TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

Chair of Robotics, Artificial Intelligence and Real-time Systems

School of Computation, Information and Technology

Technical University of Munich

Final report of project:

Data Simulation for Improving Autonomous Driving

Authors	Xinyue Dai, Raphael Feigl, Hao Liu, Danyang Long, Haoyang Sun, Jiaoyang Sun
Mentor	Zhou Liguo, M.Sc.
Project Lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)
	Prof. Dr.-Ing. habil. Alois Knoll

Jul 2023

Abstract

To train autonomous driving algorithms, a simulation environment where the agent can collect training data and verify its functionality and safety is very important. This simulation environment should include realistic details and reflect cases in the real world. We as a team, decoded our semester into building a digital simulation environment of real-world cities. We learned and used tools including Unity and Mathworks Roadrunner to build up the environment. We looked up the road-building regulations so that the road details would be realistic and correct. We extracted geographical information from online databases about the region to aid us in building up the simulation. We also did field research to add extra details to our simulation. Despite our huge effort, we were not able to complete the simulation environment because of the large volume of work. However, we have documented our methods and resources so that any future work based on our work could have much more efficient and smoother progress. Furthermore, one of our teammates analyzed a machine learning algorithm that can be trained using the Roadrunner HD maps and applied to a self-driving vehicle within the simulation environment. The analysis includes studying both the input and output data of the algorithm, as well as examining the structure of the neural network. These findings will serve as a solid foundation for future implementations.

Contents

1 Introduction

Our team, consisting of Xinyue Dai, Raphael Feigl, Hao Liu, Danyang Long, Haoyang Sun, and Jiaoyang Sun, devoted our semester to creating a digital simulation environment for training and testing autonomous driving algorithms.

1.1 Background

Our simulation environment should have factors and details that are close to real-life scenes. To achieve this goal, we selected three cities to construct their digital twins, which are Garching, Oberschleissheim, and Unterschleissheim. We have encountered a lot of problems during our implementation and the creation of a digital city takes time and effort beyond our ability. Thus, we decided to summarize our working procedures and solutions to the problems, so that groups could follow our work to fill in the blanks of our project.

1.2 Motivation

Autonomous driving depends heavily on a large amount of training data for training. Autonomous driving algorithms also need a digital environment to be tested before being deployed to real-world environments. As the concept of digital twins has become popular, we think a digital twin of a city is a very good choice for gathering training data and providing a testing environment for autonomous driving algorithms. We chose to create the cities of Unterschleissheim, Oberschleissheim, and Garching because these cities are very close to the Technical University of Munich and are very convenient for verifying the algorithms in field testing.

1.3 Tools Selection

Our supervisor provided us with two tools for us to choose from: MathWorks Roadrunner [1] and Unity [2]. Unity has a tool called 3D city generator, which generates a rough city model using geographic information. Despite its convenience, it has limitations in adding more details. Thus we decided to switch to the Mathworks RoadRunner. RoadRunner is very powerful and convenient in road creation. We exported the created roads from Roadrunner to Unity to add more details. However, this implies that we need to draw all the roads and details by hand. The workload is huge and we did not expect to complete the workload. We decided to explore the tools and try out methods to add more details to the model and record these methods as guidance for future improvements to our model.

1.4 Resources and Materials

We have used materials and resources from multiple sources. These materials and resources include Unity Assets for road building, geographical information for example aerial images of the city, regulations and rules for roads in Germany, and instructions for tools used like Unity, Roadrunner, and so on. This valuable information and resources

take us huge efforts and a long time to gather. Thus these could become useful and valuable for future teams working on this project.

1.4.1 Geographical Information

We used QGIS for acquiring aerial images, the detailed procedure will be described in the report. For information about road signs and stencils, we conducted field research and took a lot of pictures in the corresponding cities. In the later stage of the project, we found the database called BayernAtlas [3] that could be very useful for reconstructing the cities in Bavaria. This database includes simplified 3D models of all the architecture in the cities of Bavaria. Unfortunately, we found this database too late and did not have time to integrate the information into our project. However, this could become a valuable hint for the later works which will simplify the digital reconstruction of the architectures to a very large extent.

1.4.2 Instructions on the Tools

In our project, different tools were used. The main tools include Unity and Mathworks RoadRunner and other tools, for example, Plastic SCM and QGIS were also used as compliments. None of us were familiar with these tools before this project but we have learned along the process of this project. The learning material includes the Unity Learn Platform [4] and the documentation from the Mathworks website.

1.4.3 Reference of Building Roads

Based on the official instructions provided by Mathworks VectorZero youtube channel [5], we learned the operations of RoadRunner. For road specification, we borrowed the *Richtlinien für die Anlage von Landstraßen* [6] from the library and found the *Guidelines for the Design of Motorways* [7] published by the German Road and Transportation Research Association (Forschungsgesellschaft für Straßen und Verkehrswesen - FGSV) to know the standard of roads and to specify details about building the road.

2 Problems and Solution

We encountered many problems during the project. In this chapter, we will demonstrate the solutions and, if the problem has not been resolved, our advice and attempts.

2.1 Team Effort Coordination

When we tried out the tools of City Generator in Unity, it became clear that team coordination and version control are a problem. A city model is huge, usually several dozens of gigabytes. Git does not support projects of this scale. Furthermore, Unity projects always include files and metadata that are globally relevant. This makes merging two parts of the model into a larger model difficult because they will overwrite each other's metadata and make the project messy and error-prone. We found that Unity provides its tools for version control and team coordination. The tool is Plastic SCM. We successfully

implemented the version control using Plastic SCM. The detailed procedure is recorded in the Procedure chapter.

2.2 Geographical Information

Because the City Generator tool in Unity is not able to generate detailed city models and is not compatible with further changes, we had to switch to the Mathworks RoadRunner tool. However, RoadRunner does not provide tools for grabbing geographical information and we had to find the geographical information of Unterschleissheim, Oberschleissheim, and Garching ourselves. Some large databases for geographical information only provided information about cities in the United States, so at one point this became a serious problem for us. After some struggles, we found that we could extract aerial images of the three cities using QGIS. The detailed procedure for this solution is recorded in the Procedure chapter. We also made efforts to find elevation maps of the three cities. However, we did not succeed. We decided to proceed without the elevation information of the three cities because of the limited time.

2.3 Unity

In the process of the Unity part, we encountered several problems. The first one is the challenges in precise modeling. Accurate measurements and modeling are required for building restoration, how to locate the buildings is a really hard question. The map does not show the exact dimensions of each building, as well as the distance and relative position to each other. What we can do is Advanced modeling techniques and specialized tools may be necessary for precise restoration. At a later stage, on the website [3] found by Haoyang Sun, the digital position of each building is recorded. We suppose that, if we can import this information into the unity, the simulation would be much more accurate.

2.4 Large File Processing

As the project proceeded, the city model became very large and each load of the project required a long time to load into the Unity and the Roadrunner. As an example, an aerial image of Oberschleissheim with intermediate resolution occupies 5GBs of data and requires more than ten minutes to load into the Roadrunner. The complexity of the city model also leads to the Unity Engine becoming laggy and hard to operate. It would be better if we could split the city into different parts and implement them one by one. In this way, the complexity of each load would be largely reduced and would boost our efficiency. However, we did not explore this option because of time constraints.

2.5 Workload Allocation among Team Members

We also encountered problems when allocating workloads among team members. Because none of us had any experience in building a digital city, we could not estimate the workload of each mission. For example, we split the team into two groups, one group was in charge of drawing roads on an existing aerial image using Roadrunner, and the other group was in charge of adding details to the road model exported from the Roadrunner using Unity.

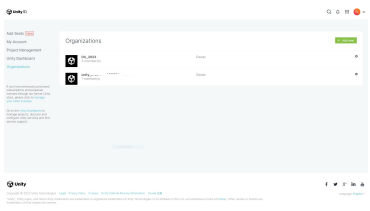
However, much more effort was required to draw decent roads using Roadrunner which made the pipeline very burdensome for the members drawing the road. This was because a real-life road includes many details for example lines, stencils, different widths in different sections when merging, pathways for pedestrians and bikes, and roads having different widths and styles in different parts of the city. The country road only has a single lane and no pedestrian pathway while the roads Constructing a digital road to fit the real-life truth requires much more effort.

3 Complementary Tools

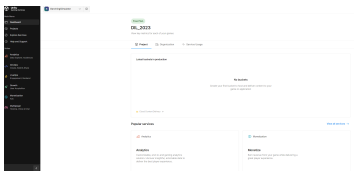
Besides the main tools, we have used complementary tools as assistance and resources in our project. Working out how to use these tools is a valuable experience and took us huge efforts. Thus, we found it meaningful to have the methods recorded in our report.

3.1 Plastic SCM for Team Coordination

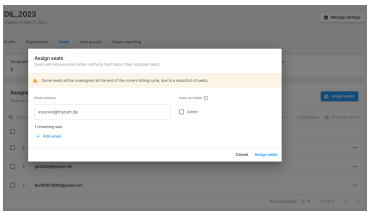
To be able to coordinate with each other on a large Unity project, we need to have a Unity Account and also install Plastic SCM on our computer [8]. First, create a Unity account, and go to my account. Then, go to the Organizations page. Create a new organization. Go to the Unity Dashboard. Go to the DevOps page. Go to Seats on the DevOps page. Click on “manage” in the Assignable Seats. Click on the “edit” icon in the manage window. If the group has under 3 members, it is free. However, if the group exceeds 3 members, you need to pay for additional seats. To save money, we advise the team to have 3 members. After ensuring that there are enough seats available, you need to assign the seats to the Unity Account of other team members. These steps are demonstrated in Figure 1. In this way, you have gathered your team in the same organization. What is left is to install Plastic SCM, sign in, and create a new repository for your project. The rest works as Git and we will not illustrate this.



(a) Creating a new organization in Unity.



(b) Extending the seats in the organization.



(c) Inviting other team members to the organization.

Figure 1: Demonstration of Unity Version control and team collaboration.

3.2 QGIS for Geographical Data

Even though geographical information, like aerial images, seems to be easy to acquire from Google Maps, However, to make it suitable to import into a project, the geographical projection information needs to be embedded in the image file. Images with these functionalities are hard to acquire. There are a lot of databases providing geographical

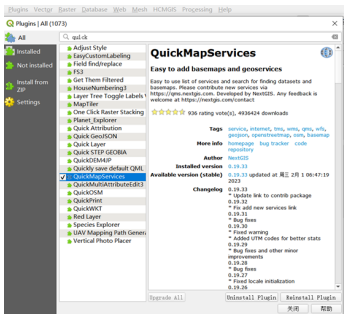
information to download, but all of them only focus on cities in the United States and do not provide data for European cities. After attempts, we have found that it is possible to extract aerial images from a tool named QGIS [9]. With an extra plugin called QuickMapService, QGIS can extract aerial images from the Google Earth database and also produce projection information, which enables the import of the image into RoadRunner. The images in this database are not of the best quality, but they are sufficient for our project. The instructions for extracting the images from QGIS are demonstrated in Figure 2. To extract the aerial image, you need to follow the following steps.

1. Download the QGIS from the website. Install the QuickMapServices in the plugins. Click on the Web Option and then in the QuickMapServices select Google, and then Google Satellite and Google Hybrid. Find the area of interest on the map.

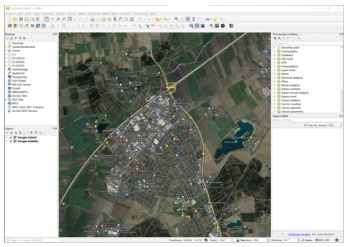
2. In the Processing Tool Box, select Raster Tools and then select Convert Map to Raster. Draw on the canvas to include the region of interest.

3. We advise setting the Map Units per Pixel to 0.1. This value has a range of 0 to 1, the smaller this value, the higher the resolution of the aerial image. Even though 0.01 will yield a better resolution, but the file will become too huge (4 GBs for a single block) and too slow to process.

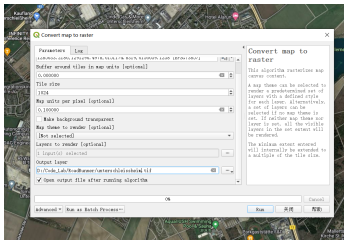
4. In the layers to render, only select the Google Satellite layer, because the Hybrid layer contains textual data. In the end, select a location in the output layer and run the process. Drag the generated file to the Roadrunner project to import the file.



(a) Import QuickMapServices plugin in QGIS.



(b) Load the Google Hybrid and Google Satellite and then locate the region of interest on the map.



(c) Set parameters to convert the selected region to raster.

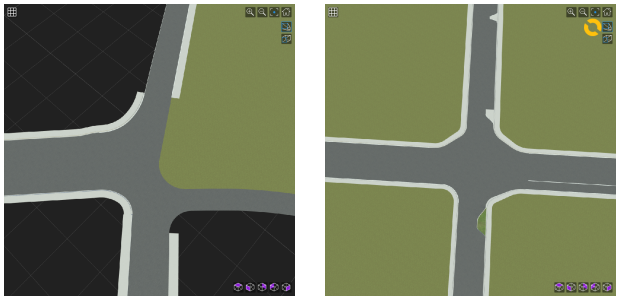
Figure 2: Instructions to extract aerial images from QGIS.

Regarding aerial images, one should notice the coordinate reference system in QGIS. The format "EPSG:3857" [10] projects a coordinate system for world between 85.06°S and 85.06°N. This is used for display by many web-based mapping tools, including Google Maps and OpenStreetMap. Meanwhile, the format "EPSG:4326" [11] projects the latitude/longitude coordinate system based on the Earth's center of mass, used by the Global Positioning System among others. The second format may create more distortions and inaccuracies in our case, which is the aerial map of Munich. Therefore, we recommend using "EPSG:3857" around the Munich area.

4 RoadRunner

After exporting the aerial image from QGIS, we import that image into Roadrunner as the base plate and create a road network of a city with its instructions.

4.1 Draw Road



(a) without Curb (b) with Curb

Figure 3: The difference between roads with and without Curb

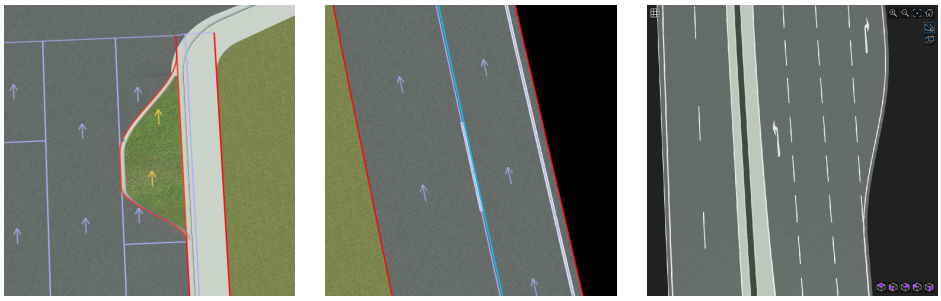
You can easily add road by following the official instruction videos and trying by yourself. We recommend choosing the road style with curbs. This can be found after clicking on the road. The difference between roads with and without curbs is obvious and is demonstrated in Figure 3. If the road does not have a curb, the sidewalk will not be continuous on the junctions. Roads with curbs will make their sidewalks more and more narrow in junction with roads that do not have a sidewalk.

4.2 Edit Road

Figure 4: Lane Edit Tools (From left to right: Lane Tool, Lane Width Tool, Lane Offset Tool, Sidewalk Height Tool, Lane Add Tool, Lane Form Tool, Lane Curve Tool, Lane Chop Tool, and Lane Split Tool)

1. Applying with the Lane Width Tool as in Figure 4 and according to *Richtlinien für die Anlage von Landstraßen* [6] and *Guidelines for the Design of Motorways* [7] as we mentioned, we set most of the width of our roads to these numbers: Sidewalk: 1.5, Curb: 0.5, Parking: 2.5-2.8, and Driving: 3.5.

2. You can add the parking spots on the side of the road by first sectioning the lane that you want to set as parking slots using the Lane Chop Tool. Then, select the sectioned rectangle by the Lane Tool and change its type to Parking. To edit the parking, use the Parking Tool to select the parking area. You can change the Entrance Side, the Angle, and the Width in the Parking Geometry.



(a) Lane Curve Tool Example (b) Lane Add Tool Example (c) Lane Form Tool Example

Figure 5: Lane Editing Tool Example

4.3 Junction Section

1. To edit and configure the junction, you can use the Custom Junction Tool to select the two ends of the roads that you want to join together, and then press space. A junction will then be created.

2. You can edit the corners of the road to fit the aerial image using the Corner Tool. To edit the logic on the junction, use the Maneuver Tool to connect the logic control points of each lain to each of its corresponding lanes.

4.4 Addition of Roads

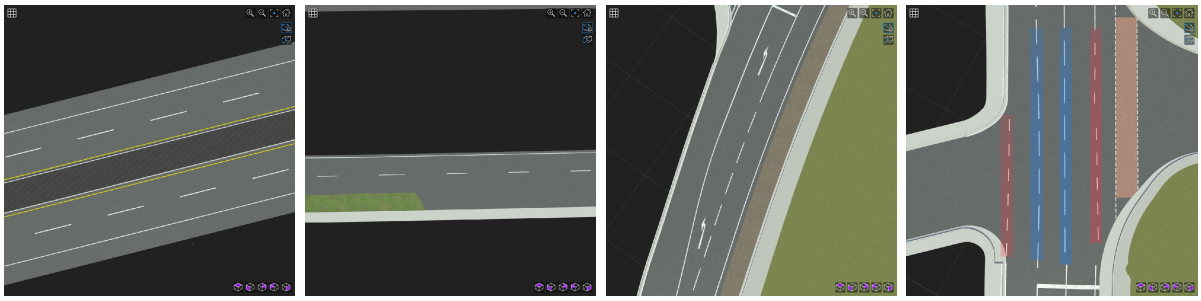
There are many details on the road that affect the behavior of a human driver. These details can not only provide valuable hints for the driving decision but also function as a very important part in the sense of whether the simulator is realistic enough. We also advise the following group on our project to keep to our effort and make the details align with the regulations.

4.4.1 Lane Marking

1. Using the Lane Marking Tool from the toolbar, you can add the markings on the road by selecting the road using the Lane Marking Tool and then dragging a marking style file from the Markings folder in the Library browser to the road.

2. You can divide the markings on the road by right-clicking on the markings. You can even delete the markings by first dividing them from other markings, and then deleting them.

3. The most common marking styles that are used are DashedSingle03 (used for markings on the middle lane of highway same direction road), DashedSingle05 (used for main roads in the city), DashedSingle07 (used for markings approaching the junction), DashedSingle10 (used for markings in the junction).



(a) DashedSingle03 (b) DashedSingle05 (c) DashedSingle07 (d) DashedSingle10 (Red: DashedSingle10, Blue: DashedSingle07)

Figure 6: Four useful lane styles

4.4.2 Area Isolation

There are usually some areas on our roads to make more space and increase safety for road users, which is defined as area isolation in our case. Here, we introduce two kinds of area isolation in Figure 7.

(a) Striped Region (b) Traffic Island

Figure 7: Striped Region vs Traffic Island

Striped Region
The striped region, in our case, is a buffer zone and turn lanes. It can be used to create a buffer zone or safety spaces between traffic lanes and other road elements.
To add the striped regions, one can right-click with the Marking Polygon Tool. We have created a special marking material which is located in the Unterschleissheim folder.

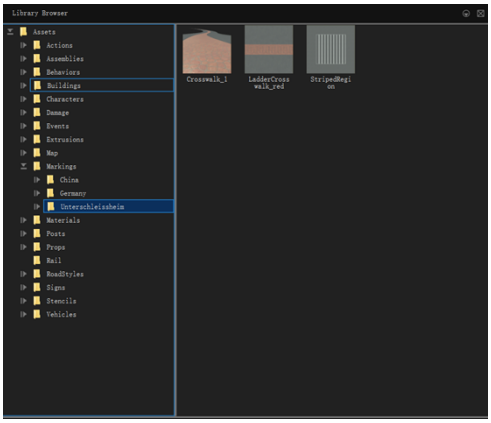


Figure 8: Markings for Unterschleissheim

4.4.3 Traffic Plates and Traffic Lights

1. Select the right item in the Props → Signs → Germany folder in the Library Browser. Open the 2D Editor from Windows. Click on the added variant in the Attributes window. This creates a copy of the format of the sign, which you can edit and reuse.

2. Use the tool on the left side of the 2D Editor to edit the sign. Remember to use the Bahnshift text font for the traffic plates because it has a similarity with standard DIN 1451 [12].

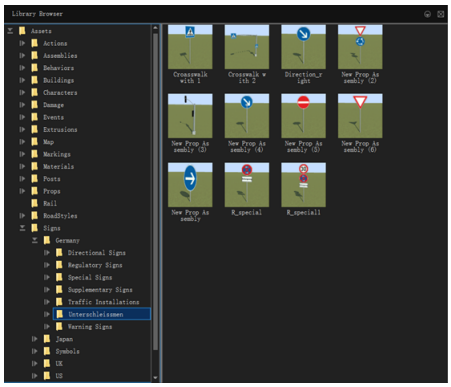


Figure 9: Signs of Unterschleissheim

4.5 Create Road Style

1. To create a new road style, use the Cross Section Tool, which can be found on the toolbar to right-click on a road that has already been modified.

2. Press the new road style by clicking on the Create Road Style in the Attributes Window. A new road style will show up in the current folder of the Library Browser.

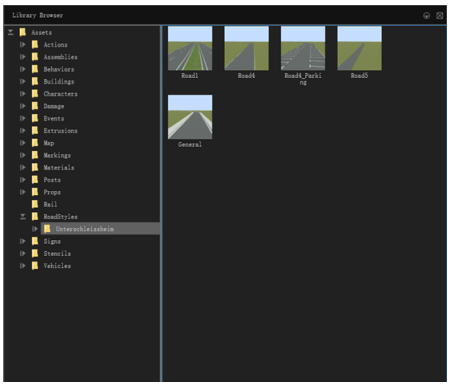


Figure 10: Road Styles of Unterschleissheim

5 Unity

5.1 Asset

5.1.1 Predispose

Before we start drawing the scene with unity, it would be better to envision the elements that will be involved in the scene, for example, pedestrians, plants, and buildings. As a powerful and popular cross-platform game development engine, Unity has a robust asset library where we can easily access a wide range of graphics, and model resources to accelerate the development process.

5.1.2 Create folders according to the hierarchical type of the model

When working with models in Unity, organizing the assets and files efficiently is important for maintaining a structured project. Follow these steps to create the folders.

5.1.3 Modify

After we import the resources into the asset, there is still more modifying work needed to do to suit our specific projects, such as scale, rotation changing, and appearance rendering method transforming. For the scale and rotation changes, we can go into the inspector window, and enter the specific values. For the appearance rendering method transforming, Unity provides the opportunity to change various properties such as lighting, materials, textures, shaders, and lighting. These options can be seen in the inspector window. It is necessary to know that when we modify one item in the asset, the changes will apply to all the scenes using this item. There is a tip we can know: when we import the building asset, sometimes there will be a diffuse reflective layer of the building which generates a diffuse reflection of the building with natural light. However, it could be shadow rendering method does not work with the version of Unity being used, or it could be a limitation of the Mac's display, this layer produces a fuzzy shell outside the model. In this case, the strategy we adopt is deleting this layer, then the building can be displayed well. In addition, there is an *add component* function that allows users to attach additional functionality and behaviors to buildings. We have not used this function, but we think this could be a useful tool for future work, such as adding some scars on the road.

5.2 Import the Road Runner model into the Unity

When we import the road model from Roadrunner, we take the following steps:

1. Prepare the file in the Roadrunner: pack the Roadrunner files into a folder. Create a new Unity project: We choose 3D (HDRP) as our template

2. Import the roadrunner model and place it in the scene. Left-click the *Asset* folder, which is in the project window at the bottom left of the screen. Then right-click and select *Import New Asset*, or simply drag the Roadrunner folder into the Asset window. Wait a while for Unity to process the import, the waiting time depends on the complexity and size of the model. After the importing process finishes, we can now drag the .fbx file from the project window into the *SampleScene*, and finally, the model can be displaced in the *Scene* window. A little tip is that you can set the axes to (0,0,0).

5.3 Restore the road details in Unity

1. After we have imported some building assets into Unity, we can do many operations to fine-tune the placement and scale the objects. These operations are realized by controlling the mouse in conjunction with the keyboard. We summarize some common operations:
   1. left-click+Q: Selection
   2. left-click+W: Drag
   3. left-click+E: Rotation
   4. left-click+R: Change the scale of the building (you can also achieve this by changing the scale in the transform window)
   5. Press and hold the scroll wheel: move and pan the object within the Unity editor.

If you want to save the new component after the change, you can drag the file from the layer into the asset and select it.

2. Restore the buildings according to the real scene
Access to maps with real street views. In this step, we referenced multiple live maps and drove ourselves to the field for a full range of photography. Listed below are some of the live map resources we refer to

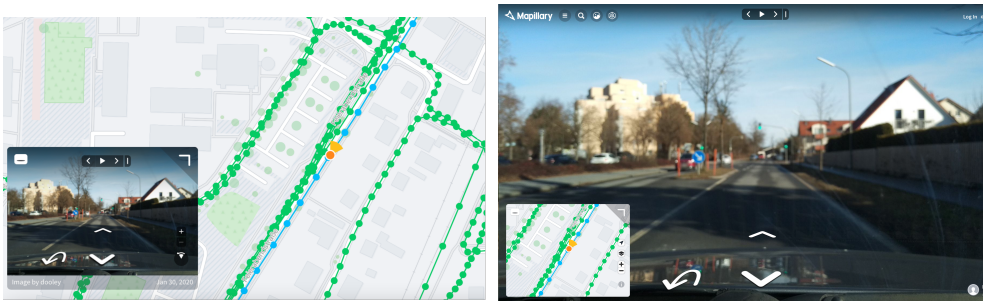


Figure 11: Street information from Mapillary [13].

(a) Our field trip to collect detailed data.

(b) Bird view from Google Earth [14].

Figure 12: Collecting geographical information from Google Earth and field trip.

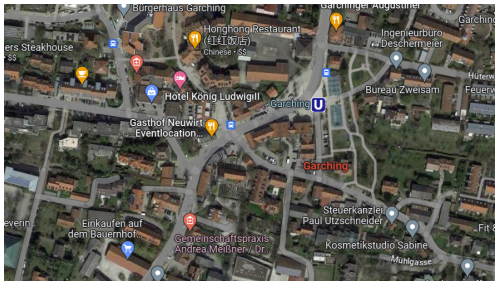
6 Garching 3D Scene

We modeled and digitally twinned this city in Unity as 1:1 as possible, based on a high-precision map of Garching, the city around Munich. We transformed the real-world map data into a virtual urban environment including road networks, lanes, traffic signs, signals, and buildings. This map modeling allows us to generate scenarios that provide realistic urban scenes for developing and testing autonomous driving systems. The Garching city scene for our simulator is designed in the following steps.

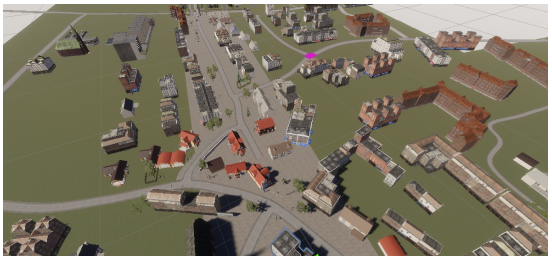
6.1 Map modeling

We use OpenStreetMap, GoogleMap, and custom map data as a basis for building virtual city environments. This map data contains the road network, traffic signals, lanes,

intersections, buildings, and other important landmarks. We read and parse map data to extract key information such as roads, lanes, traffic lights, buildings, etc. This information is used to construct a virtual urban environment.



(a) Bird's eye view Garching road network in Google Maps



(b) Bird's eye view Garching road network in our scene

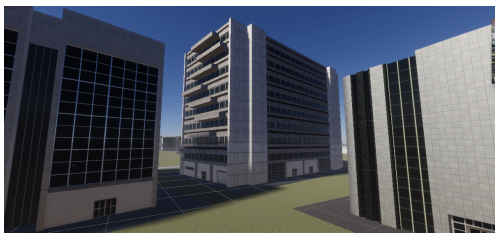
Figure 13: Google Maps vs Our Scene

6.2 Road Generation

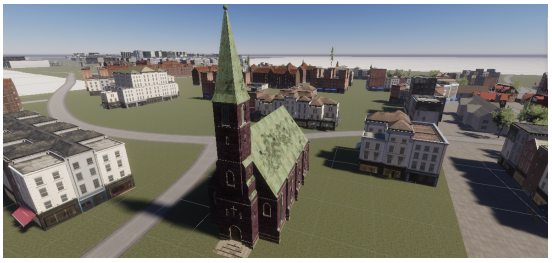
Based on the map data, we use a drawing tool in Unity called *Easy road* to build and generate our road network. We also generate a road grid based on road rules and geometric properties. Roads can be straight, curved, freeways, or complex intersections. In addition, users can manually create and modify roads in the simulator using customization tools. We generate lane markings and lane boundaries on roads based on lane information from map data. These lane markings include lane lines, dashed lines, solid lines, etc., which are used to indicate the areas and rules for vehicle travel.

6.3 Buildings and Landmarks

Our 3D scene places virtual buildings at locations based on the building information in the map data. These buildings can be buildings, stores, public facilities, etc. Landmarks such as bridges, squares and landmarks can also be added to the simulator.



(a) Urban buildings



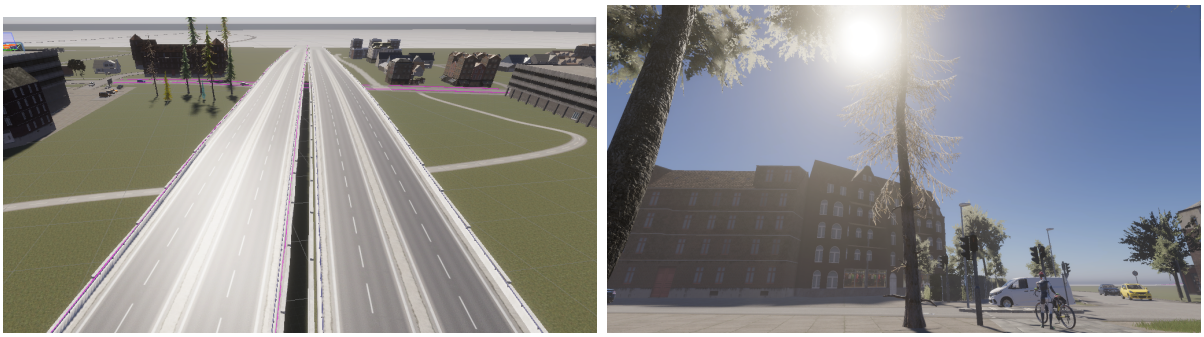
(b) Landmark building

Figure 14: Urban buildings and Landmark building

6.4 Lane markings and traffic rules

The simulated 3D city scene generates the correct lane markings and traffic signs on the road based on the lane markings and traffic rules information in the map data. This

includes lane lines, intersection markings, traffic signals, etc. These markings and rules play an important role in vehicle behavior and navigation.



(a) Lane markings on the viaduct. (b) Sunny weather conditions

Figure 15: Lane markings and weather in the simulation.

6.5 Lighting and Weather Conditions

We also provide realistic lighting and weather simulation for urban scenes. Users can select different weather conditions such as sunny, cloudy, rainy, or night and adjust lighting and shadow effects to create realistic environments.

6.6 Dynamic and static objects

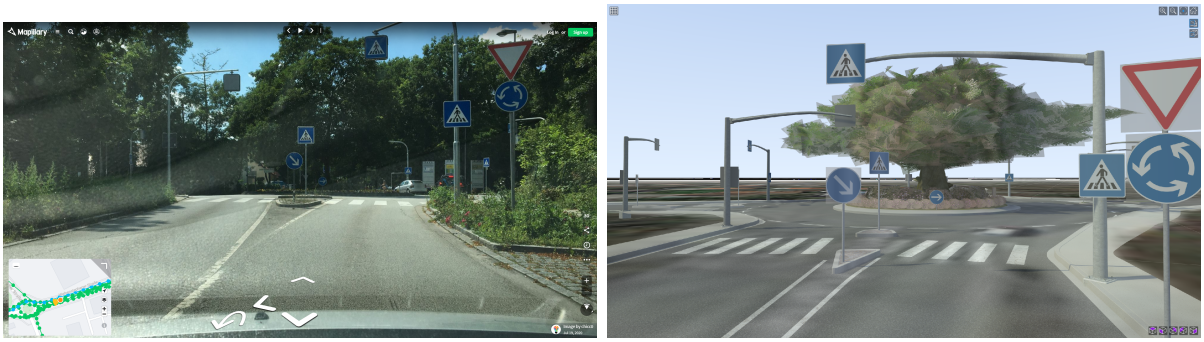
At the same time, the various types of environments in our scene consist of 3D models of static objects (such as buildings, vegetation, traffic signs, and infrastructure) as well as dynamic objects (such as vehicles and pedestrians). All models are carefully designed to balance visual quality and rendering speed: we use low-weight geometric models and textures but maintain visual realism by carefully crafting materials and using variable levels of detail. All 3D models have a common scale and their size reflects the size of real objects. By following these steps, the 3D scene Garching is able to simulate a realistic city scene including road networks, buildings, traffic rules, and lighting effects. This provides developers and researchers with a highly customizable and realistic platform for testing and training autonomous driving systems.

Figure 16: Dynamic vehicles and pedestrians on the street, as well as static objects

7 Result Demonstration

7.1 Case 1 Traffic circle simulation

As a popular intersection design, roundabout traffic plays a critical role in the public transport system. It could effectively manage traffic flow, minimize conflict points, and improve overall traffic efficiency.
However, it also comes with a complex traffic situation. The main problems with it include multiple entry and exit points and lanes, peak hours, and high traffic volume, and some drivers may fail to respect the right of way of other vehicles on the roundabout. So, we need to build a simulation scene for Traffic Island.
In this case, we select a traffic island (48.286355, 11.584840) in Untershleisshiem, and rebuild it in Roadrunner with a high degree of similarity.
Figure 17a displays the real scenario. It can be easily found that various guideboards are set at the crossing, including signs of Yield, Pedestrian Crossing, Turn Right, and Traffic Circle. Using the toolkit in RoadRunner, we did a 1-to-1 simulation of the traffic signs, seen in Figure 17b. In addition, we restored the surrounding greenery in Unity.



(a) Real view (b) Simulation in RoadRunner

Figure 17: Case 1 Traffic circle simulation

7.2 Case 2 Simulation of intersections near attractions

As all know, Munich is a tourist city with many attractions, the city is dotted with attractions, big and small. However, being close to an attraction also means high traffic and congestion, a mix of transportation (cabs, internet taxis, or other means of transportation), and potentially a lot of pedestrian traffic.
In this case, we chose a crossing (48.284724, 11.553562) near Untershleisshiem Seen. This lake is a very popular lake in the Untershleisshiem area, and at this intersection, many different types of signs are installed to indicate the different modes of transportation next to the attraction. What is worth mentioning is that we can see a white, rectangular-shaped sign in Figure 18a. It represents “Rettungsweg”, and is used to indicate the escape route. There is no existing model in RoadRunner, so we processed a made this model by ourselves in RoadRunner, making the available signs more diverse. In addition, we completed the surrounding natural landscape in Unity. The final result is presented in Figure 18

(a) Real view (b) Simulation in Unity

Figure 18: Case 2 Simulation of intersections near attractions

8 Learning Algorithm

8.1 Introduction

Another task involved analyzing and implementing a self-driving vehicle (SDV) algorithm using precise HD maps derived from real-world environments. This section explores how real-world driving examples were used as training data in [15] to develop a driving policy for an SDV through reinforcement learning. Our goal is to assess the algorithm's adaptability to a Unity game engine-based driving simulator.

8.2 Architecture

To achieve autonomous driving, a self-driving vehicle (SDV) must address two main challenges: path planning and object detection. Path planning involves determining the desired trajectory for the vehicle, while object detection entails identifying and recognizing various elements such as other vehicles, traffic lights, lanes, and general obstacles like roadwork sites.
In the study referenced as [15], real-world driving scenes in the form of high-definition (HD) maps were inputted into a simulator. The simulator then applied a driving policy to guide the vehicle along a specific path. The predicted path was subsequently compared to the path taken in the real-world example. By evaluating the deviation from the expert driver's example, the driving policy was adjusted accordingly. This iterative process involved feeding a substantial amount of data into the algorithm, enabling the SDV to learn and imitate the driving behavior of the expert driver.

8.2.1 Inputs and Outputs

In order to understand and implement a similar algorithm for incorporating self-driving functionalities in a vehicle, it is essential to be familiar with the given inputs and desired output. Determining the desired output is relatively straightforward, as it involves defining the vehicle's position in a specific state using X and Y variables, along with a yaw value indicating its facing direction.
The required inputs can be obtained by studying the l5Kit dataset used in [15]. They are presented in Table 1. The term "number of samples" refers to the maximum perception

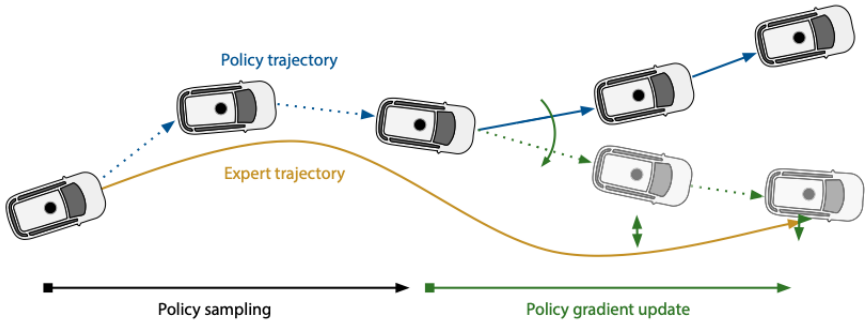


Figure 19: One iteration of policy update [15]

capacity, meaning that up to 30 other traffic participants can be considered within a single time step. Similarly, for the lanes, this means that up to 30 interpolated X and Y values representing the boundaries of a lane, corresponding to a certain lane length can be considered.

Table 1: Algorithm Inputs

Input	**Properties**	**No of Samples**
SDV position	X, Y, yaw pose of current point in time	1
Other Traffic Participants	X, Y, yaw pose of current point in time	up to 30
Lanes mid	interpolated X, Y points of lane center	up to 30
Lanes left	interpolated X, Y points of left lane boundary	up to 30
Lanes right	interpolated X, Y points of right lane boundary	up to 30
Crosswalks	polygon boundary of crosswalks	up to 20

8.2.2 Neural Network

To implement a self-driving vehicle algorithm based on HD maps and object detection, it is essential to analyze previous significant work in that field. Therefore, I examined the neural network architecture presented in [15], which is specifically designed for path planning. In order to incorporate all the features listed in Table 1, a graph neural network is utilized. The inputs are passed through a 3-layer PointNet network, resulting in a 128-dimensional feature descriptor. This descriptor is then aggregated using a scaled dot-product attention layer. Finally, a Multi-Head Attention layer outputs the action that the SDV should take.

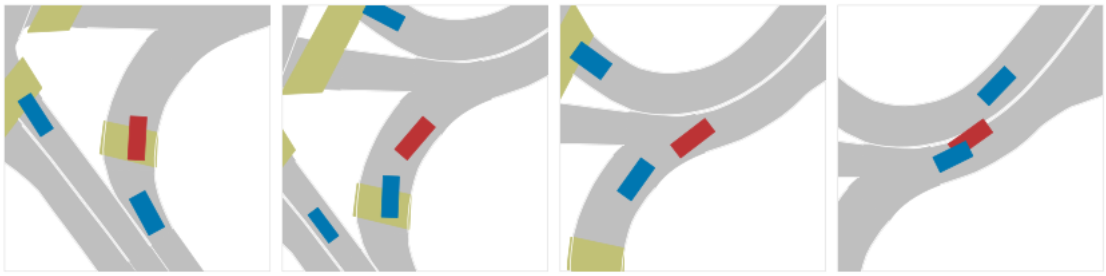


Figure 20: 4 graphical input examples. Red = SDV, blue = other traffic participants, grey = lanes, yellow = crosswalks

8.2.3 Object Detection

To perceive its surroundings, a self-driving vehicle needs to detect objects in its environment. State-of-the-art object detection models like YOLOv5 [16] already exist and do not need to be developed from scratch. For our work, we aim to incorporate real object detection within our Unity simulator. To achieve this, we need to attach a camera to our vehicle that is capable of capturing images of the environment. To that end, I have implemented a C# script to capture snapshots from our vehicle's dash cam.

8.2.4 Practical Implementation of Urban-Driver

To validate the work presented in [15], I attempted to run and analyze the provided Jupyter notebook within the l5kit repository. However, I encountered challenges related to package dependency versions, as the code was published some time ago. In order to execute the example in l5kit/examples/urban_driver/, I had to use Python version 3.7, Numpy version 1.19.5, and wheel version 0.38.4; otherwise, the installation would fail. After resolving the dependency issues, I was able to train the model using the provided example data.

8.3 Outlook

The ultimate objective of this project is to have a Unity-based driving simulator with self-driving vehicles trained on the proposed algorithm. The DI-Lab project aimed to understand and verify the algorithm's structure, inputs, and outputs, as well as to grasp the overall project concept and implement initial steps. Moving forward, as part of the "Forschungspraxis" phase, I will continue by simulating the bird's-eye view inputs depicted in Figure 20 within Unity. I will process these inputs and feed them into the algorithm to obtain initial training results based on our own simulator. Additionally, I will integrate a LiDAR sensor into Unity to simulate real object detection capabilities.

References

[1] *Mathworks RoadRunner*. https://www.mathworks.com/products/roadrunner.html.

[2] *UNITY ENGINE*. https://unity.com/.

[3] *Bayern Atlas and Simple 3D Architecture Models*. https://geoportal.bayern.de/bayernatlas/?topic=ba&lang=de. Accessed: 2023-07-18.

[4] *Unity Learn Platform*. https://learn.unity.com/. Accessed: 2023-07-18.

[5] *VectorZero*. https://www.youtube.com/@vectorzero1558. Accessed: 2023-07-18.

[6] Road and Transport Research Association. *Richtlinien für die Anlage von Landstraßen*. FGSV, 2012.

[7] Road and Transport Research Association. *Guidelines for the Design of Motorways*. FGSV, 2011. URL: https://www.fgsv-verlag.de/pub/media/pdf/202_E_PDF.v.pdf.

[8] *Unity Version Control*. https://unity.com/solutions/version-control. Accessed: 2023-07-18.

[9] *QGIS Homepage*. https://www.qgis.org/en/site/. Accessed: 2023-07-18.

[10] *EPSG:3857*. https://epsg.io/3857. Accessed: 2023-07-18.

[11] *EPSG:4326*. https://epsg.io/4326. Accessed: 2023-07-18.

[12] *DIN 1451*. https://en.wikipedia.org/wiki/DIN_1451. Accessed: 2023-07-18.

[13] *Mapillary*. https://www.mapillary.com/. Accessed: 2023-07-18.

[14] *google earth*. https://earth.google.com/web/@48.16450537,11.58641181,507.87739675a,169.90387884d,35y,0h,0t,0r. Accessed: 2023-07-18.

[15] Oliver Scheel et al. "Urban driver: Learning to drive from real-world demonstrations using policy gradients". In: *Conference on Robot Learning*. PMLR. 2022, pp. 718–728.

[16] Glenn Jocher et al. "ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation". In: *Zenodo* (2022).