii) Simulate the LiDAR on the physics using traces [11]. When simulating the LiDAR on physics data, the sensor will only be exposed to a simplified version of the world that is meant to be handled by the physics engine. That means that the collision meta-data will need to have higher quality in order to serve also the needs of the LiDAR sensor. When approximating objects with simpler shapes, details about the sharpness of edges might be abstracted away. Those issues are not present when using the shader approach to simulate the LiDAR data, as the shaders will have access to information embedded into the materials. For every triangle, the materials will convey information about the nominal and other parameters (roughness, metallic, specular) for each point within the triangle. When this information is taken into account in the calculation of the reflected light, we can get much more precise results than when using only the triangle.

Radar sensors. Radars are harder to simulate accurately than other sensors. Also, different radars have different levels of abstraction for low level data. Like with the physics, the approach here is to approximate the shapes of various objects with simpler geometries for which radar cross sections are well known. The radar sensor would only have to work on those simple objects instead of working on the full environment.

2) Integration of libraries: Another area of integration challenges is introduced due to using libraries and other functions that can be used to extend the platform. Unreal engine has a handy plugin architecture, which allows third

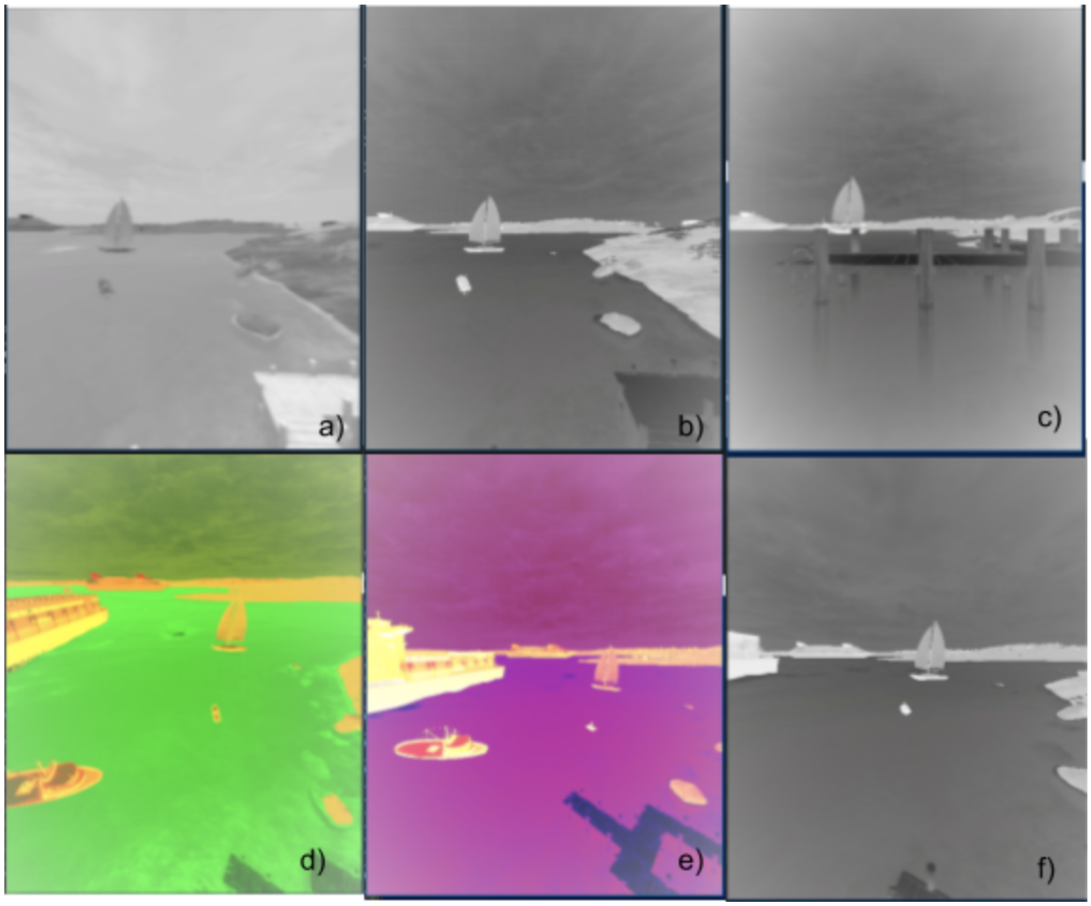

Fig. 4. The AILiveSim platform offers various shaders that can be stacked, in order to reproduce traits or imperfections of images from real sensors. f) is the baseline thermal camera shader, to which we add: a) Blur effect, b) Grain lines effect, c) Heat bleed+lens effect, d) Thermal image RYG effect, e) Thermal image WRB effect.

parties to extend the functionalities by providing actors, actor components or other unreal specific objects. To this end, our platform provides specific Blueprint interfaces for sensor developers to implement and integrate with AILiveSim. For example, if one would like to develop a specific sensor, he would implement the *ALS_Sensor_Actor* interface. That would allow the sensor to be configurable in the configuration file just like any sensor provided by the platform, attached to the actor, and to expose the data collected in a streamlined fashion. Finally, some vendors have developed proprietary, commercial algorithms to simulate sensor data. We have performed technical feasibility studies regarding their integration in AILiveSim, but no licensing negotiations have been carried out at this point.

3) Integration with Open-Source software: One of the benefits of using the Unreal Engine is the community of talented people susceptible to create open source projects, solving specific simulation problems better than many others. If a customer developing a plugin is willing to make it available as open source, he might get others interested to contribute too. To protect business interests and related intellectual property rights, it is possible to decouple the algorithms and the interfacing with the simulation, and make the algorithm part available is open source with a suitable licence. However, as with other third party libraries, no negotiations with other vendors have taken place.

B. Simulation execution challenges

Currently, when using the AILiveSim platform, every sensor is simulated locally in the simulator. That can cause performance issues especially when the computer running the simulation is not powerful enough. Slow-downs in the simulation will cause the data to arrive slower than real time, which is harmful for obvious reasons. So far, we have considered two potential solutions to this potential performance problem: distribution and acceleration.

1) Distribution: To improve the performance of the simulation, we have considered distributing the sensor simulation to several computers. At least all the camera sensors are easy to distribute with the Unreal Engine. The engine proposes a client-server architecture that is very well suited to this kind of situation: the server executes the physics once, and every client has local approximations of the state of the world. In case of network delays, the difference might become too large between the client and the server, in which case some

rollback operations might be needed, which may cause partial invalidation of past sensor data. Therefore, all sensors that have really high coupling with the physics or overall high frequency might need to be executed on the server to avoid correcting wrong predictions. Such high coupling is a limiting factor for distribution.

2) Acceleration: Another issue to consider is the simulation speed. In the Unreal Engine, there are two main threads where things can happen, the logic tick and the physics tick. It is possible to create faster threads or events, but they can only access such data that is not currently processed by the engine, or there will be a serious performance hit. The physics thread is where all the physical simulation actually happens. It controls the movement of all objects in the world. In AILiveSim, this thread is configured to run at least at 100Hz (Fig. 5), and the Unreal Engine can use a variable time step time below this upper bound, called sub-stepping in the Unreal Engine. The other important tick is the logic tick, called update in the Unreal Engine. The logic tick will execute more computing-intensive operations, like monitoring, decision-making, reaction to events and general logic-related functions. The logic tick happens at much lower frequencies than the physics tick, and the therefore the logic tick is also the preferred place to handle communications with the outside world.

Fig. 5 presents a visualization of how decisions and executions are handled in AILiveSim. We need to execute the routines in the correct threads to guarantee a smooth execution even at higher speeds. When accelerating the simulation, we generally need to accelerate the physics thread while the logic thread might not speed up quite as much. The physics thread needs to stay as lightweight as possible but execute all the routines that need to control the movements and forces every time. However reaction to events or planning can be run at slower pace. For example, when controlling a ship, the decision on the trajectory changes infrequently, but its application can be implemented with low level controllers like PIDs that are lightweight and benefit to run every physics update to keep smooth controls. In AILiveSim, the users can control vehicles remotely from their machine while the simulation is executed on a distant server. The delay incurred by the network becomes a problem when executing the simulation much faster than real-time. For that reason, the decoupling of decisions and controls is very important. Low level execution and control can be executed locally on the server, while higher level decisions can be executed remotely and at lower frequency.

V. Lessons learned

To begin with, the Unreal Engine is a very powerful game engine that offers solid possibilities for making simulations of and for serious applications. However, there are certain limitations and constraints that must be taken into consideration when developing them. In particular, the Unreal Engine is designed for real-time applications. Therefore, it will prioritize the main rendering loop at the cost of physical simulation when running short on resources. The consequences of setting the priorities this way only really matter if the implications are incompatible with the requirements associated with the subject we are simulating. In the following we will cover some of the limitations we have encountered. In addition, we present the solutions we have chosen as well as a brief evaluation of the impact that these decisions have on the model in comparison with reality.

A. Precision of the model against real-life simulations

It is clear that the simulation will use models for the environment and the vehicles, meaning that we use a collection of techniques that produce a result that is "close enough" to the reality. Defining "close enough" really depends on the requirements that are set to applications.

One concrete example is the wake trail that follows a boat and propagates small waves behind it. Every boat that is moving will create waves on the water surface that is caused by the displacement of bodies of water along with the shell of the boat. In our case, we decided that the fluid simulation was not critical to get the overall behaviour of the boat correctly. To control autonomous ships, we figured that it was enough to have a higher level of abstraction.

However, for our camera sensors the wake sometimes plays a big role in identifying the boat on the sea, since the boat itself might be much smaller than the trail it leaves behind. For that reason we have decided to focus on having limited visual elements for our cameras as shown in Fig. 6, but that will not have any physical impact on the boats.

B. Communication between controller and simulator

It was a requirement of AILiveSim to be able to accelerate the simulation. The biggest performance bottlenecks are currently associated with the volumes of sensor data that need to be generated and transmitted to the client algorithm, as listed in the following:

- 2 Mp (megapixel) camera, FullHD at 30 frames per second generates around 33 Mbps (megabytes per seconds) uncompressed. In AILiveSim, we offer Portable Network Graphics (PNG) compression that reduces the number down to 29 Mbps.
- LiDAR like VLP16 rotating at 10 rpm generates around 2.7 Mbps data. Adding layers, rotation speed, or point density can produce considerably more data, up to 20–70 Mbps.
- Sonar will result in about 10–100 Kbps.
- GPS will result in about 50 Kbps.

Typical setups include several cameras and LiDARs on one single vehicle, creating large amount of data to be generated and transmitted. Depending on the network configuration, and particularly if the simulation is executed on a remote server on the cloud, there are chances for packet loss.

Two protocols were considered for this communication, TCP and UDP (Transmission Control Protocol [16] and User Datagram Protocol, [15], respectively). The TCP protocol is impacted by the latency of the network in case of packet loss while the UDP protocol is not. Initially, TCP was used

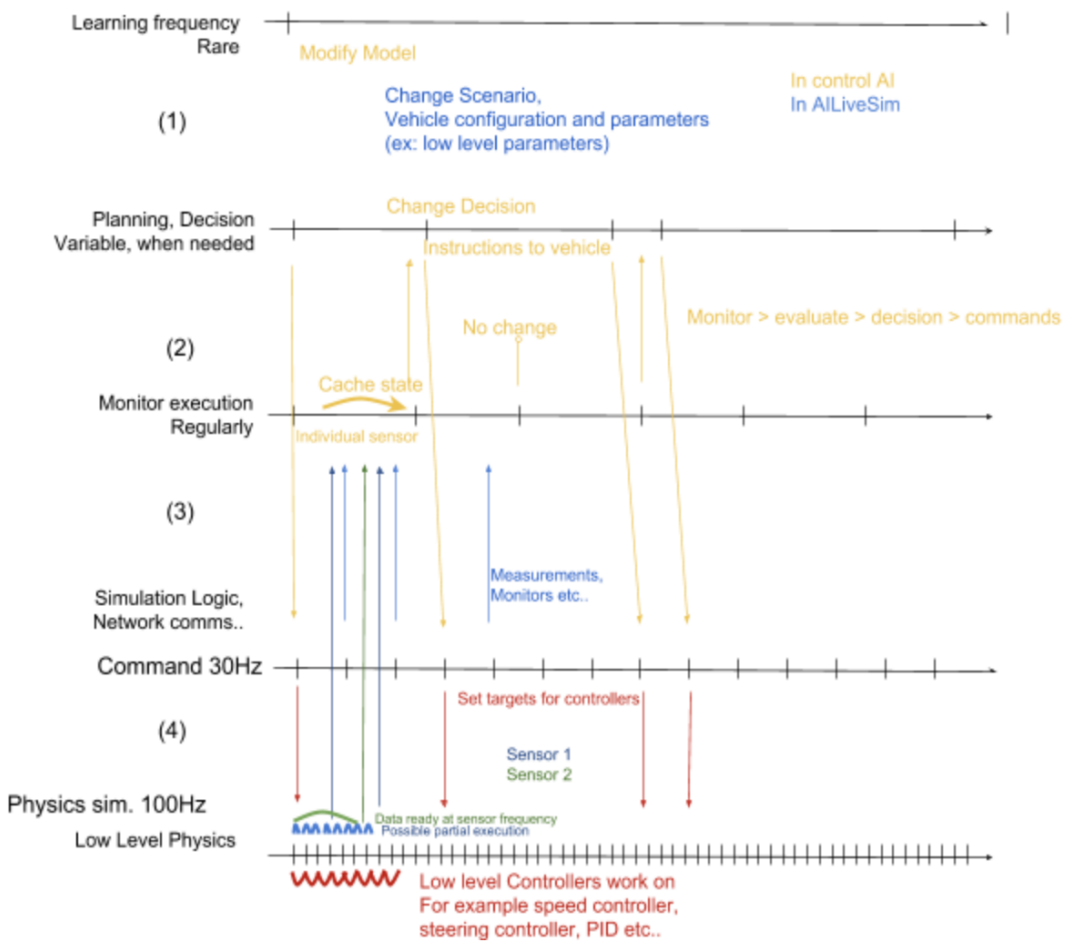

Fig. 5. Graph visualization of the decoupling of the decision and execution to maximize the precision even when accelerating the simulation.

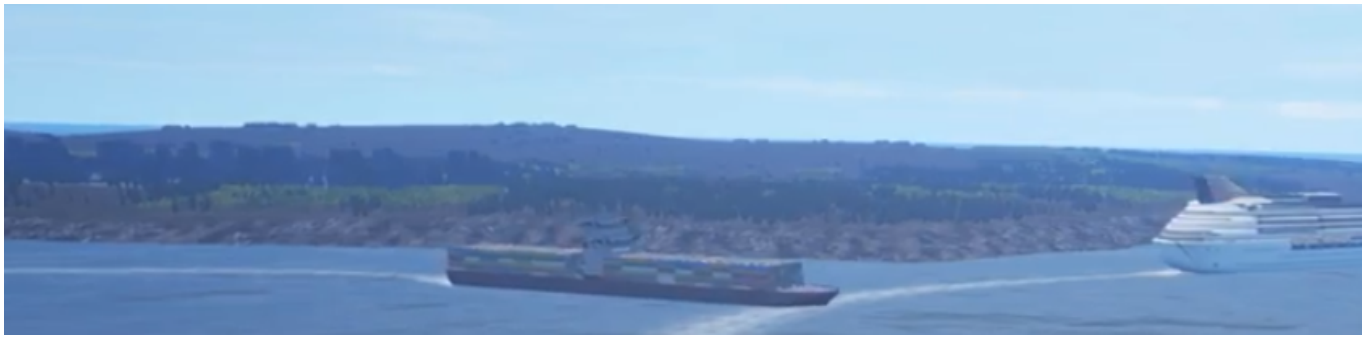

Fig. 6. Visual simulation of the wake trails behind boats.

for communication between sensors and client algorithms. The protocol offers guarantees on the transmission of data. However, loss of throughput could become considerable in some cases. To manage this, we decided to use UDP for transferring sensor data issued from simulations. UDP offers less guarantees on the transmission of packets, but it is much faster to transfer large amount of data using UDP than TCP. However, TCP is still used for commands and controls sent from the controller to the simulator, in order to keep guarantee of message integrity.

VI. Future Work

Our long term vision is to demonstrate value of training autonomous systems in virtual environments. While our work is still at an early stage, we believe that the following topics will take us closer to the eventual goal.

Distribute the simulation of the sensors to several machines. Here, the goal is to partition the simulation of the state of the world and the simulation of each sensor to different

machines. Such design introduces the the challenge to keep those in sync so that the sensor readings that are simulated match to the state of the world at a given time.

Distribute the simulations of larger amount of scenario on the cloud. To run through a large number of test cases at a reasonable time, we can distribute test execution to the cloud. The challenge with such approach is to find an easy, secure, and private way to make the controlling algorithm to the cloud so that it can be invoked together with each instance of the simulation. As one of our requirements is to make no assumptions on the client algorithms, the coupling has to remain loose.

Improve the quality of the simulation of the sensor data, by adding defects that are observed in real data. In the case of data-set creation or for training various low-level algorithms like segmentation or detection, we need to be able to generate data that has similar characteristics as the real sensors. To solve such issues, we would like to try various approach and compare them.

Automatic reconstitution in AILiveSim of situations that are similar to some samples provided as input. Our objective is be to create a network that could generation simulation parameters that minimize differences, given one or several sensor inputs. We believe that this would be an interesting problem to solve, as it would make it easier to create specific situations.

VII. Conclusion

Virtual reality has become a viable option that offers various opportunities for building autonomous vehicles of the future in a more agile fashion than relying solely on physical vehicles. However, designing such environments in an extendable and modular way is not straightforward. Such development requires combining knowledge from various domains.

We have developed AILiveSim for training autonomous vehicles in virtual environments. We want it to serve several applications, from testing, awareness, learning, and we are well aware that techniques associated with these activities are constantly evolving. Also we believe that the domain of application is just a context and content, and there is no need for building domain-specific system for every domain. Therefore, AILiveSim can serve numerous applications in different domains, as demonstrated in this paper by the introduction of two new application domains, each with specific requirements.

During the development, we faced numerous challenges, many of which we have been able to solve with well-established engineering practices and support provided by the Unreal Engine. These include the design of a plugin system for extensions to the platform and variance associated with the different domains and their characteristics. The remaining problems, mostly associated with performance, are an important direction for our future work. We wish to study the benefits that the AILiveSim platform can bring to specific algorithms, like reinforcement learning or model predictive controls with models of real vehicles and reals sensors.

References

[1] Gregory D Abram and Turner Whitted. Building block shaders. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 283–288. ACM, 1990.

[2] Mark Austin. Google built an entire fake city to test the AI of its driverless cars. *Digital Trends, Aug. 27, 2017.*, Available at https://www.digitaltrends.com/cars/google-fake-city/.

[3] Sagar Behere. *Reference architectures for highly automated driving*. PhD thesis, KTH Royal Institute of Technology, 2016.

[4] Karl Berntorp, Tru Hoang, Rien Quirynen, and Stefano Di Cairano. Control architecture design for autonomous vehicles. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 404–411. IEEE, 2018.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[7] Epic Games. Unreal Engine. *Online: https://www. unrealengine. com*, 2007.

[8] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[9] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[10] Cristian Gorgorin, Victor Gradinescu, Raluca Diaconescu, Valentin Cristea, and Liviu Iftode. An integrated vehicular and network simulator for vehicular ad-hoc networks. In *Proceedings of the 20th European Simulation and Modelling Conference*, volume 59, 2006.

[11] Carl Gutwin. Traces: Visualizing the immediate past to support group interaction. In *Graphics interface*, pages 43–50, 2002.

[12] Marko Höyhtyä, Jyrki Huusko, Markku Kiviranta, Kenneth Solberg, and Juha Rokka. Connectivity for autonomous ships: Architecture, use cases, and research challenges. In *Information and Communication Technology Convergence (ICTC), 2017 International Conference on*, pages 345–350. IEEE, 2017.

[13] Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application, 2018.

[14] Terry Huntsberger, Hrand Aghazarian, Andrew Howard, and David C Trotz. Stereo vision–based navigation for autonomous surface vessels. *Journal of Field Robotics*, 28(1):3–18, 2011.

[15] IETF RFC 768. User Datagram Protocol. 1980.

[16] IETF RFC 793. Transmission Control Protocol. 1981.

[17] ISO26262. Road Vehicles – Functional Safety. *International Standard ISO/FDIS*, 26262, 2011.

[18] Jerome Leudet, Tommi Mikkonen, François Christophe, and Tomi Männistö. Virtual environment for training autonomous vehicles. In *Annual Conference Towards Autonomous Robotic Systems*, pages 159–169. Springer, 2018.

[19] Liyun Li. Iterative em planning: A flexible motion planning platform for autonomous driving on urban roads. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, pages 374–379. IEEE, 2018.

[20] Mikael Lind, Anders Brödje, Richard Watson, Sandra Haraldson, PE Holmberg, and M Hägg. Digital infrastructures for enabling sea traffic management. In *The 10th International Symposium ISIS*, 2014.

[21] Mikael Lind, Mikael Hägg, Ulf Siwe, and Sandra Haraldson. Sea traffic management–beneficial for all maritime stakeholders. *Transportation Research Procedia*, 14:183–192, 2016.

[22] T. Mrtensson, D. Sthl, and J. Bosch. Continuous integration impediments in large-scale industry projects. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 169–178, April 2017.

[23] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[24] Thomas Statheros, Gareth Howells, and Klaus McDonald Maier. Autonomous ship collision avoidance navigation concepts, technologies and techniques. *The Journal of Navigation*, 61(1):129–142, 2008.

[25] Enmei Tu, Guanghao Zhang, Lily Rachmawati, Eshan Rajabally, and Guang-Bin Huang. Exploiting ais data for intelligent maritime navigation: a comprehensive survey from data to methodology. *IEEE Transactions on Intelligent Transportation Systems*, 19(5):1559–1582, 2018.

[26] UCG Commandant. International regulations for prevention of collisions at sea, 1972 (72 colregs). *International Standard ISO/FDIS*, 26262, 2011.

[27] Shufeng Wang and Hong Zhu. Catest: a test automation framework for multi-agent systems. In *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*, pages 148–157. IEEE, 2012.

[28] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception, 2018.

[29] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling, 2018.