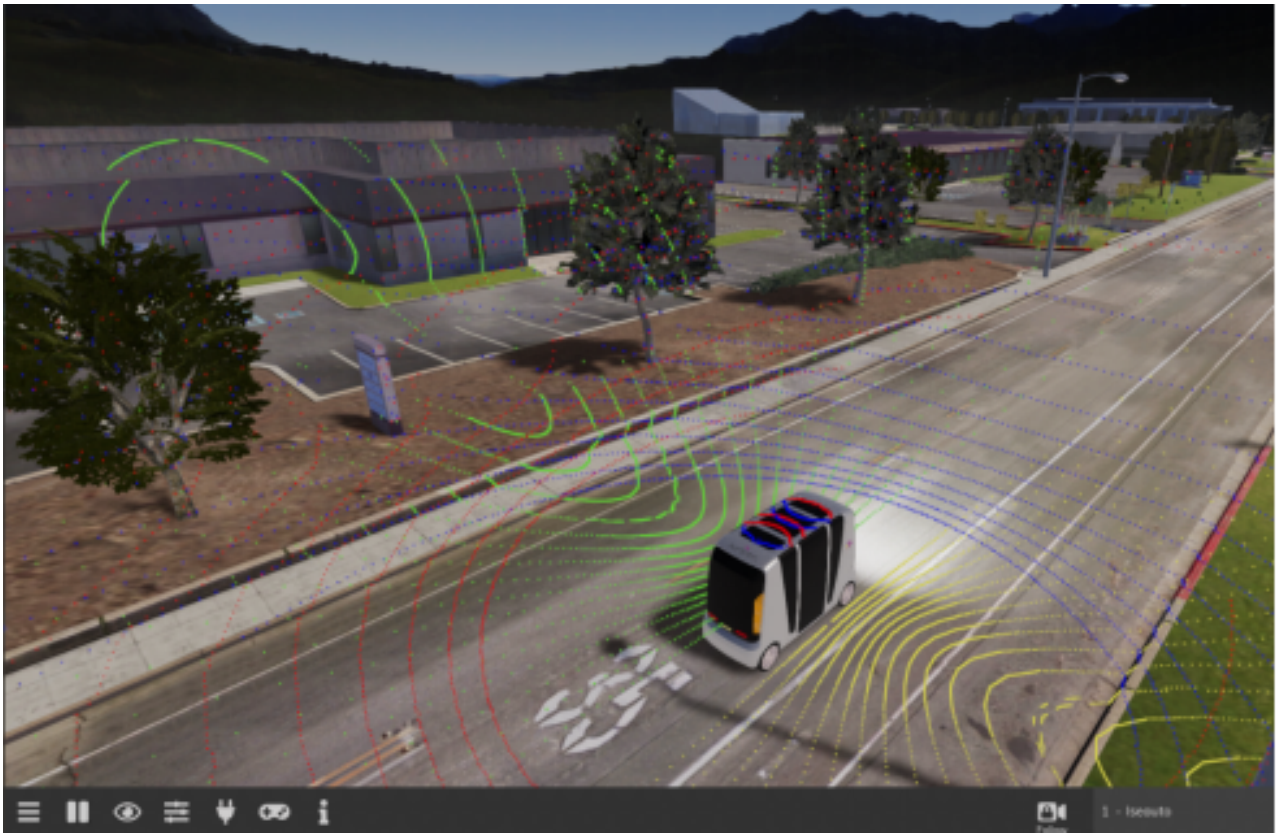
PolyVerif validation report



Validation team:
Team leader: Raivo Sell
Digital twin and mapping: Baris Cem Baykara
Simulations and scenarios: Mohsen Malayjerdi
Field experiments: Ehsan Malayjerdi
Reporting and recommendations: Ingmar Sell

Tallinn 2021-2022

Table of Content

Intro

The validation group tested and validated the PolyVerif framework by using a unique open-source full scale AV shuttle.

The validation will be carried out in two steps.
Step 1. Example scenario and virtual environment model creation of selected areas in the city of Tallinn. The validation area will be in a mixed environment where people are moving around and not always using the correct pedestrian crossing to cross the driveway. Low traffic is present including self-driving shuttle.

Step 2. Real-life validation connected to self-driving AV shuttle iseAuto. Developed scenarios and PolyVerif framework is validated with real equipment on the real traffic.

The work was conducted in four stages as follows:
S1: Digital Twin Route in Estonia
S2: Validation scenarios (2-3) in PolyVerif
S3: Scenario Validation with Physical Device
S4: Detailed Report of Methodology for use by US Services

This report describes all stages and results of simulations and experiments. It also includes recommendations on how to proceed with the PolyVerif framework based on the current situation (Jan-Feb 2022) and the methodology to create a digital twin of the specific area.

Operational Design Domain (ODD)

ODD defines operating conditions under which a given driving automated vehicle thereof is specifically designed to function.
The experiment ODD is defined as follows:

Domain	**Parameter**	**Value**
Weather	Precipitation	Light rain or snow, no ice rain or hail
	Temperature	-10 to +15 deg C
	Wind	0 to light wind
Geographical	Area1	TalTech campus
	Area2	Tallinn harbour
Time	Season	Autumn, winter (in Estonia)
	Time-of-day	Daylight, no direct sun
Traffic	Road	Asphalt, two lane, two directions
	Traffic	Light traffic, car approaching to intersection from both lanes
	Pedestrians	Randomly crossing roads on the marked pedestrian crossings

	Traffic signs	Smart pedestrian crossing, regular pedestrian crossing
Vehicle	Type	4 seat experimental AV shuttle - iseAuto #3
	Speed	7-10 km/h
	Software	Autoware.ai

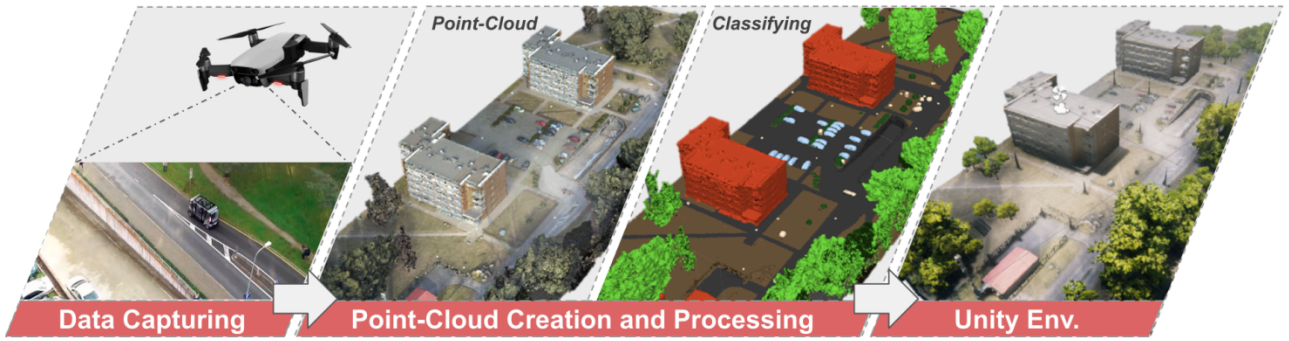
The simulations and experiments conducted in this validation work are following ODD, defined in the table above. Depending on the particular experiment day, real conditions may slightly differ and may not include all parameters from defined ODD. However, the main parameters are not exceeded.

Stage 1 - Digital Twin of the areas

Digital Twin is a virtual copy of a real subject. In this case creation of a digital twin consists of **Model Abstraction** and **Characterization**.

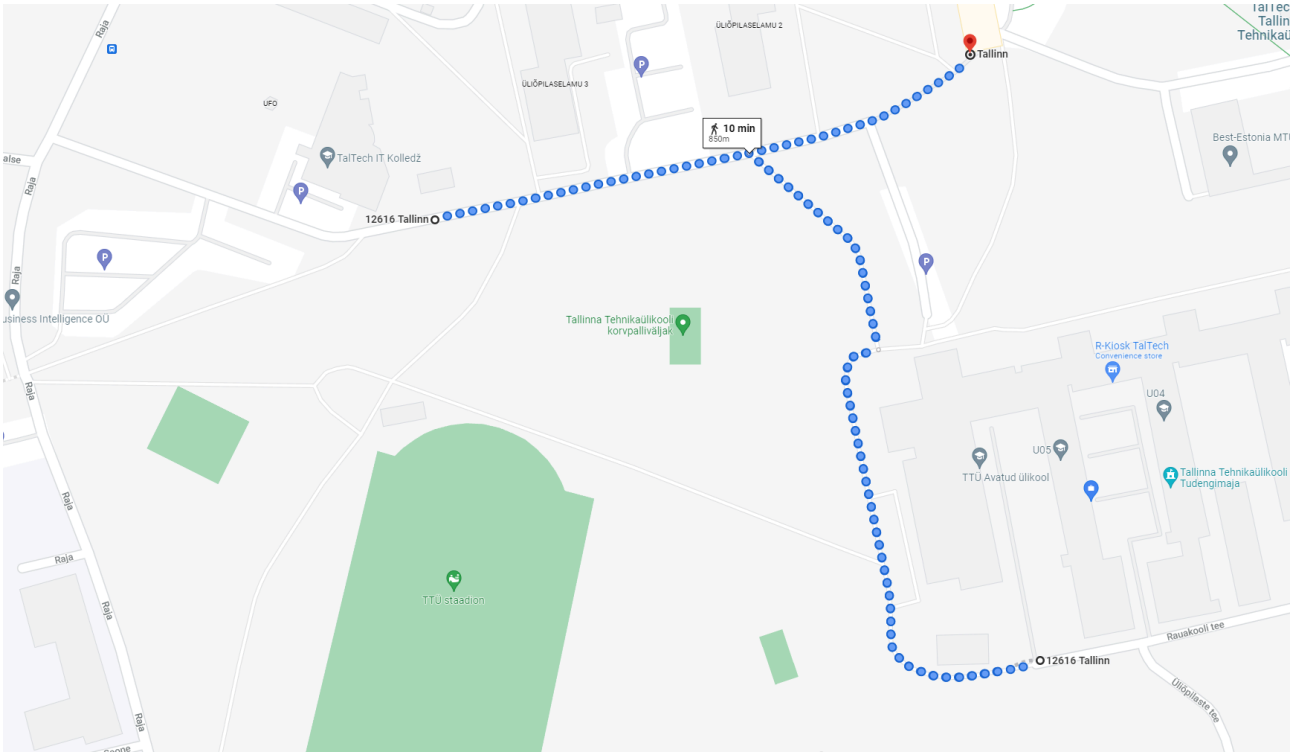
Digital Twin creation from the area of interest is divided to the following activities:

1. Aerial image capture
2. Lidar mapping
3. Post-processing of raw data
4. Classification
5. Virtual environment creation

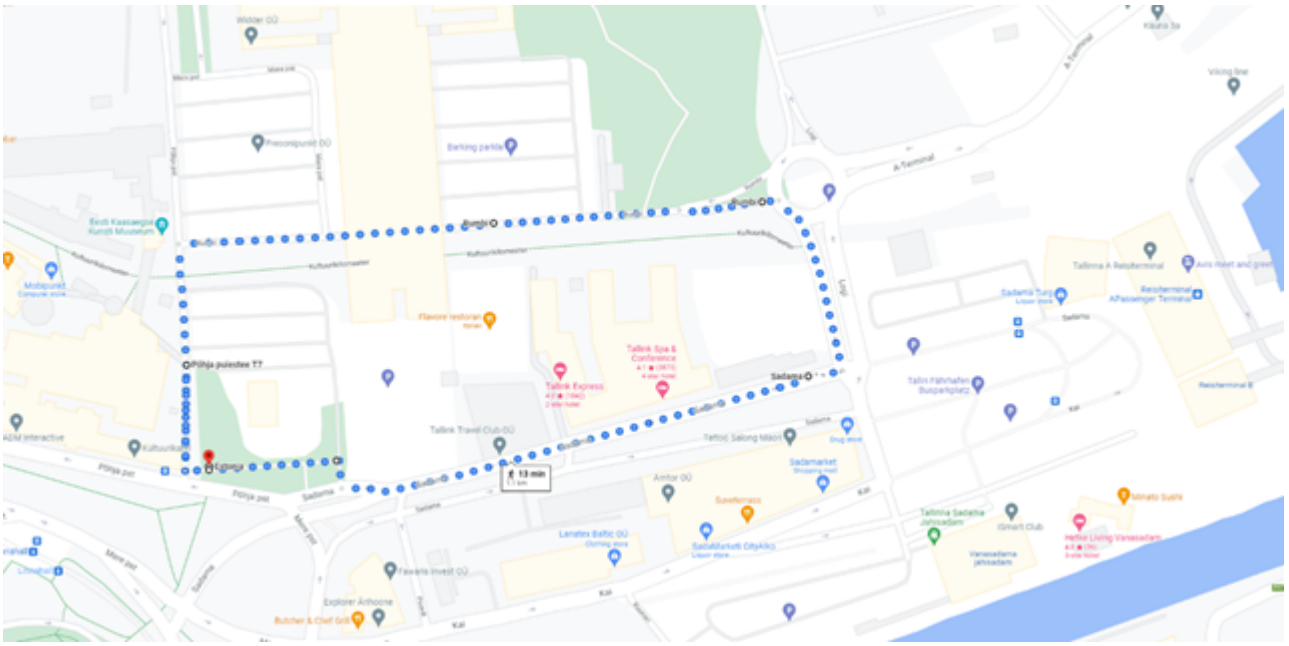


Selected areas in Estonia, Tallinn are TalTech campus test track and a route connecting public transport tram stop to Tallinn Harbor A terminal.

Area #1 - TalTech campus
Route length is: 0.85 km



Area #2 - Tram stop to Tallinn harbor Terminal A
Route length is: 1.1 km



Mapping with Lidar

The main purpose of mapping with lidar is to collect raw pointcloud data in order to identify static objects and distances. Pointcloud data is later used for low-speed autonomous driving and is a basis for vector or lane-let maps.

Recommended Hardware: Pointcloud mapping is carried out with a test-vehicle equipped with 32 channel Ultra Puck 3D lidar.

Range	200 m
Range Accuracy	+/- 3 cm
FoV – Horizontal	360°
FoV – Vertical	-25° to +15° (40°)
Angular Resolution – Vertical	0.33°
Angular Resolution – Horizontal	0.1° – 0.4°
Rotation Rate	5 Hz – 20 Hz

https://velodynelidar.com/products/ultra-puck/

Mapping was done with one drive-through with a speed of 5-7 km/h. Lidar was firmly mounted on the EV car 2 m from the ground.

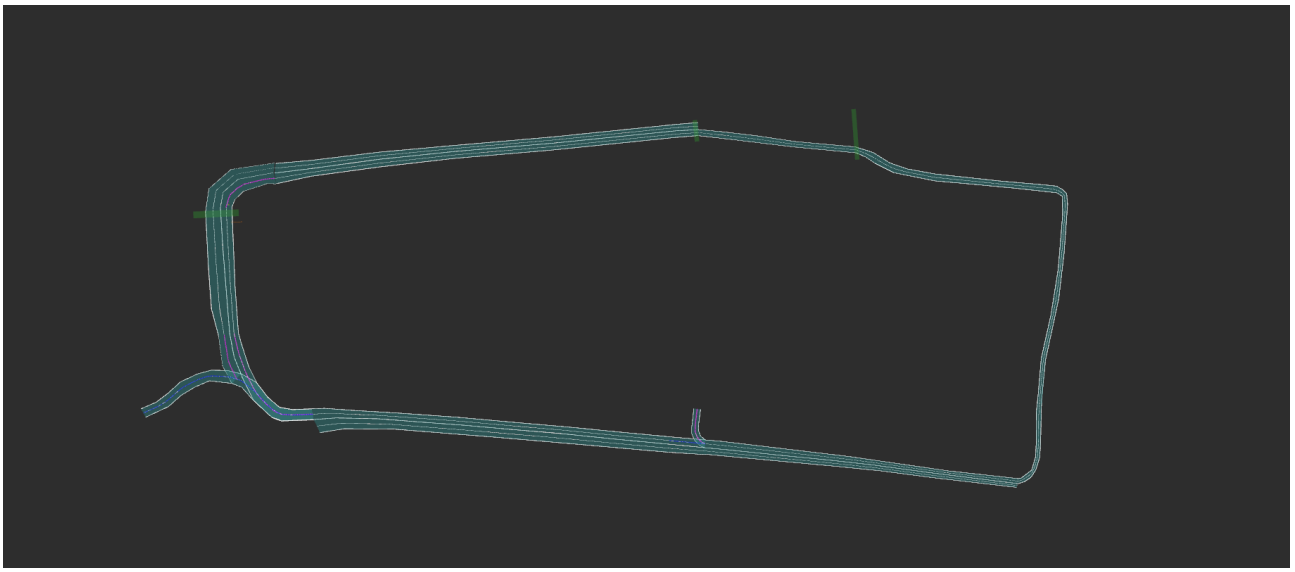
The snapshots of the mapping results are shown below.

Vector/Lanelet creation

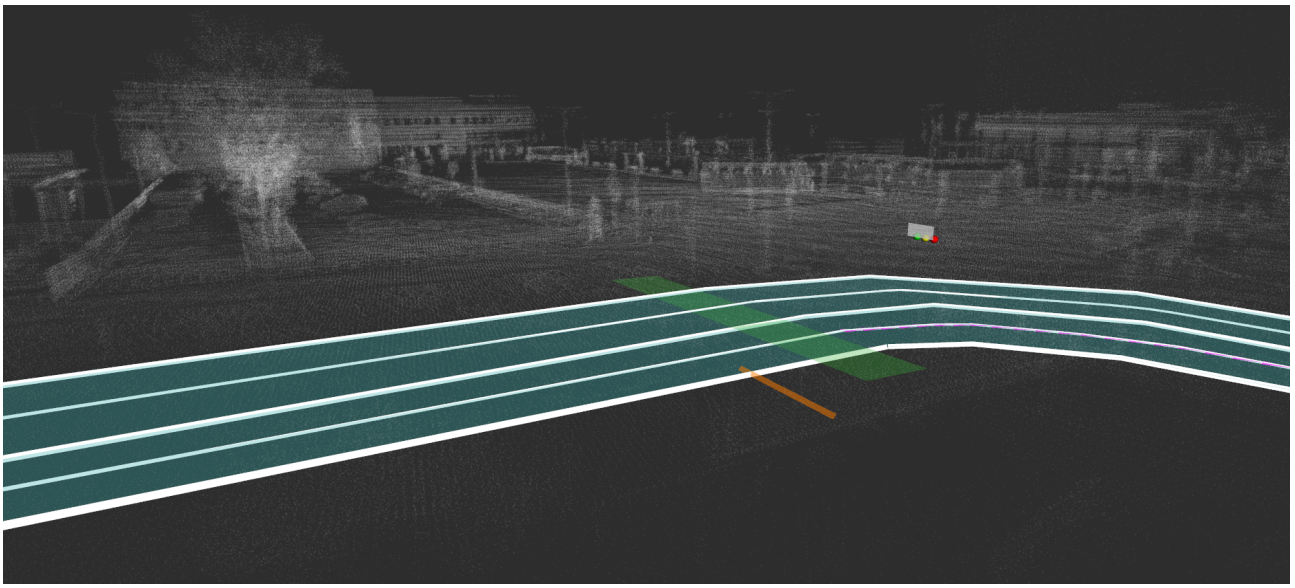
On public roads, one of the models is a map of lanes, interconnection and traffic regulations connected to them. The map has to be complete in both topological and geometrical terms.

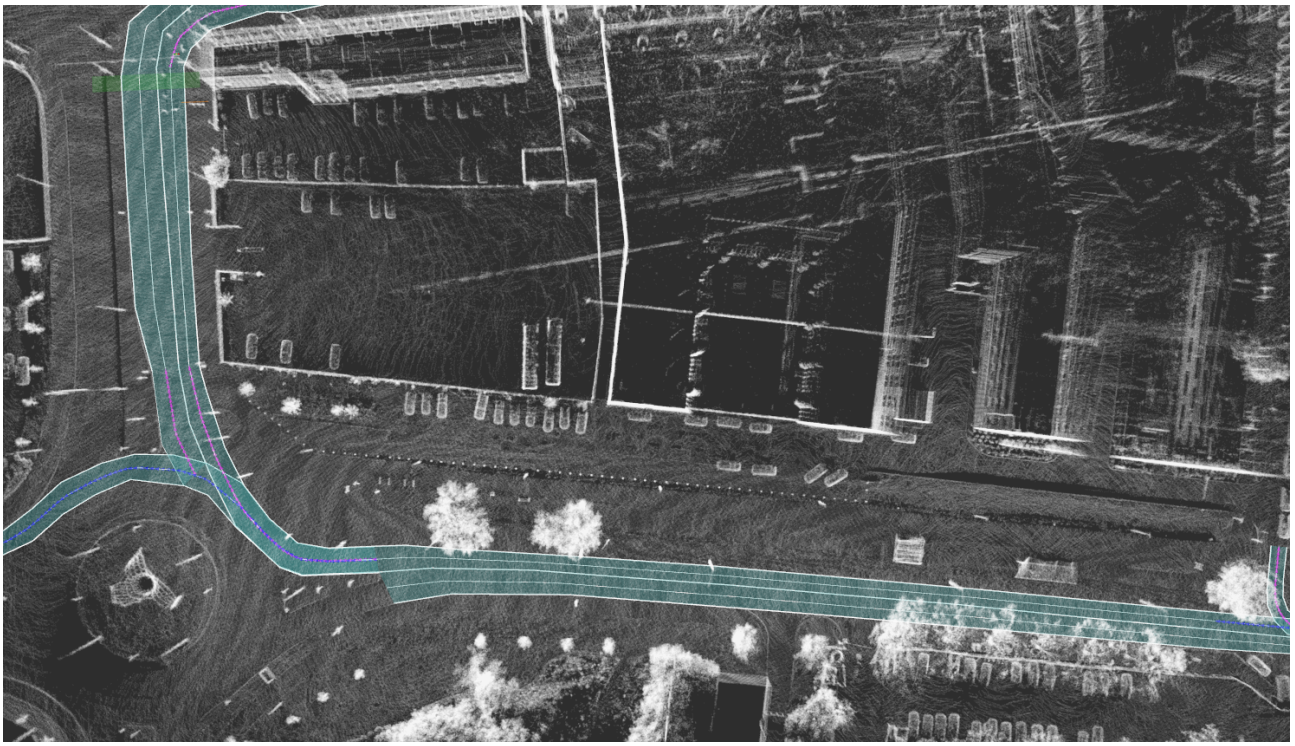
Lanelet is a novel concept for map representation, to fill this gap. The lanelet elements model the relevant parts of the environments, which are atomic, interconnected drivable road segments, geometrically represented by a left and right bound. The bound is approximated by a list of points, yielding a polygonal line or a polyline. A set of elements was identified to describe traffic regulations and attribute those elements to the lanelets. The roles of the bounds specify the driving direction, here from left to right. The lanelets compose the road network with lanes, roads and intersections. The resulting map, which we call lanelet map, is then used to infer situations, to predict their evolutions as well as find a route from the current position to the journey's destination.
The new version of lanelet map is **Lanelet2** which is a C++ library for handling map data in the context of automated driving. It is designed to utilise high-definition map data in order to efficiently handle the challenges posed to a vehicle in complex traffic scenarios. Flexibility and extensibility are some of the core principles to handle the upcoming challenges of future maps.

The following images describe the lanelet maps of area #2




Kursi
2b
Rumbi
Rumbi
Rumbi
Kultuurikilomeeter
Kultuurikilomeeter
Kultuurikilomeeter
Rumbi
Rumbi
35
m
5
7
Tallink Spa & Conference
17
Sadama
Hub
Circle K
Tallink
31
Kursi
Põhja pst
Põhja pst
Sadama
Sadama
Sadama
Sadama
6
6
6a
4
5





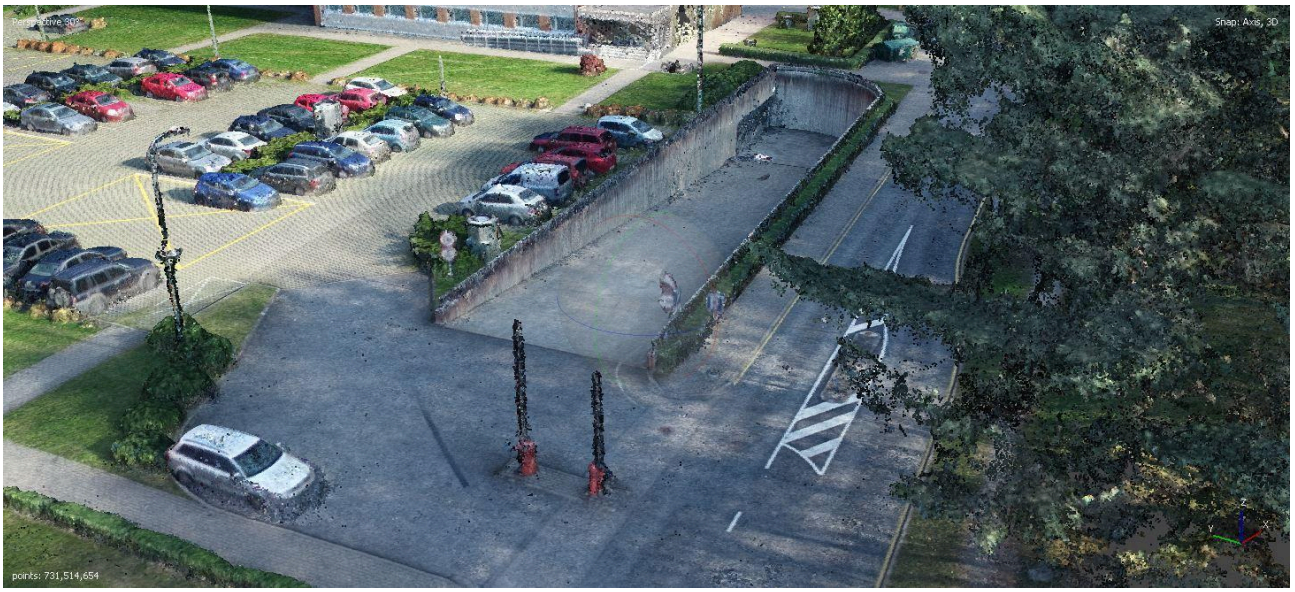
Areal mapping

In order to have a realistic map, textures are needed to be created for the simulation area. Aerial surveying is conducted in order to obtain an RGB point-cloud which is then used to create a terrain object in unity with textures. The surveying has to be conducted with georeferencing in mind. This could allow for easy plug and play where a point cloud map is needed without losing geographical coordinates. Once the point cloud is obtained through photogrammetry software, it has to be segmented and classified.

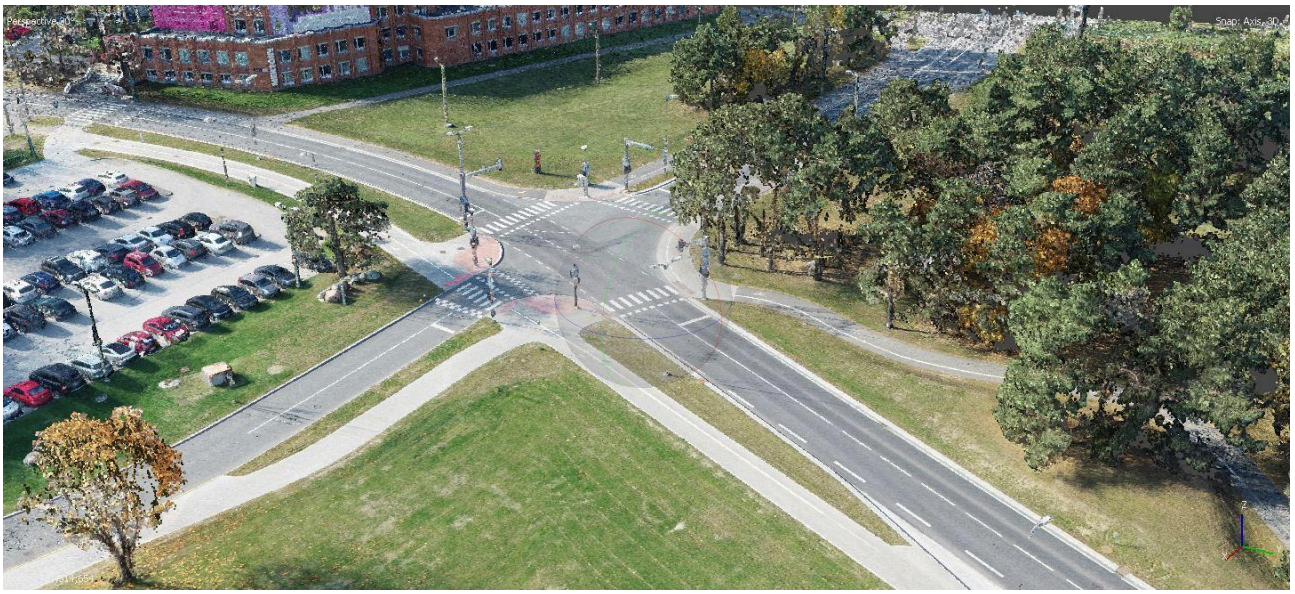
Recommended Hardware:
DJI Phantom 4 RTK with the base station (see image below) would be the best choice for mapping if photogrammetric surveying is needed. However smaller models like the DJI Mavic Air or Pro

series in conjunction with ground control points could yield similar results for the cost of work hours.

The following images shows maps from Area #1.


Snap: Axis, 3D
points: 731,514,654

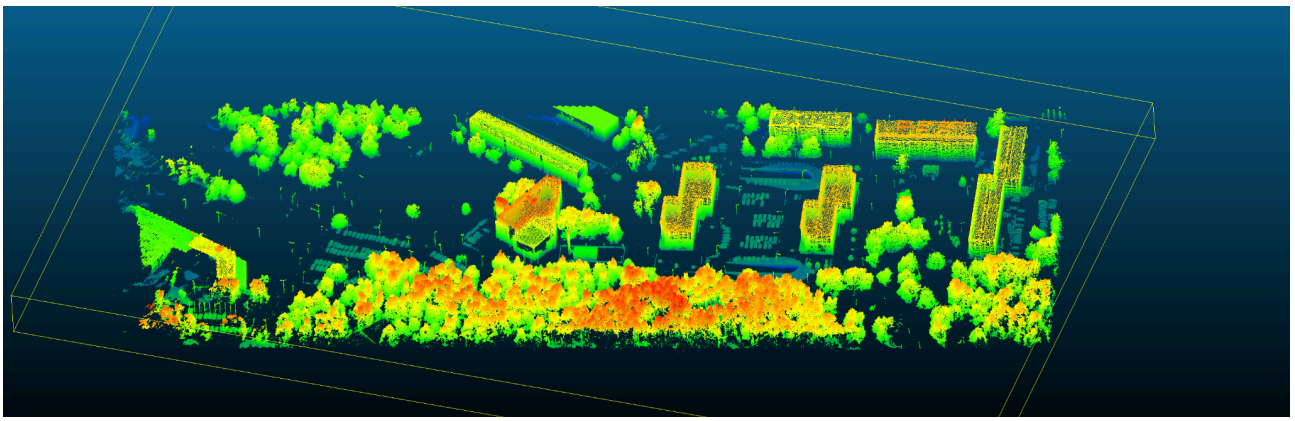

Perspective 30°
Snap: Axis, 3D



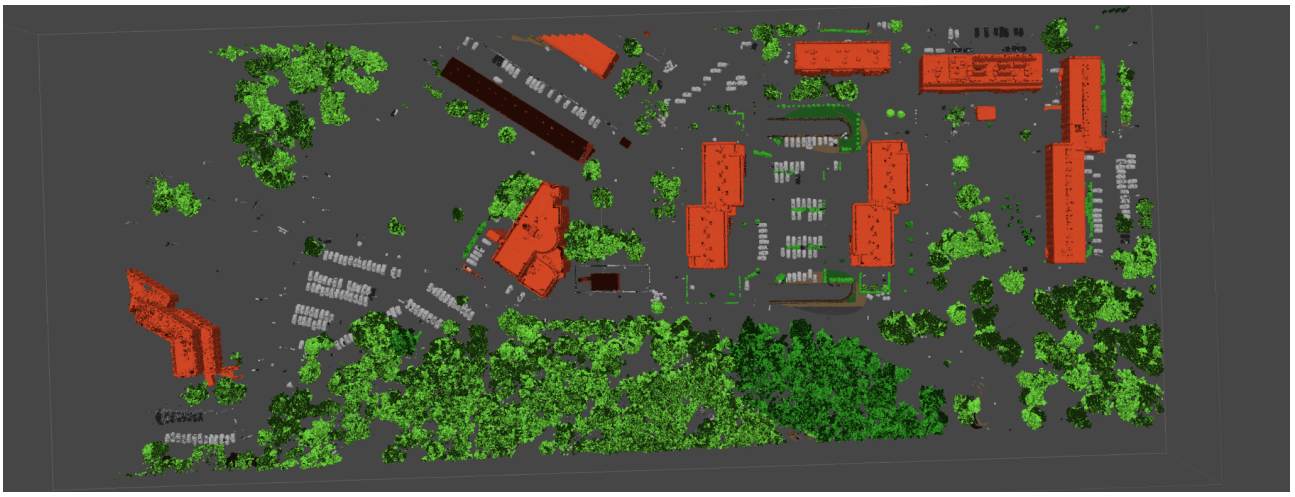
Segmentation and Classification

Segmentation is layering out different object types on the point cloud. Segmenting a point cloud into two layers, off-ground points and ground points makes it easier to select a group of points in respective layers for classification. CloudCompare, a 3D point cloud and mesh software, has great tools to perform segmentation operations.

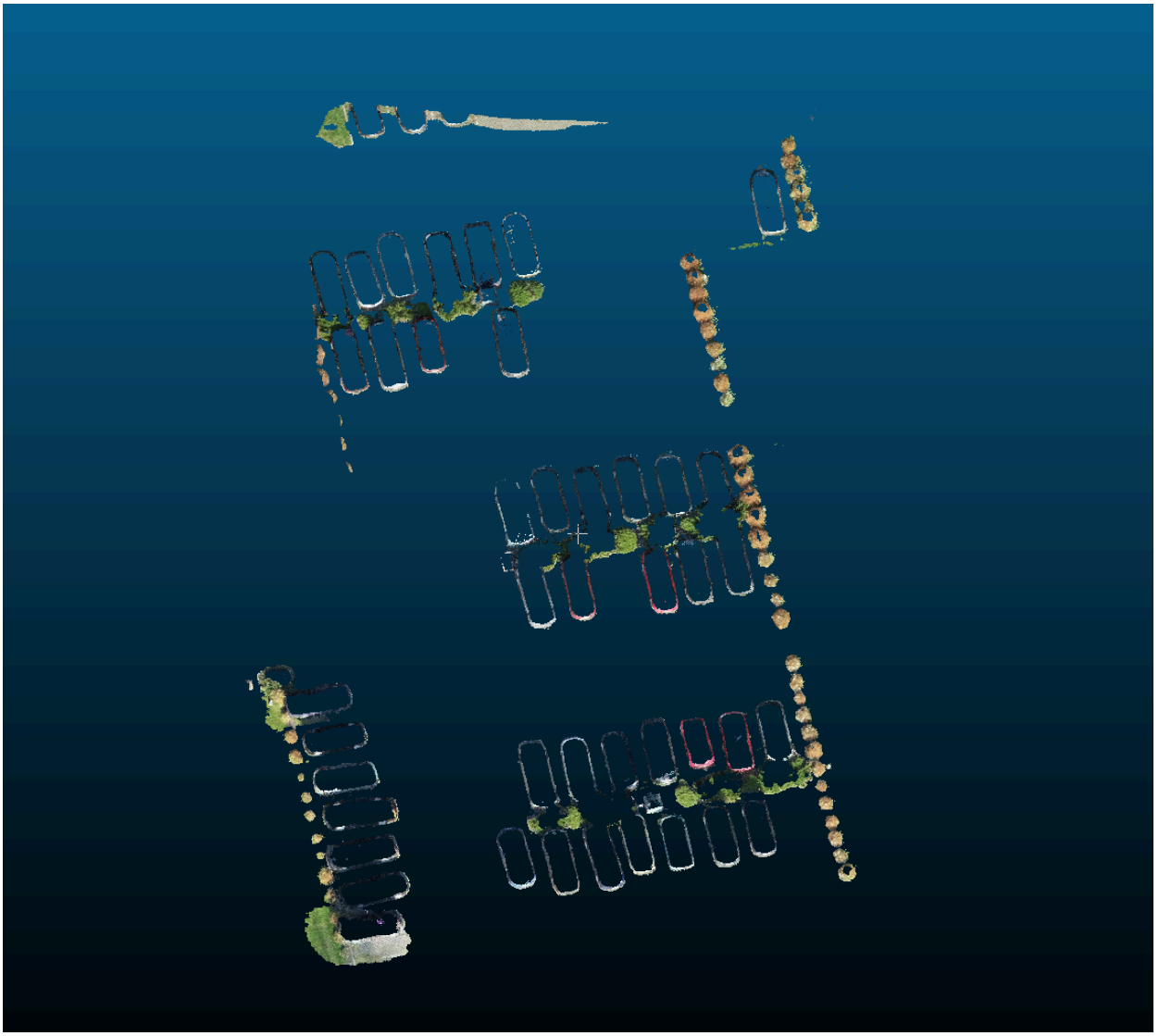
Selecting points based on a Z-direction threshold yields a rough separation of ground points.

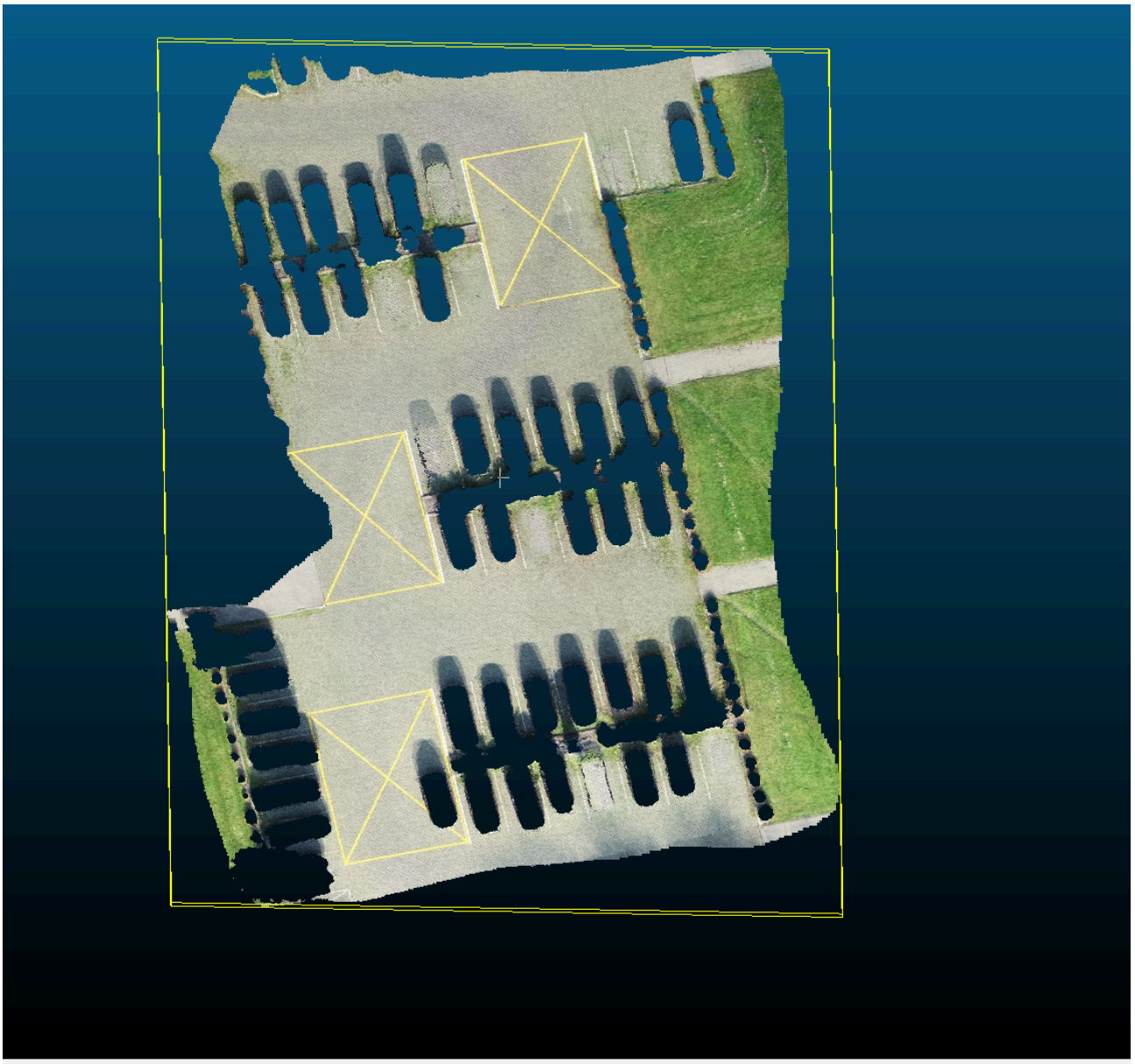


The resulting point clouds can then be classified based on the LAS 1.4 specification.



Although CloudCompare does a really good job separating these points, due to differences in the shape of the terrain, there might be some points not belonging to the ground on the segmented ground points. In these cases, the ground point needs to be segmented in parts.



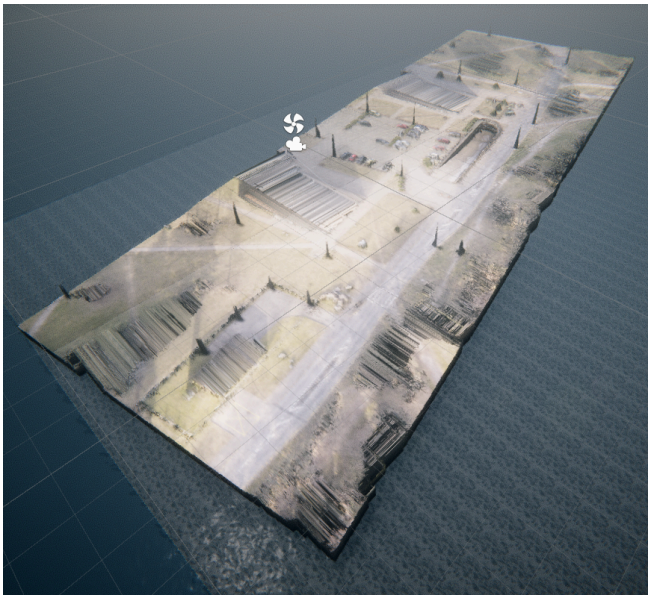


Once a desired level of detail has been achieved, the segmented parts can be classified and merged back together.

Virtual Environment Creation

LGSVL

The virtual environment is created in Unity with an in-house plugin with a classified point cloud file as the input. The plugin reads the point cloud and creates a terrain object based on the user's desired filters (ground, road, low vegetation etc.). The points classified as vegetation can then be used to create a mask to spawn pre-modelled objects on the correct locations.



As a result of Stage 1, Digital Twin is completed for Area #1. Pointcloud and vector map (lanelet) creation procedure is demonstrated on Area #2.

Carla Map

Creating a Carla map requires additional steps and unlike LGSVL, it runs on Unreal Engine. The point cloud obtained from previous steps is used to create a digital surface map which represents the surface of the environment in a 2D grayscale image format, and necessary textures for the ground. Once these are ready, they can be used to create a carla map. The most straightforward option is to use RoadRunner software from MathWorks. RoadRunner is a 3d scene editor for

simulating and testing automated driving systems. It can take a digital elevation map or a digital surface map as well as an orthomap and create roads and terrains on them. Road creation is handled manually by tracing the roads with RoadRunners built-in creation tools. Once a scene is created props like buildings and vegetation can be added on top of it. The textures may be applied and an OpenDrive map can then be exported directly to carla.

Stage 2 - Validation scenarios in PolyVerif

In this stage, we are validating 2-3 selected scenarios by applying the PolyVerif framework.

Design of Experiment: Given a simulatable model, the design for the experiment stage exercises the model in various configurations with an eye towards exposing “worst-case” conditions and showing the completeness of testing (coverage).

Purpose: Finding a reliable way to find edge case scenarios and cover most of the high-risk scenarios in simulation.

Two main approaches for generating driving scenarios are:

A Third-party Application (Scenic)

Scenic is a probabilistic programming language, and a Scenic scenario defines a distribution over both scenes and the behaviours of the dynamic agents in them over time. In this section in order to evaluate the Scenic performance, we will utilise it to create scenes in our working environment. To install the Scenic on the test PC, these requirements were followed:

- Ubuntu 18.04
- LGSVL 2020.06
- Python 3.8
- Scenic 2.0.0

NB: Ubuntu 18.04 has originally python 2.7 and 3.6.9 which does not follow the Scenic requirement. In order to provide the newer version for the OS, we installed Python 3.8. Then installed the Poetry virtual environment software based on the python3.8 command as follows.

```
curl -sSL https://install.python-poetry.org | python3.8 -
```

Next, to run the Scenic on the poetry environment, we should first define the poetry virtual env. python version correctly. To do that, After installing the poetry, in the Scenic project folder run:

```
poetry env use /full/path/to/python
```

Then, check the *`poetry env info`* to ensure that the python version is set 3.8. Finally, we follow the Scenic installation process; *`poetry install`* and *`poetry shell`*. In this step, if you need to activate the LGSVL Python API to communicate with the simulator while running the scenarios, you should follow the Python API installation through the poetry shell.

One of the basic needs for generating scenarios is the vector map that defines the constraint and traffic rule for the Scenic. The Scenic accepts **OpenDRIVE** map xodr map files. To create our TalTech campus OpenDRIVE map, we used the simulator Unity built-in feature to manually mark the traffic region inside the virtual environment (see Fig 2.1), then export them into the proper format. However, the Scenic 2.0.0 parser didn't recognize the map exported from Unity and we found out that it's a parser issue and we have to upgrade the Scenic parser to the latest version[src/scenic/formats/opendrive/xodr_parser.py].

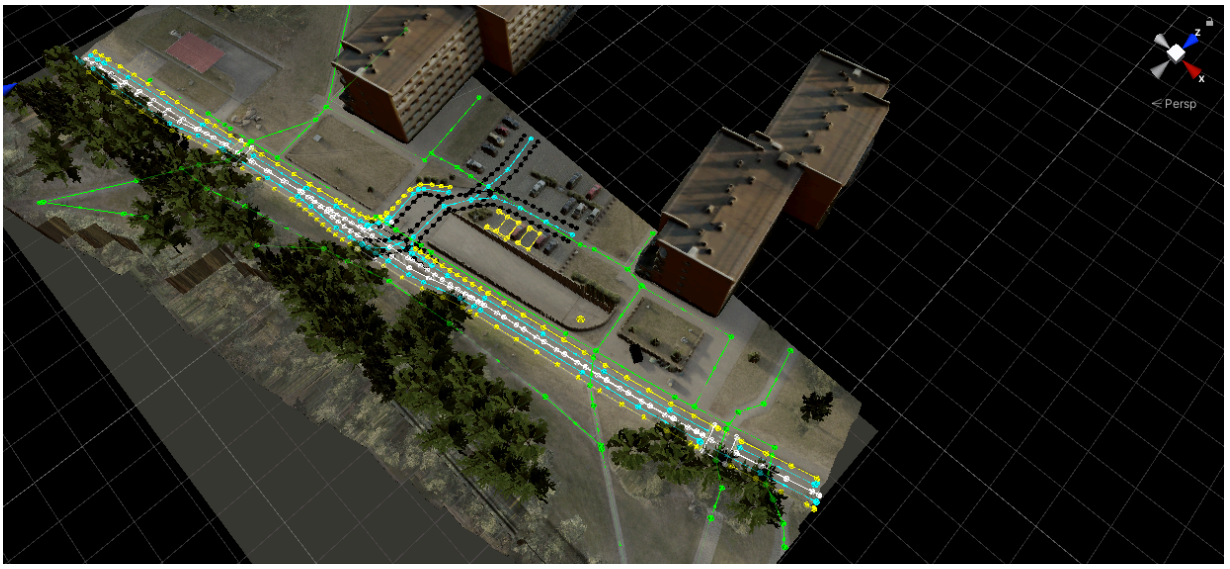


Figure 2.1 : Unity OpenDRIVE map creation. Green lines are pedestrian pavements.

Here we use a sample scenario file as follows, The iseauto is controlled not by its software but with the simulator collision avoidance feature :

```
param map = localPath('../../examples/lgsvl/maps/OpenDRIVE.xodr')
param lgsvl_map = 'TalTech'
param time_step = 1.0/10
model scenic.domains.driving.model


behavior PullIntoRoad():
    while (distance from self to ego) > 5:
        wait
    do FollowLaneBehavior(laneToFollow=ego.lane)

ego = Car with behavior DriveAvoidingCollisions(avoidance_threshold=10)
rightCurb = ego.laneGroup.curb
spot = OrientedPoint on visible rightCurb
badAngle = Uniform(1.0, -1.0) * Range(5, 10) deg
parkedCar = NPCCar left of spot by 0.5,
                facing badAngle relative to roadDirection,
                with behavior PullIntoRoad


require (distance to parkedCar) > 5
monitor StopAfterInteraction:
       for i in range(50):
              wait
       while ego.speed > 1:
              wait
       for i in range(50):
              wait
       terminate
```

To run the scenarios, This line should be run in the shell:

```
scenic examples/lgsvl/Car2.scenic   --model scenic.simulators.lgsvl.model     --time 200
--param map examples/lgsvl/maps/TalTech.xodr --simulate
```

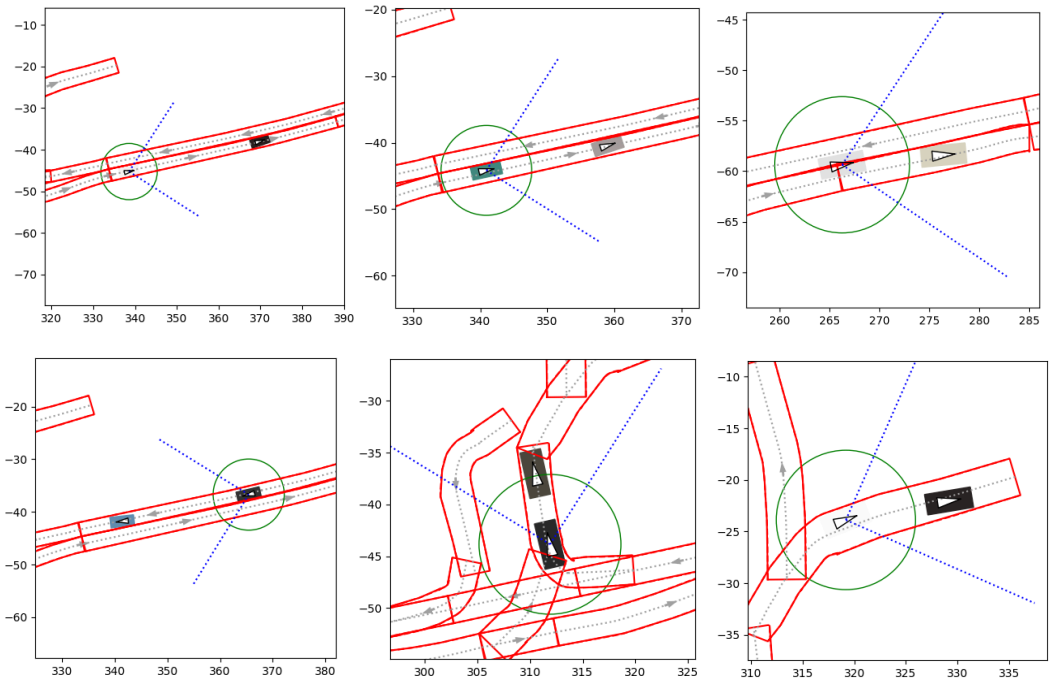


Figure 2.2 : Scenarios generated by the scenic in the region specified by the map.

Figure 2.3 : Scenarios generated by the scenic inside the LGSVL.

Identified issues

Overall, we used the Scenic to generate the scenario for the validation of the Autoware algorithm. Although, we finally found steps to run the Scenic on our platform as mentioned before but still it's better to consider the following issues:

- **Virtual Environment Creation:** LGSVL Lidar virtual sensor does not work with Terrain objects even with an appropriate mesh renderer and collider. Point cloud import throws errors but works.
- **Scenic and Python API Installation**: Python 3.8 is required for running Poetry and the Scenic. It is not provided in Ubuntu 18.04 by default. There was no straight documentation

for handling this Python version conflict. Also, the way you should add the LGSVL Python API to the scenic shell should be mentioned.
- **Scenic Map Requirement**: To create various scenarios in an environment, an OpenDRIVE map should be available of the area. There is no note for providing that kind of map and its standards. We tried different ways to export the xodr file of the area (using online tools to get an open street map then converted it to xodr) but none of them worked unless the Unity xodr map export feature.
- **OpenDRIVE Map Parser**: Polyverif uses Scenic 2.0.0 which does not parse the Unity xodr file correctly and face errors. We used the 2.0.1 beta version parser to use our map during the process.
- **Scenic Actors Behaviour:** To define and control the behaviour of actors, especially the ego inside the scenarios, there is not much material discussing how to involve control software such as Autoware in the process. The example includes behaviour controlled by the simulator itself that is not the aim of validation.
- **Scenic Samples:** There is a lack of several step by step examples to define a complex scenario for LGSVL. Most of the examples that existed inside the LGSVL folder can not be run for some reason.

- Internet connection requirements for the new SVL simulator limits the flexibility of the simulations.
- Low size file acceptance for the assets such as new virtual environment and vehicle.
- Lidar virtual sensor models defects
- Small development community
- Shutting down the whole simulator project.

Recommendations

The setup process for achieving a test run for the full pipeline is far too complicated and time-consuming. The entire software stack could be containerized and run with one script. There should be a way of launching an example verification process without having to do more than 5 minutes of active work. All the required assets for launching an example process should be included in the repository itself.

There should be a consideration of the requirements that the user needs to run the platform. Since PolyVerif purpose is to ease the test and validation process for non-professional users, the documentation part should cover all different steps from running a scenario to validation of the result. As you are utilising the Scenic for the scenario generation part, it is so important to make everything ready to use for the user. For such a project like Scenic that is under development, users face unknown problems and it is better to predict them and show how to tackle them. One recommendation is to implement docker to put everything in a container and ease the use of the whole platform. Other than that, the documentation that helps the user to install and run the example is crucial. In the following you can find some suggestions for the improvement:

- Consider the OS compatibility and the required libraries.
- Add how-to prepare a xodr file and an environment to your test platform.
- Include some simple to complex test cases in your examples.
- Add Autoware.ai support. Autoware.ai is much more mature and will remain to play an important role for the following years.

Stage 3 - Scenario Validation with Physical Device

In this stage, the chosen scenario was performed both in simulation and the real world with the intention of finding inconsistencies between the two. Both performances were recorded and compared. For more in-depth diagnostics and comparison the autonomous driving data were also recorded.

Description of the equipment

Vehicle: iseAuto ver 1.2 #3 (Level 4 Autonomous Vehicle, 6 seat minibus for urban mobility) powered by open-source Robot Operating System & Autoware.

iseAuto ver 1.2 #3 Technical parameters

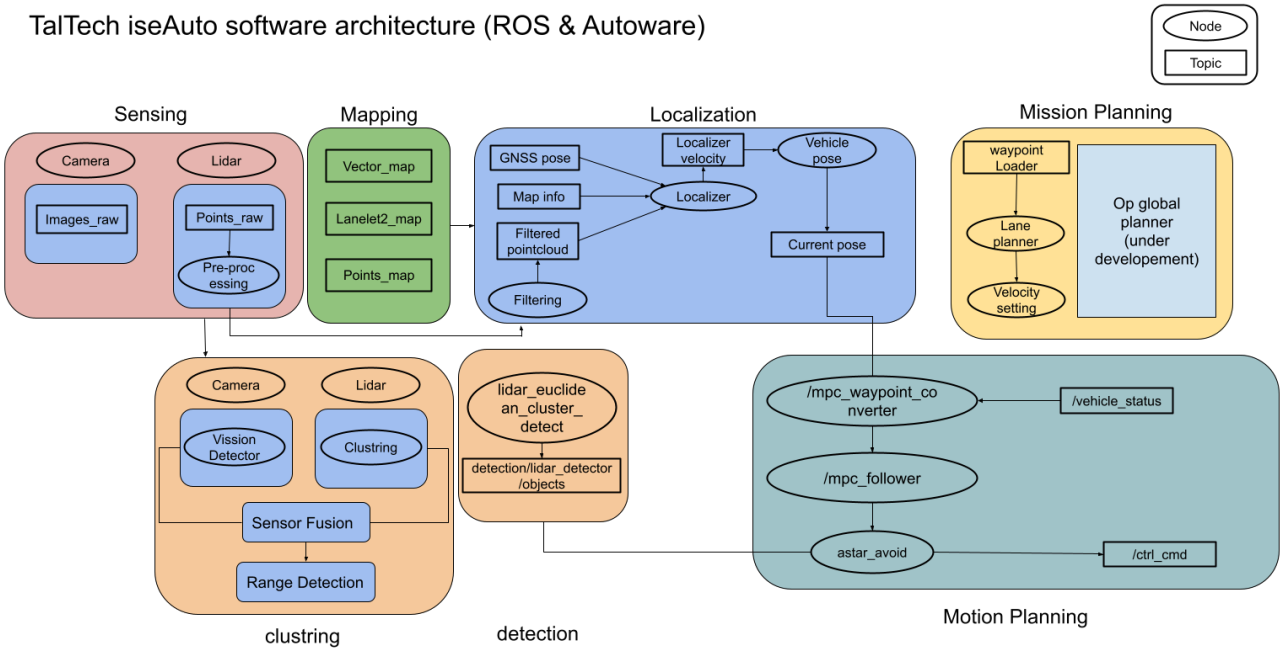
- Passenger capacity: 4 + 2
- Speed: avg 10 km/h, max 50 km/h
- Turning radius: 9 m
- Main motor power: 47 kW
- Battery: 16 kWh
- Mass: 1250 Kg
- Height: 2,4 m
- Length: 3,6 m
- Width: 1,50 mm

Sensors:

- Main lidars Velodyne x 2
- Side lidars x 2
- Front safety lidar
- Front cameras x 4
- Back camera
- RTK-GNSS + IMU

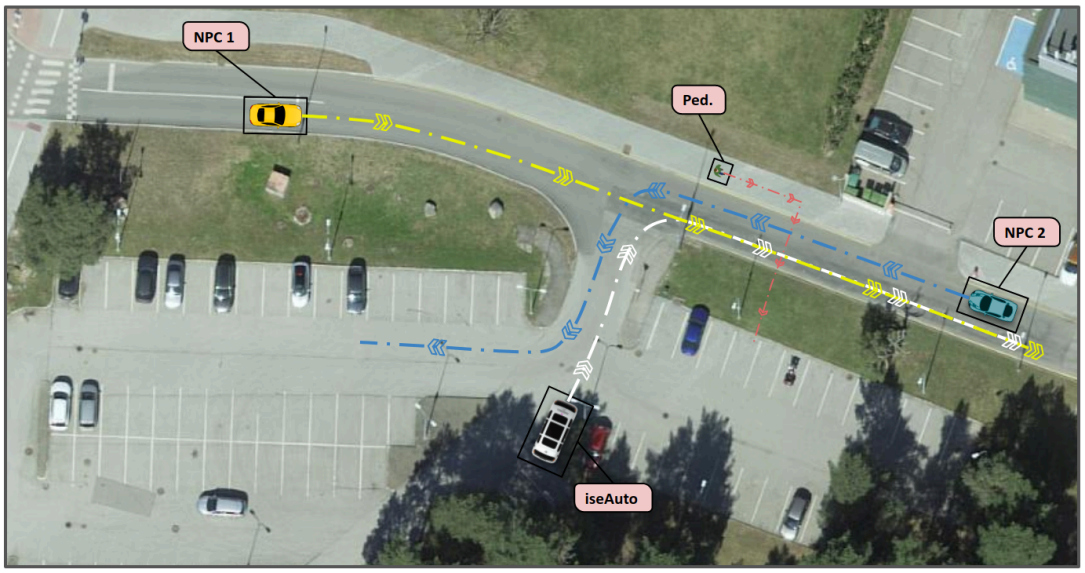
The vehicle software is running on the ROS and autonomous driving stack Autoware.ai Customized software architecture is shown below.

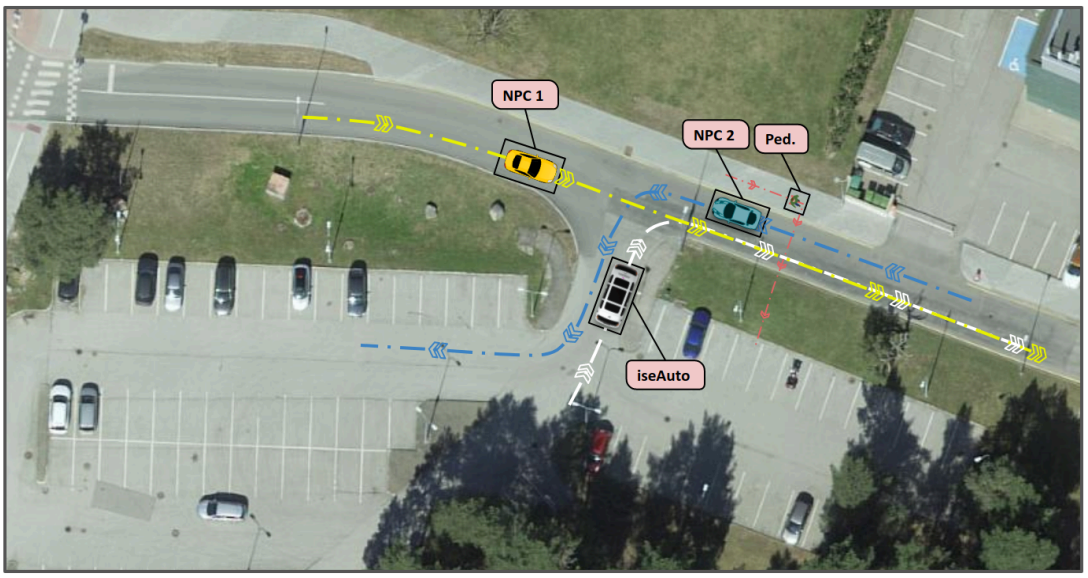
TalTech iseAuto software architecture (ROS & Autoware)

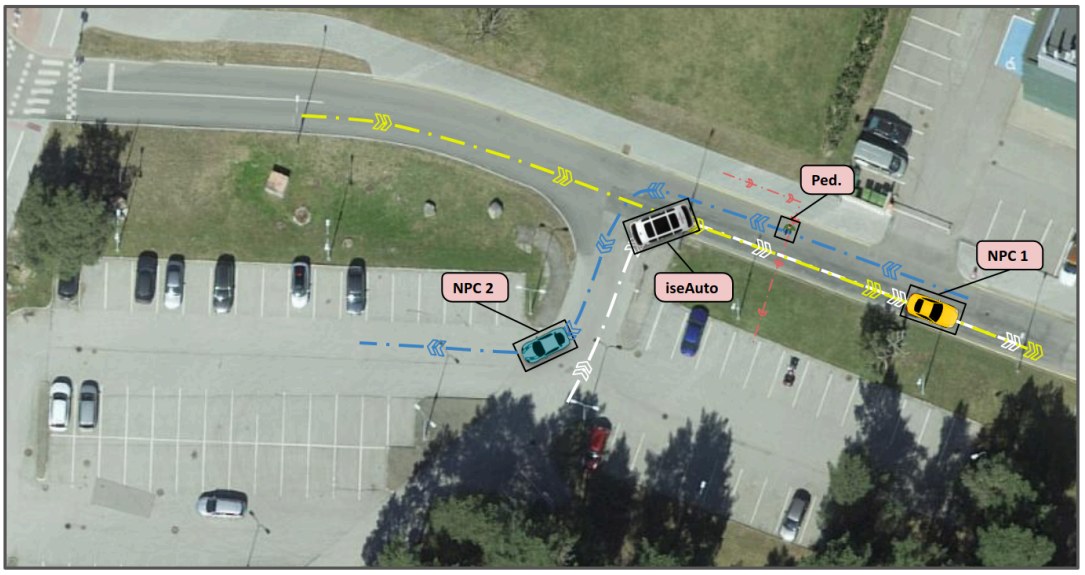


Scenario

Validation scenario was chosen based on Area #1 - TalTech campus as described in chapter Stage 1. In this route, the most challenging situation was selected involving the intersection, and other cars. Such a scenario was chosen because it includes multiple aspects that are perceived to be difficult for autonomous vehicles. The scenario includes different yielding situations and a passenger crossing the street illegally. The defined scenario is visualized below.







Due to complications introduced by internal issues in the SVL Simulator, it was decided to use a simplified version of the scenario for validation. The decision was also influenced by the fact that the weather conditions would have made it difficult to replicate the scenario in physical validation.

Simulation

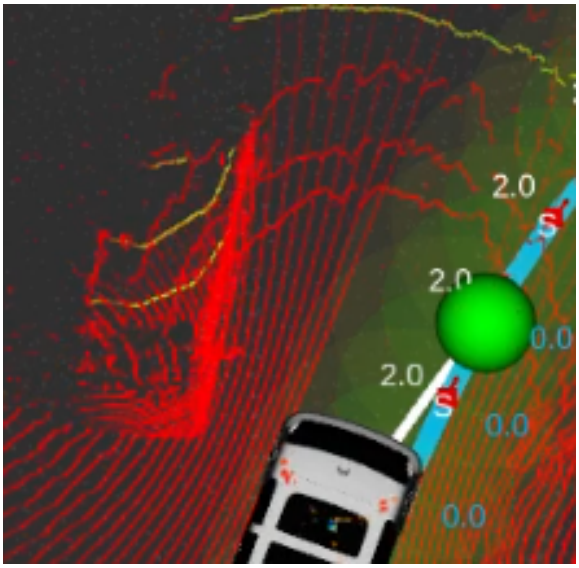
SVL Simulator 2021.3 was used for the digital twin validation. The environment was created and brought into the simulator using areal 3D scans done with a commercial drone. Free software was used to derive 3D data from the drone captures. Based on the scenario functional level different NPC (Non-Player Character) cars are defined to play in the scene. Each NPC needs at least a starting point and a trajectory to follow. This trajectory includes points, direction, and speed followed by the NPC. Furthermore, the ego (iseAuto), which is controlled by Autoware, should be spawned in a position in the environment, therefore, it needs an initial position and orientation.
In the above simulation pictures, they show a frame of a scenario in which the ego and an NPC are operating. NPC 1 waypoint and starting position already has been shown in the scenario figures. Its speed range is [1.5 - 4] m/s which means during the scenario simulation, NPC moves with a variable speed to sound more realistic, while the ego speed reaches 3 m/s maximum in this scenario.

Physical validation

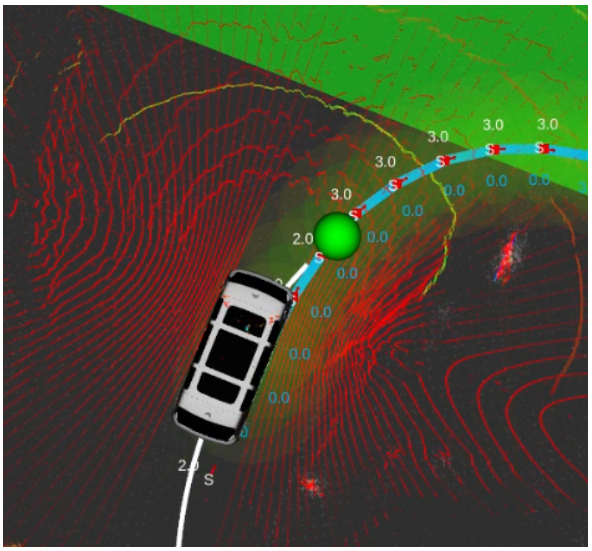
Physical validation was carried out in January 2022. The initial scenario was simplified due to the harsh weather conditions. Roads were narrowed by snow and huge snow piles were blocking the visual view as well as lidar field of view. This kind of weather was in Estonia from the beginning of December, thus not allowing to complete the initially planned full-scale scenario. However, additional valuable data was gathered on how AV handles hard weather and snowy roads. How waypoints must be dynamically modified and how to detect snow piles.

Detection Validation: The detection system successfully perceived and classified the obstacles.

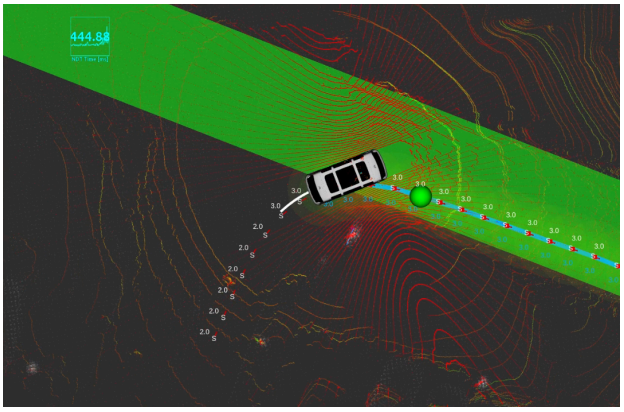
Perception Validation: All the sensors perceived the obstacles with sufficient quality to perform classification and use the data in decision making.



Localization Validation: The vehicle was able to localize itself in the predefined map.



Control Validation: The vehicle was able to follow the worked out path with sufficient precision.



Decision Validation: The vehicle was able to make the right decision and yield based on the classified data.

Gathered data

The vehicle successfully detected the approaching car and stopped to give it the right of way.

Link to video footage: https://youtu.be/TJZQ9N-2ObQ

A rosbag was recoded during the physical validation. The rosbag contains all sensor data and internal messages used in autonomous driving like decision making and path planning.

Data will be available by request

Conclusions

The experiment introduced more complications than expected. A substantial amount of the problems were caused by factors that were not directly influenceable by the PolyVerif developers or the team running the experiments.

The chosen simulator LGSVL has a lot of critical issues. The development was discontinued in the process of creating this report, and this will most probably cause issues in the future. The cloud servers, which the simulator relied upon, were shut down, and an alternative solution was provided for running the server locally. The solution, however, turned out to be extremely buggy and even made some features unusable. The simulator had issues with the accuracy of sensors and with such a small community, debugging was very difficult. The team found out that LGSVL has some fundamental flaws and bugs, which the developers acknowledged but never fixed.

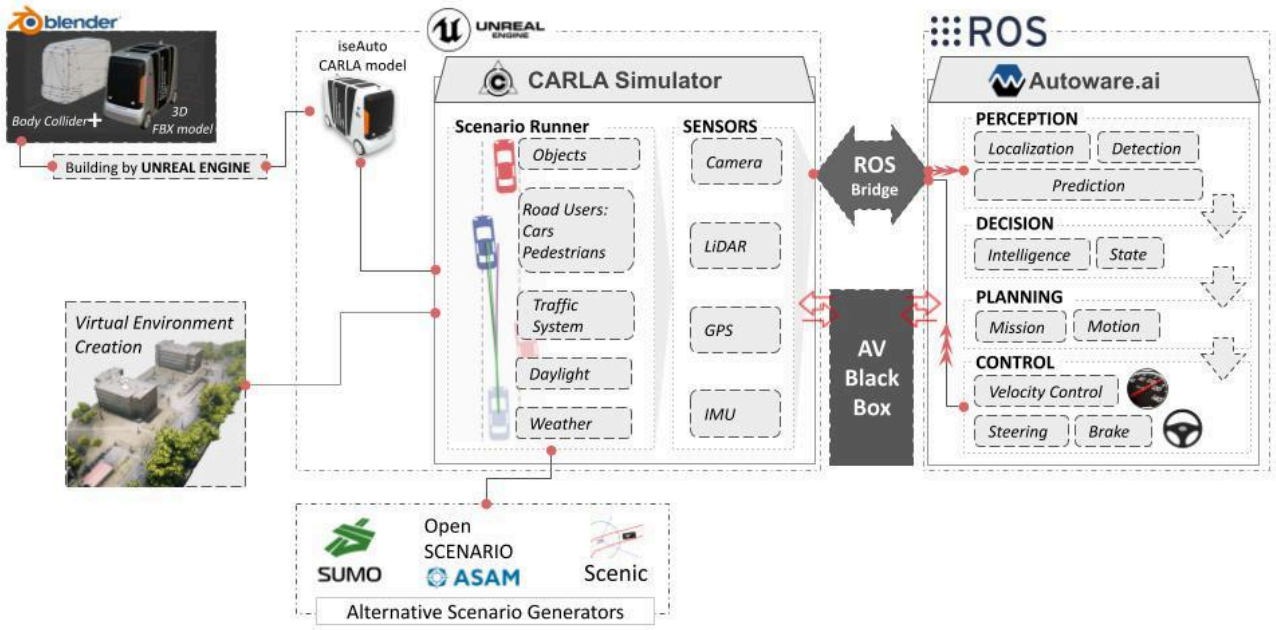
Digital twin creation was successful but still introduced some unnecessary hurdles. One example is that the LGSVL lidar virtual sensor does not work with terrain objects, even with an appropriate mesh renderer and collider. Importing the point cloud also threw a lot of errors but ended up working.

The PolyVerif suite proved to be quite difficult to work with. The software stack should be much easier to set up and run. Containerisation software like docker is strongly suggested to implement to achieve quick setup and no environment configuration.

One of the more unpredictable issues ended up being the weather conditions. Because of the proximity of Tallinn Airport a permit is required for flying the UAV used for mapping. Testing proved that for the photogrammetry software to produce usable 3D meshes, good weather and light conditions are mandatory. This meant that two unpredictable and uncontrollable aspects had to line up perfectly to create the digital twin environment. During the physical validation stage, there was a lot of snowfall and the shuttle ended up getting stuck, which made it difficult to produce a scenario similar to the simulations.

Simulation result validation in a real test case.

In conclusion, the validation team is suggesting to migrate PolyVerif to an alternative simulation framework and continue to develop verification and validation tools which are very much required and relevant for safety evaluation. The following diagram is the proposed software platform for the scenario-simulation-validation toolset.



The main architectural changes include moving from Unity and LGSVL to Unreal Engine and CARLA. The validation team's experience and public data suggests the latter suite to be a better fit to achieve our goals. The overall digital twin creation pipeline will remain quite similar. The main difference being that the vehicle model and virtual environment have to be conditioned for Unreal Engine instead of Unity. CARLA also supports advanced alternative scenario generators, which are optional and work alongside the default one included in the suite.

The validation team considers PolyVerif a very valuable and important tool in the safety evaluation toolbox and continues the joint effort to make it more mature.

Publications

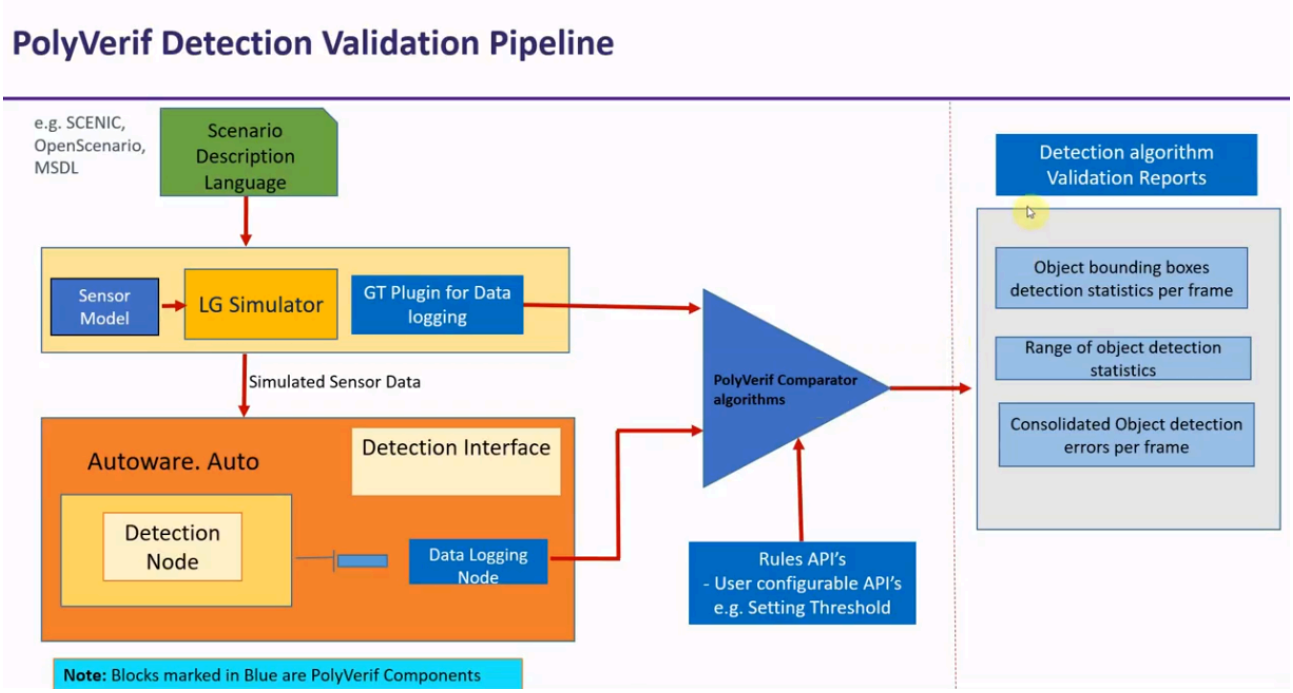
1. Raivo Sell, Ehsan Malayjerdi, Mohsen Malayjerdi, Baris Cem Baykara, Safety Toolkit for Automated Vehicle Shuttle - Practical Implementation of Digital Twin, The 2022 IEEE ICCVE – International Conference on Connected Vehicles and Expo, Florida Polytechnic University, Lakeland, FL
2. Mohsen Malayjerdi, Quentin A. Goss, Mustafa Ilhan Akba, Raivo Sell, A Two-Layered Approach for the Validation of an Operational Autonomous Shuttle

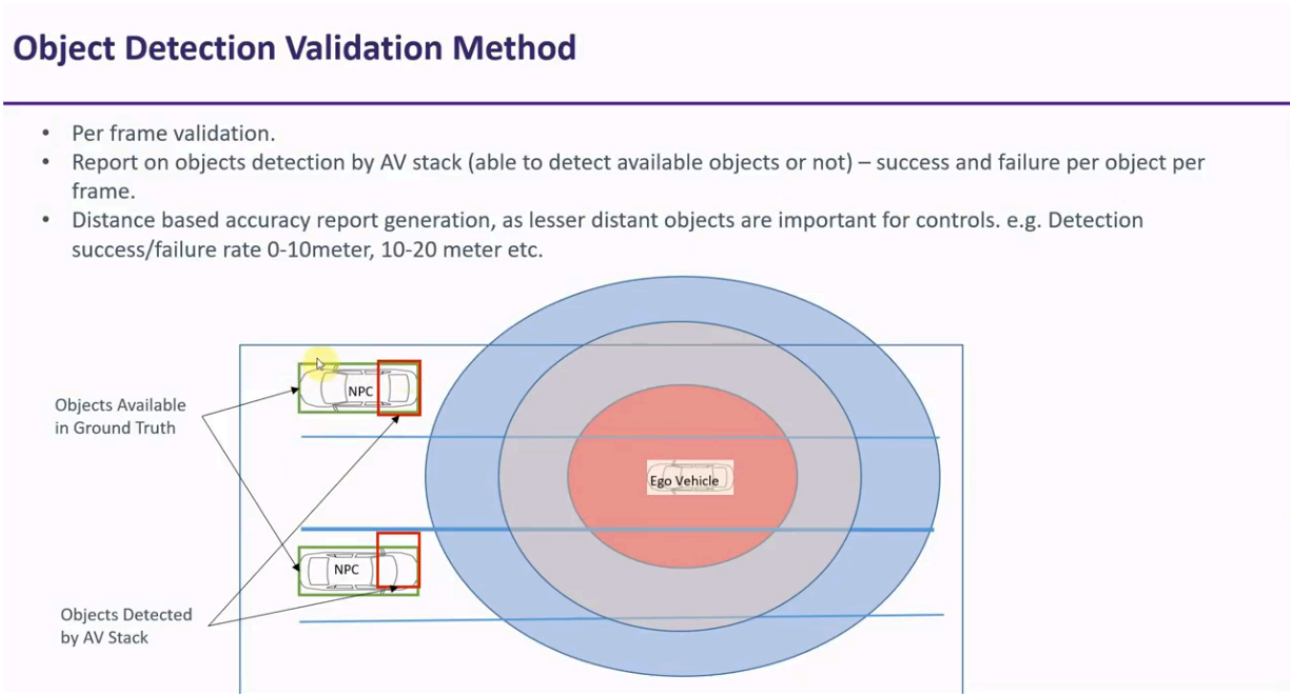
Annex 1 Background information of PolyVerif concept

Diagnostics When the virtual environment reveals a potential issue, the last stage is to transition from the virtual scenario back down to the physical world for confirmation of issues. Is the problem one of modeling or can the error occur in “reality?”

Design and validation tasks include:

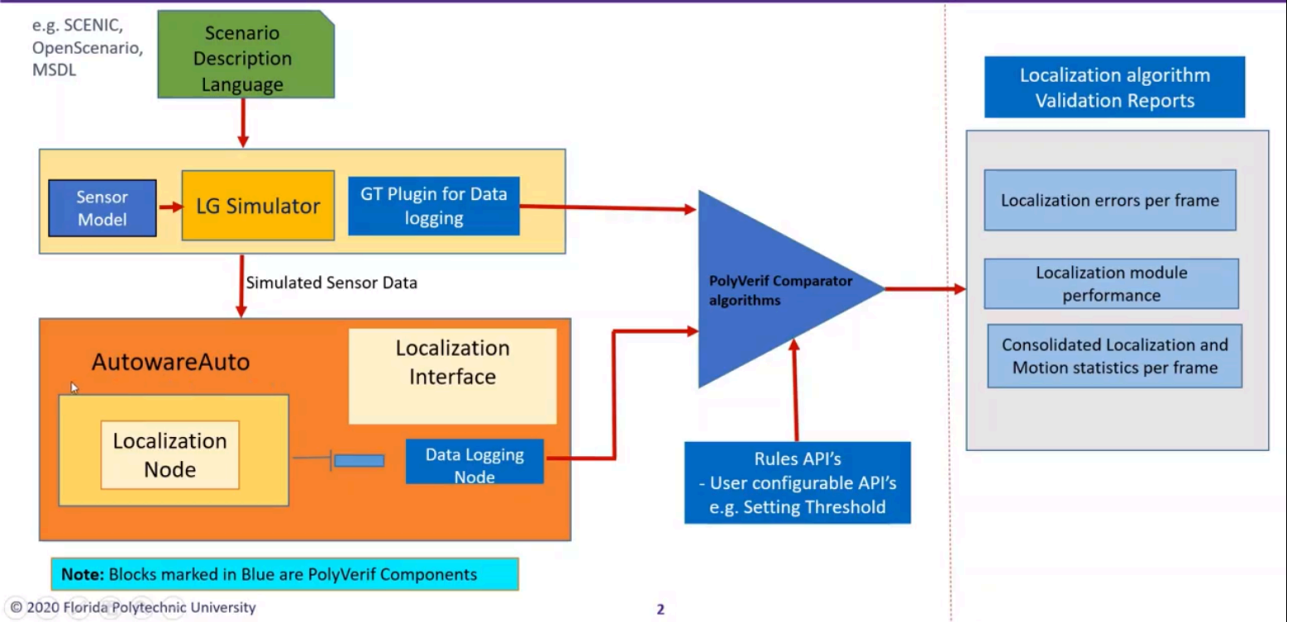
1. **Detection Validation:** Do the sensors actually “see” the objects of interest?





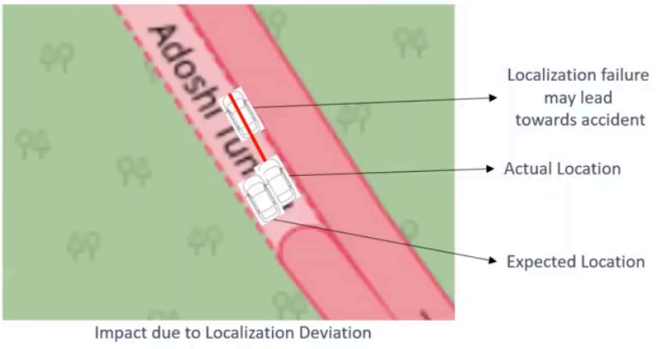
2. **Perception Validation:** Having “seen” the objects, are they recognized sufficiently to determine future movement?
3. **Location Validation:** Decisions on movement are based on the current position, is the current position “known” ..both globally and relative to local objects.

Localization validation pipeline

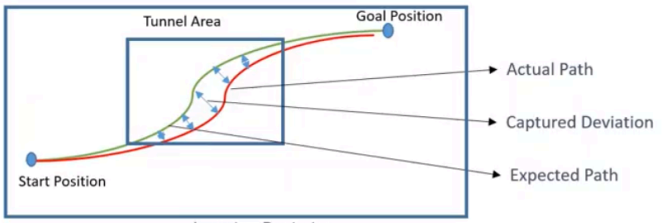


Location Validation Mechanism

- Deviation from ground truth location and ego vehicle's location in AV stack.
- Max/Min/Mean deviation parameters
- APIs for GPS & IMU noises
  - GPS noises
    - Enable / Disable at runtime
    - Add variation/noise
  - IMU noises
    - Enable / Disable at runtime
    - Add variation/noise



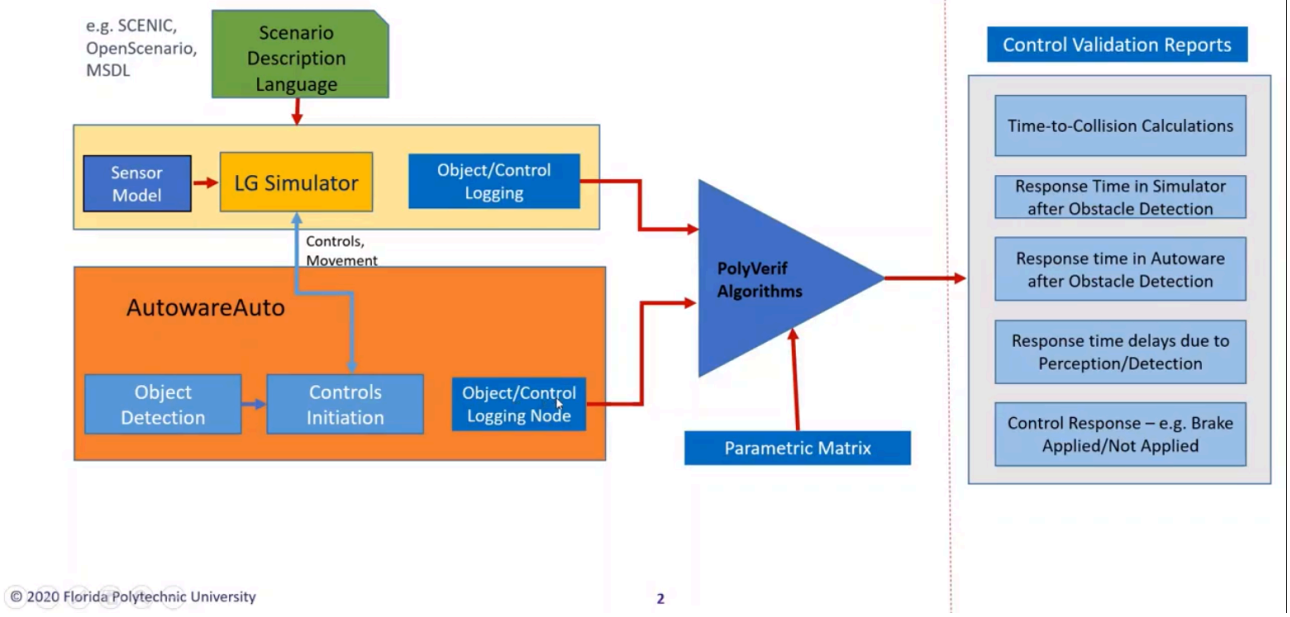
Impact due to Localization Deviation



Location Deviation

4. **Control Validation**: Many tasks in autonomy are control systems (Ex.cruise control). Are these systems stable under environmental noise?

PolyVerif Control Validation Pipeline



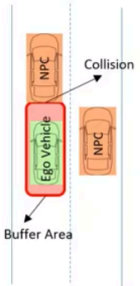
Control Validation Parameters

- TTC - Time to Collision
- Response time available with object's original location (Object's GT data)
- Response time available in AV stack with perception stack's performance
- Delay in response due to perception stack in AV

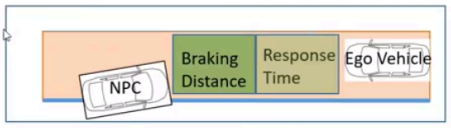
TTC Calculations

$$\mathrm{TTC}_i = \frac{X_i(t) - X_{i-1}(t) - l_i}{\dot{X}_i(t) - \dot{X}_{i-1}(t)}$$

where $\dot{X}_i(t)$ denotes the speed, X the position, and l the vehicle length

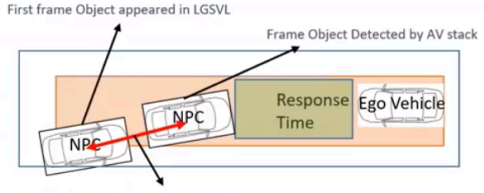


Response Time



Response Time and Braking Distance

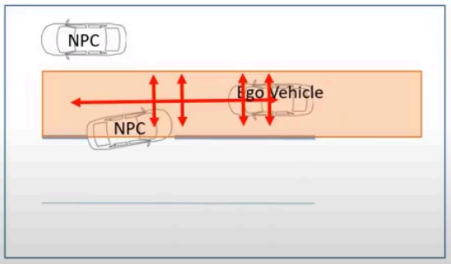
Delay In Response Due To Perception



Time to Collision Calculations - Scope

In current coverage, TTC computation are implemented considering below scenario and covering:

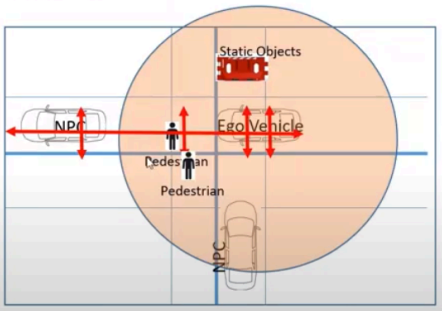
- Front and back side collisions from vehicles
- Highway and single lane scenario's



Current Coverage

Future Scope :

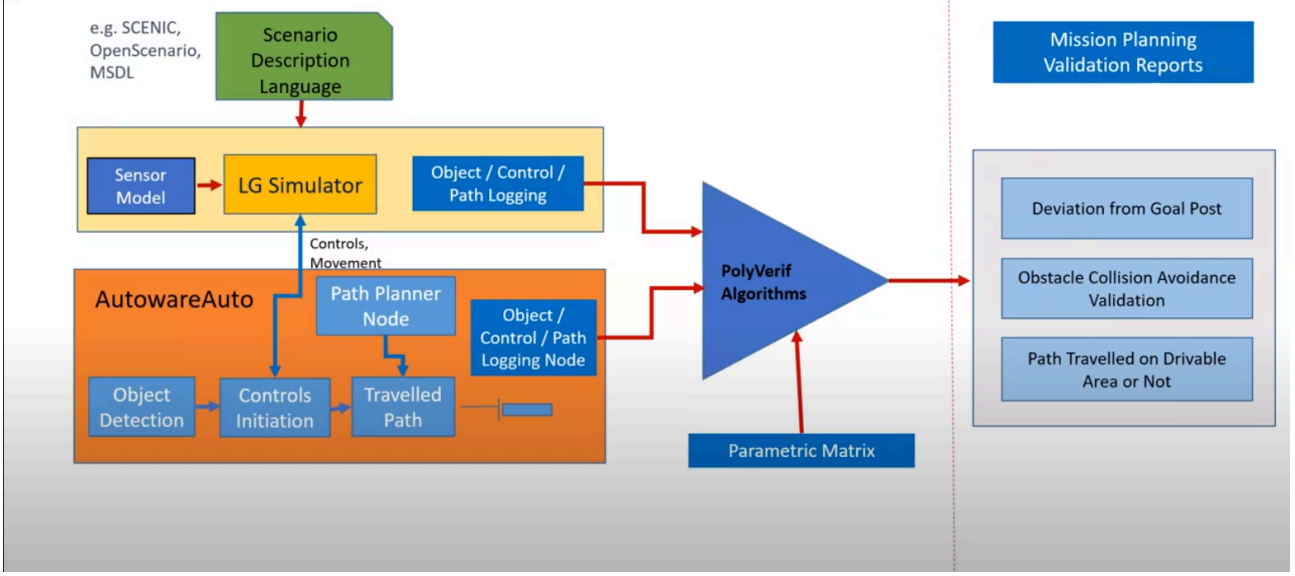
- Collisions from all sides
- Pedestrian and their dynamic motion
- Include static objects and other customized objects in simulation



Future Scope : Various Scenario

5. **Decision Validation**: Even when perception is perfect and control systems provide stability, are the correct choices on path planning being made?

Mission Planning Validation Pipeline



Mission Planning Validation Parameters

Metrics calculated:

- Goal Position Achieved
- Vehicle on Drivable Area
- Obstacle Avoidance

Future Scope :

- Driving Path Validation – Safer Path Travelling
- Obstacle Avoidance Path Validation