Design and Implementation of a Teleoperator's Workstation

Duarte KEMPER
Joren GEERTS
Martin ULBRICH
David HONER

Supervised by:

Prof. Dr. rer. nat.
Toralf TRAUTMANN
Prof. Xavier MASIP
Dr. Eva MARÍN

08/02/2023 – 21/06/2023
—
EPS Project February 2023
—
EPSEVG

I. Foreword

We are a team of four European students who participated in the European Project Semester at the Universitat Politècnica de Catalunya in Vilanova I la Geltrú. The team members come from different countries in Europe. Duarte Kemper comes from France but is Dutch and studies mechanical and electrical engineering at the Ecole Nationale d'Ingénieurs de Tarbes, Joren Geerts comes from Belgium and studies electromechanics at Thomas More University in Geel. Martin Ulbrich comes from Austria and studies design and media at High School St. Pölten in Sankt Pölten and David Honer who comes from Ireland studies product design at the TU in Dublin.

Two of our mentors, Prof. Xavier Masip and Dr. Eva Marín work at the CRAAX department of the UPC, which is the Catalan acronym for Advanced Network Architectures Lab. Our head mentor is Prof. Dr. rer. nat. Toralf Trautmann who is from the MechLab department of the Hochschule für Technik und Wirtschaft of Dresden. He offered us this project in association with his lab. We will be working on this project for nearly five months. Other students will probably continue where we left off, as there is much more to it than what can be done in one semester.

Figure 1: European Project Semester Participants
From left to right: Duarte Kemper; Prof. Dr. rer. nat. Toralf Trautmann; David Honer; Joren Geerts; Martin Ulbrich

II. Abstract

The project aims to implement a way for a teleoperator to control an existing self-driving car if the autonomous driving algorithms fail to respond to the encountered situation. The project will rely on the existing code developed by the MechLab Team at the HTW in Dresden, who have converted a BMW i3 into a self-driving car using surround and proximity sensors and a homemade software that controls the vehicle's speed and steering. The car is also able to detect pedestrians and other obstacles thanks to a deep learning algorithm dedicated to this part.

Teleoperation systems pose many challenges, such as providing the teleoperator with the same level of situational awareness as a driver in the car. The driver needs to focus more on the surroundings, and therefore teleoperated drivers will have to rest more often and take more breaks. To address this challenge, the teleoperation system will use high information density sensors, including LiDAR, radar, and ultrasonic sensors, to provide the driver with an overlay of detected obstacles and the predicted path, enhancing reality to compensate for latency in communication by taking some workload off the operator.

Another big challenge is to switch between the autonomous and teleoperated driving modes, as there are different problems that can appear. Most noticeably, during the time it takes for the operator to get aware of the situation and respond to the call, the car must be able to safely stop and wait for instructions from the operator. The failure to do so could result in dangerous or even deadly situations for the autonomous vehicle’s occupants as well as for the other road users, who do not need to wait for the communication to be established.

One of the last great challenges is allow stable and fast communication between the car and the teleoperator. This can be achieved by narrowing the data transmitted for example by reducing video quality in predefined cases, or by ensuring redundancy in the communication media. Nevertheless, a complete loss of communication is not impossible, so a protocol needs to be defined in order to safely halt the vehicle while waiting on the reconnection of the transmission.

To fulfil this project, our team will use MATLAB and Simulink in combination with different toolboxes from the MathWorks company. We will try to develop a human-machine interface for the teleoperator, implement a way for the operator to take over control of the vehicle, build scenarios to test and simulate our different programs and much more. All of this is done in order to build safer and more reliable autonomous vehicles for the future.

III. Table of contents

IV. List of tables

V. List of figures

1 Introduction

Future transportation for both individuals and society will be largely dependent on automated driving functions. Remote monitoring will be necessary for the safe execution of these functions as well as to overcome complex situations which driving algorithms cannot solve yet. These ideas are referred to as "teleoperated driving", as the driver is not physically present in the car. In this report, we will expand further on this subject, trying to implement the technology in a car that is, for now, only able to drive autonomously in certain circumstances.

Since the beginning of the autonomous driving concept, manufacturers of such vehicles are continuously working on improving the self-driving algorithms. But even with the latest advance in deep neural networks, there are still a considerable number of situations where the human decision-making ability is superior. Therefore, the demand for teleoperated technology is increasing, as this means a human will take over control of the vehicle if the algorithm struggles.

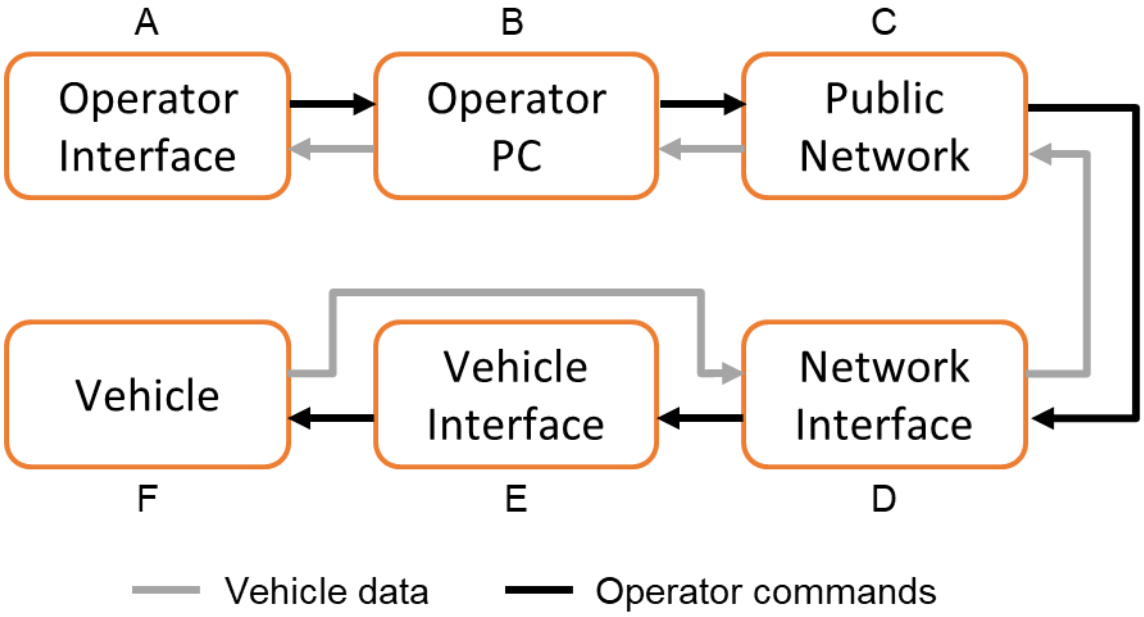


Figure 2: Bloc diagram of a simplified teleoperation communication system

In our case, a self-driving car developed by the HTW Dresden is already available. Our main objective is to implement a teleoperator control system in the car's algorithm as well as the operator's remote interface. In other words, the sections D, E and F are already existent, and they will only require slight modifications in order to work with our teleoperation system. Nevertheless, to allow us to test our system in a safe way, we will start by using software models of the last three elements provided by our mentor.

Because of the extremely high complexity of such a teleoperation system and the sheer amount of work that would have to be carried out to develop a commercial version, our project has been based on a limited and well stated number of tasks in order to build the base of the system. These tasks were selected based on their duration, which we estimated by deepening our knowledge in the required fields at first. This will allow future students to build upon our project, while still being able to present functioning concepts during our final defence.

In the first section of this report, we will develop the current state-of-the-art of teleoperation and related products our project is based on. This will give a better understanding of the sections A, B and C of the teleoperation system on which we are working. We will continue by explaining our different choices and uses of software as well as the varied reasons that got us there.

The following section will be dedicated to the first element A of the teleoperation system, where we used simple but effective ways to create a raw human-machine interface. After this, we will develop more about elements B and C as we encountered many issues when moving from the simulation to a real-world test. Finally, we will dedicate a section of this report to the procedure we went through in order to create realistic simulations to test our system in.

Meanwhile we made all our intellectual capacity available for this EPS project, a student at the HTW in Dresden has been working on developing a new vehicle interface as well as a network interface for their test vehicle in Germany. This should not impact any of our work, as we have not modified anything of those sections in a meaningful way.

2 Project statement

The final goal of our project is to implement in an existing self-driving car, a way for a teleoperator to control the vehicle if the autonomous driving algorithms cannot respond to the encountered situation. To fulfil this task, we will use the MATLAB and Simulink software, as all code is already written with these languages.

2.1 State of teleoperation technology

In robotics, a machine is called teleoperated when the control information it uses to achieve a task is not generated by itself, but by an external person or device. This information is then transferred to the working device, while the necessary information for the decision-making person or device is communicated back.

Teleoperation has already been in use for an extended period of time but mostly on a local scale, with remote controls for toys, devices or machines where the operator stands next to it. In the last two decades, teleoperation has been implemented in many other fields to enable operators to execute tasks while being located hundreds or even thousands of kilometres away.

After teleoperation proved its success in the military field, the medical field also began implementing it for different use cases. The biggest use is telesurgery, where a surgeon from a certain hospital can perform surgeries on patients from a different hospital without the need to travel the long distance every time. Also, teleoperation is now used increasingly to deliver meals to patients in hospitals.

Many different research projects by car manufacturers, universities and private companies are looking at implementing this teleoperation technology in vehicles, in order to hopefully facilitate future transportation.

2.2 Background

At the HTW in Dresden, Toralf Trautmann and his team have managed to convert a BMW i3 into a self-driving car by implementing a lot of surround and proximity sensors as well as a homemade software able to control the vehicle's speed and steering. Therefore, our project will heavily rely on the already available code developed by the team in Dresden.

As autonomous driving is a field of heavy research, the team at the HTW is continuously improving their technology too. But during this project, we will assume that the car as well as its software are not evolving enough to impact our upcoming and already fulfilled tasks.

Figure 3: BMW i3 automated by the MechLab Team on the test track in Dresden

2.2.1 Autonomous driving

Path tracking is a vital component of autonomous driving and most current path tracking research is based on different positioning sensors, such as GPS, cameras, and LiDAR sensors. The autonomous path tracking system consists of a differential drive kinematics module, pure pursuit module, a PID control module, and two actuators that control the steering and throttle as shown in Figure 3.

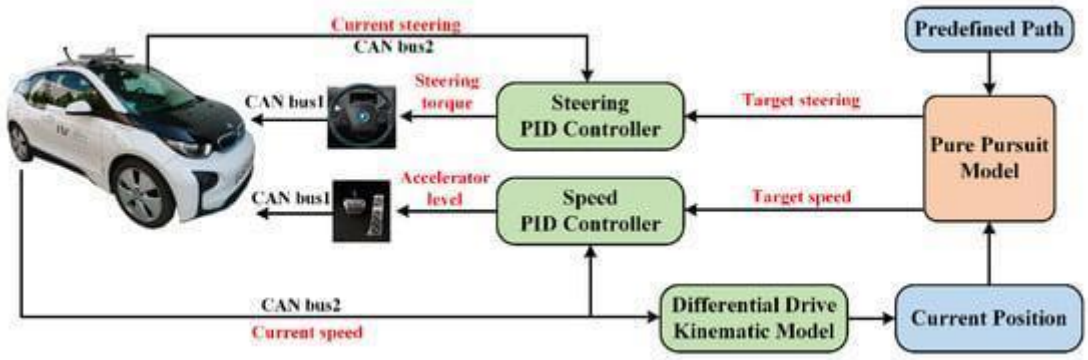


Figure 4: General overview of the path tracking algorithm implemented in the BMW i3

The differential drive kinematics model calculates the current position in relation to the original position based on rear wheel speed, which can be read from the CAN bus. The pure pursuit model then outputs the target steering angle and target vehicle speed based on the error between the target position in the predefined path and the current position. The target steering angle and vehicle speed are then sent to the PID controller to generate a steering and throttle command in order to drive the vehicle.

Recently, Dr. Cao, a guest scientist of the Hangzhou Jiliang University who works on autonomous path tracking and driving, came to visit the MechLab in Dresden. Once he returned to his home country, he successfully managed to start and stop the autonomous driving of the car in Dresden from China through a TCP/IP connection. This is really promising as we will also need to communicate between the car and a remote location for our project.

2.2.2 Pedestrian detection

In order to prepare their autonomous vehicle for real world tests, the team from Dresden has designed and implemented a pedestrian detection algorithm in the car. The video with the following link gives you a visual idea of the working of the car equipped with the pedestrian detection. https://www.youtube.com/watch?v=aGPtWeZi2o0

Figure 5: Screen capture of the video presenting the pedestrian detection algorithm from the HTW Dresden

In Figure 4, we can see the Livox LiDAR data in the bottom left corner as well as the real situation. On the LiDAR data we observe the two road marking stripes in blue, as the white paint is highly reflective. The less reflective parts are shown in colours closer to red. The detection of pedestrians and cars is based on a deep learning algorithm.

The LiDAR data is processed in real-time and outputs 3D "boxes" that enclose the detected obstacle. If there are obstacles on the trajectory of the autonomous vehicle, the path box will turn red, and it will apply the brakes in a controlled manner in order to stop the car before the obstacle. As soon as the path is clear again, the box turns green, and the car will start moving again. The pedestrian detection is running on a Nvidia Jetson Nano and is written in Python, whereas the driving is done by a Simulink program running on a laptop. The two computers communicate using the TCP/IP protocol.

2.2.3 Automated and teleoperated driving regulations

Nowadays, with many private and public companies designing and testing autonomous vehicles, the regulations must be adapted in order to control their appearance on the public roads. The German Road Traffic Act available in Appendix 1 includes provisions for the operation of autonomous driving vehicles in certain operating areas.

The main points contained are that such vehicles must be equipped with technical equipment that allows them to perform driving tasks independently, and to bring the vehicle into a minimum-risk state in certain

situations, for example when the system limits are reached, technical faults occur, or when the vehicle is deactivated. In order to still be able to qualify somebody as responsible for the different actions and accidents, a human person is still compulsory, but he can be located remotely. Known as the Technical Supervisor in the GRTA, he is responsible for authorizing driving manoeuvres, deactivating the vehicle and ensuring the safety of the vehicle occupants and other road users when active.

The GRTA also specifies requirements for the defined operating areas in which autonomous vehicles may be utilized. These areas must be locally and spatially determined public street spaces that meet the requirements set out in Paragraph 1e Alinea 1. The technical equipment of autonomous vehicles must be able to bring the vehicle into a minimum-risk state if the continuation of the journey is only possible by violating road traffic regulations. Also, the vehicle must be able to immediately report to the Technical Supervisor any impairment of any component whether it impacts its functionality or not.

These exhaustive regulations make it clear that for the moment, any autonomous vehicle must have an assigned teleoperator who can be held liable for the problematic situations or accidents caused by the car. This in turn means the teleoperator connection must be error proof and that such a connection will still be necessary for a long time.

2.3 Teleoperation

With teleoperation of a vehicle being an overly complex field, we will try to develop the most important parts of the system in the following section. For every part, we will also address the different problems that come along. Some of our tasks developed further down in this report consist of solving these problems, as they are the key to a flawless immersion in the remote driving experience.

2.3.1 Sensors and driver immersion

The main difficulty with teleoperation is to permit the Teleoperator to sense the vehicle's surrounding area in the same way as if he were in the car. This means that the driver needs to focus a lot more during his work to keep track of all the elements around the vehicle. In the end, the driver will be exhausted earlier than during real life driving, and therefore teleoperators will have to rest more often and take more breaks. Luckily, most cars that could use teleoperation are already equipped with a lot of high information density sensors for autonomous driving.

The minimal required information for the driver is a 270° field of view, centred on the front of the car. With this information, most of the components are visible, but because of the two-dimensional properties of screens, distance estimation is difficult for the driver. This could be solved by using other sensors available on the car, with for example radar or ultrasonic sensors. By overlaying this information on the screen, or near the camera views, the driver can simply read off the information, needing fewer mental efforts.

There is also a notable difference between a teleoperator using a conventional screen and a teleoperator using a head mounted display (HMD) as stated in [1]. The tests carried out show that the operators equipped with HMDs were, in general, prone to less collisions but had worse distance estimations. This is mostly due to lack of training as the majority of participants did not have prior experience. Therefore, HMDs could help in the perception of the vehicle's environment for the operator.

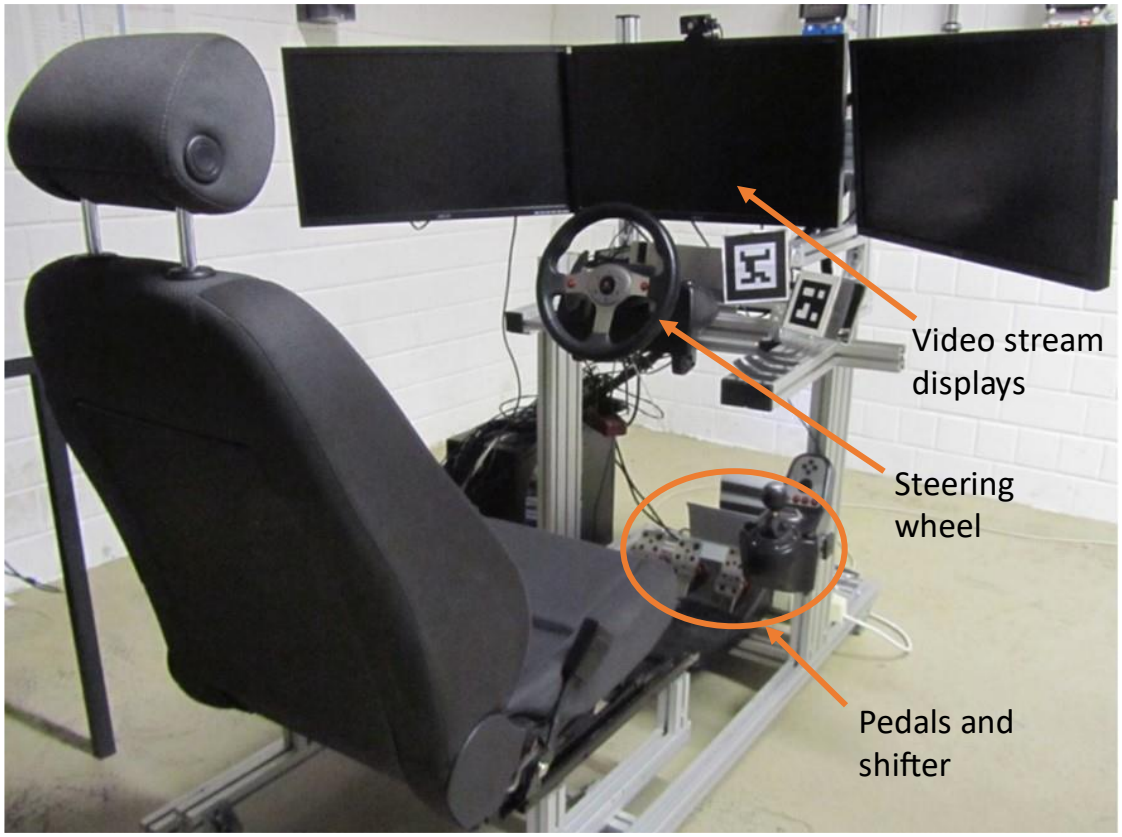


Figure 6: Typical setup of a teleoperator's workstation (taken from [1])

Also, most cars are equipped with at least one simple type of LiDAR, providing a 3D point cloud of the environment. A 3D point cloud can be seen as a set of data points in space that represent where the sensor measured the object's surface. These data are normally processed by the autonomous driving algorithms, but during teleoperation, the data could be processed by another algorithm to overlay the different detected obstacles on the video and therefore helping the driver in focusing on suspicious elements in the environment.

Finally, to compensate for the latency of the communication between the vehicle and the teleoperator, a trajectory predictor could be used, overlaying on the video the path the car will follow with the current steering angle and speed, creating enhanced reality for the operator. The operator would then be able to steer according to the planned path and not the vehicle's position. Another addition would be the implementation of a free corridor, which is the distance travelled by the car if it would need to come to a sudden stop and should therefore always be kept free by the driver.

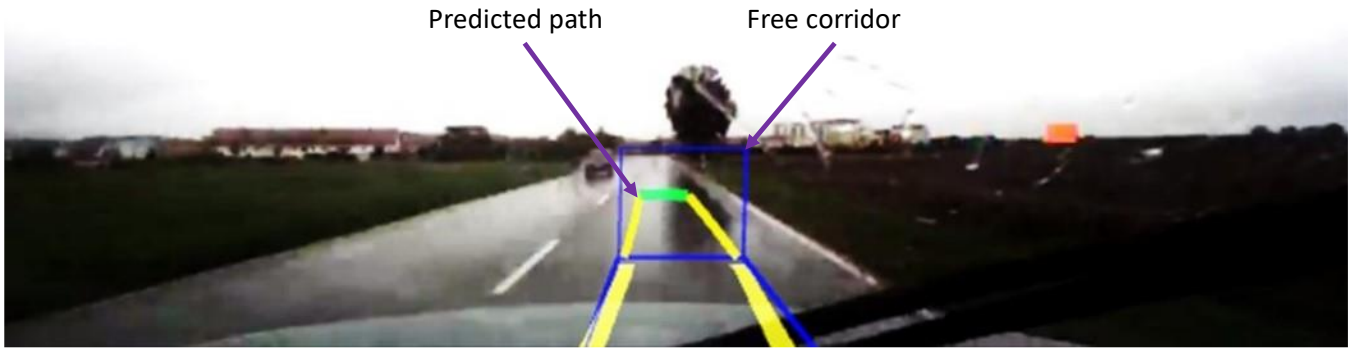


Figure 7: Example of a trajectory prediction overlay and a free corridor for emergency braking (taken from [1])

2.3.2 Switching from autonomous to teleoperated driving

In order to get assistance from a teleoperator, the autonomous algorithm first must make a teleoperation call, which is answered as soon as a teleoperator is available. The problem appears when such a call is initiated in a high-speed driving scenario. In a big teleoperator centre, the time elapsed between the algorithm making the call and the driver answering to the call could be small due to the high number of operators. But the first thing he must do is get to know the situation that causes trouble.

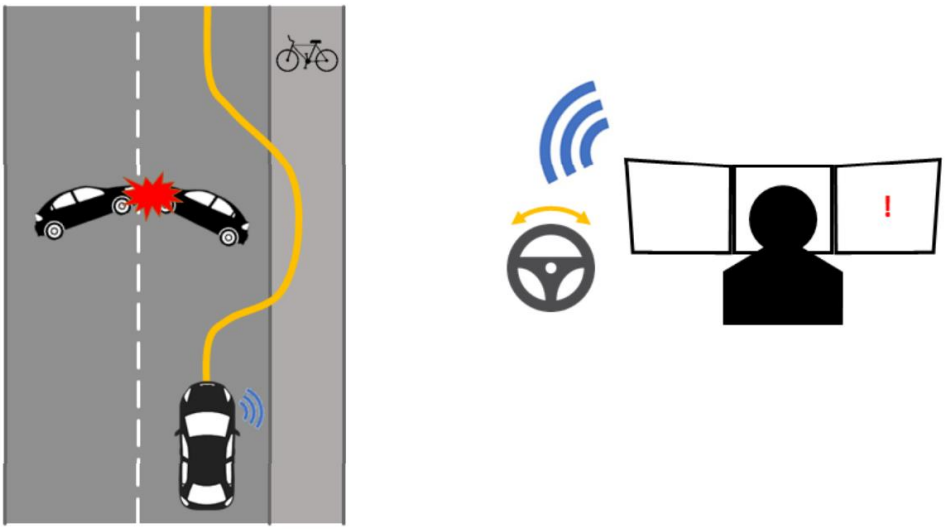


Figure 8: Visual representation of a situation responsible for a teleoperator call (taken from [3])

Therefore, the time spent between the start of the call and the operator acting on the car could be quite long, and at higher speeds, the distance travelled by the vehicle gets impressive. To remedy this, the car's software

must carry a protocol that securely slows down or even stops the car before control is taken over, to allow the operator to familiarize with the encountered situation as can be seen in Figure 8.

This is a critical part for autonomous and teleoperated driving as a badly implemented protocol could lead to very dangerous situations for occupants and other road users able to respond to the unforeseen situation without latency.

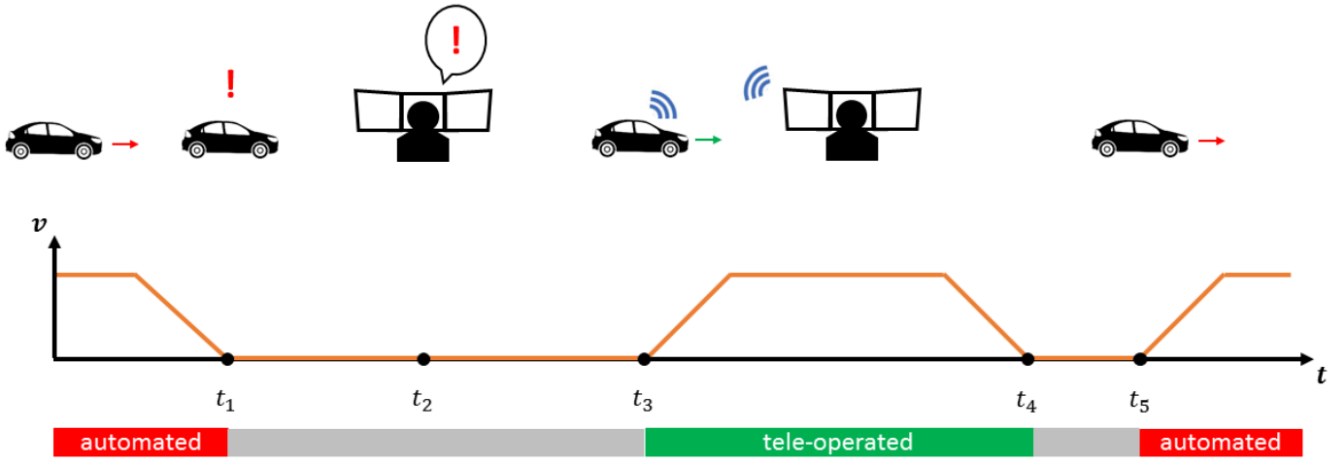


Figure 9: Typical scenario of the takeover from automated to teleoperated driving (taken from [3])

2.3.3 Network and communication

No matter what type of teleoperation is used, there will be a part of the system where data must be transmitted between an operator and a machine. In the case of teleoperated driving, control inputs from the driver are sent to the car, and the car transmits data from its sensors to provide necessary information to the driver. This in turn implies that the car and operator's workstation need to be able to communicate without issues.

Therefore, teleoperated vehicles are equipped with wireless communication systems, of which most are using the 4G and 5G mobile network. Nevertheless, these networks typically experience variable throughput and latency and uneven coverage. Because of these reasons, different solutions could be implemented to ensure better communication between driver and vehicle.

One of such solutions for example consists in utilizing two networks at the same time (with two sim cards from different providers) and so allowing for communication on the network with the highest speed and best coverage. It also provides redundancy in the case of a failure in one of the operator's networks or other issues that might occur.

With teleoperated driving focusing mostly on providing the operator with a live view of the vehicle's surroundings, another solution proposed in [4] tries to process the acquired video data in real-time before transmitting it to the operator. This method could allow the dynamic modification of the resolution of the video stream based on available network bandwidth. The idea is that the algorithm selects different sections of the video data that are considered regions of interest for the operator and compresses the leftover parts to a lower resolution.

3 Research and courses

The first weeks of our project were mainly dedicated to the seminars of our EPS course. The time that was left we used to do some research in order to better understand the difficulties we were bound to encounter in our tasks and therefore have a better judgment on the feasibility of the goal. This time was also used to familiarize with MATLAB and Simulink, as only one of us had some but limited experience with the software.

3.1 3D BMW model for simulation

One of the proposed tasks was to create a model of a BMW-i3 to use during simulations in order to make them closer to reality because this is the car used by the HTW Dresden. After doing a lot of research on this subject, we mentioned that there was a high probability this task would be really difficult to complete. This is because there are a lot of different steps that must be carried out on different and complex software, and in turn, it would be extremely time consuming.

3.1.1 BMW model for the Simulink model

After spending a solid amount of time researching the workflow needed in order to be able to import the car model in our Simulink model, we have come up with the main steps that would have to be gone through in order to import the car in the simulation software. Beneath is an overview of the different steps that must be accomplished.

First, a 3D mesh file of the BMW-i3 must be created. This is a file with the different outside surfaces the car is composed of, to enable the simulation to detect collisions between the vehicle and its surroundings. There were some high-resolution ready-made files on the Internet, so this part was quickly finished.

For the second part, we would have to setup the bone hierarchy of the mesh using a free and open-source software called Blender. This procedure basically describes which parts of the car are linked together and how. For example, the front left wheel of the vehicle is linked to the body, and can move up and down because of the suspension, and turn left and right with the steering.

Table 1: Table of the mesh parts for the vehicle

Vehicle Part	Name
Chassis	VehicleBody
Front left wheel	Wheel_FL
Front right wheel	Wheel_FR
Rear left wheel	Wheel_RL
Rear right wheel	Wheel_RR
Steering wheel	Wheel_Steering
Left headlight	Lights_Headlight_Left
Right headlight	Lights_Headlight_Right
Left indicator light	Indicator_L
Right indicator light	Indicator_R
Number plate	Vehicle_Plate
Brake lights	Lights_Brake
Reverse lights	Lights_Reverse
Front left brake caliper	BrakeCaliper_FL
Front right brake caliper	BrakeCaliper_FR
Rear left brake caliper	BrakeCaliper_RL
Rear right brake caliper	BrakeCaliper_RR

Having done the research, we know how to accomplish this part, but we have not tried to do it yet, as it is probably the most time-consuming part of the whole task. In fact, the downloaded mesh must be divided in separate parts in order to make a bone hierarchy out of them. In Figure 9 you can see the various parts that have to be created and what names they should be given in order for the program to recognize them.

The third step consists in importing the file in Unreal Engine. This program is a game engine used to develop multiple computer and video games. As we were only trying to gauge the amount of work required for this task in particular, we decided to try and import the skeletal mesh with only the VehicleBody defined. Unfortunately, this is where we started to encounter issues.

During setup of the MATLAB plugin for Unreal Engine, when trying to activate the plugin, an error message stating that the plugin was not compatible with either the Unreal Engine version or the operating system would appear. After spending a lot of time trying to get the plugin to work, we reached out to MathWorks, the parent company of MATLAB, for help.

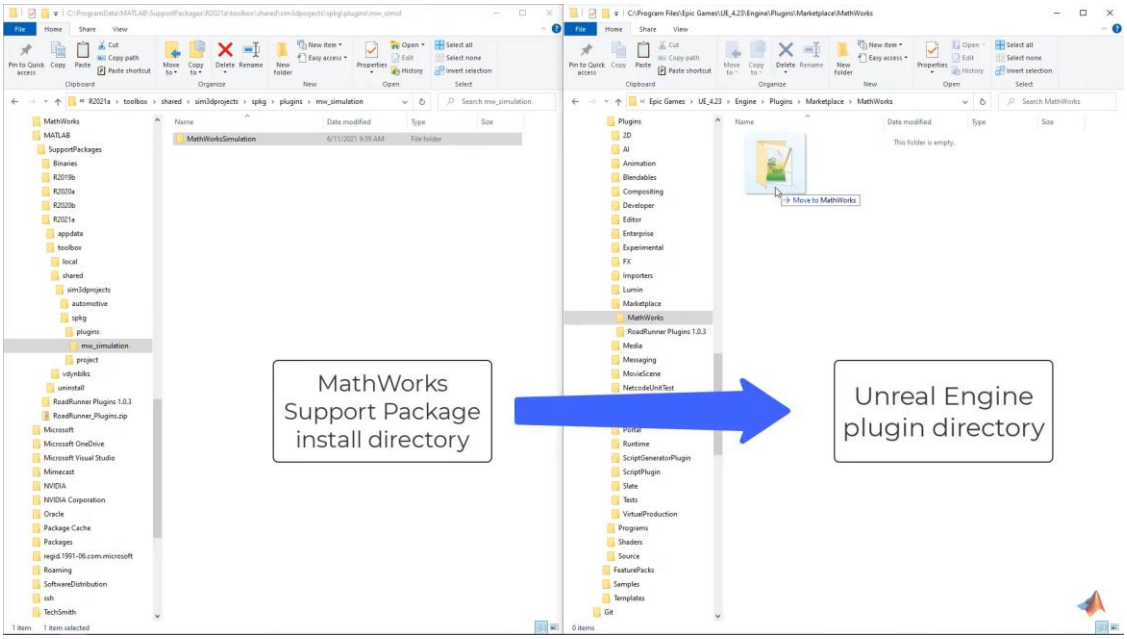


Figure 10: Screen capture of the tutorial video on how to use Simulink with Unreal Engine

After thoroughly going through the different steps included in their response, we managed to get the plugin working on only one computer. At this point, even if we would have been able to get it to work properly on all of our machines, we decided that this task was too difficult and would take too long to achieve only a small part of the project.

The last step would have been to import the car model in Unreal Engine through the help of the plugin, but this was not even tried because having it operational on only one computer was too restrictive. Not working on this task will only impact the aesthetic part of the project. In fact, this visualization is solely to be able to simulate the situation as closely as possible to reality, as sensors can pick up all the different obstacles and objects in the scene.

At the end of our project, with much more knowledge about Unreal Engine and the different steps required to export scenes, we decided to try one more time to create a model of the vehicle to be used during the Simulink simulations. After running an additional command in the MATLAB command window that we were not aware of was needed, a folder containing an unreal project example was created. This finally allowed us to follow the workflows stated above, without encountering more issues.

With these steps now completed, we imported our vehicle model while ensuring to select the right options in the Editor. Once the import was completed, we could export the vehicle in order to use it in Simulink, without forgetting to select the right path in the Simulink blockset. At last, we were able to use our BMWi3 model for our visualisations as will be shown later in the report.

3.1.2 BMW model for RoadRunner

As has been explained in the section above, we failed to prepare the BWM-i3 model to use it directly in the Simulink models. At this time, we received an email from a MathWorks France employee who asked if he could learn about our project, it being closely related to his field of work and knowledge. As most of our work needed to be carried out on MATLAB and other products from the company, we were very enthusiastic and offered him to join one of our weekly meetings.

During this meeting, we explained our different goals of the project, primarily the fact that we were looking at modelling Vilanova i la Geltrú for our simulations and the issues we encountered trying to prepare the vehicle for Simulink. Seeing what we were trying to achieve, he advised us to use a different workflow, using a different software called RoadRunner.

To import the model of the car in RoadRunner, the only mandatory steps were to scale the 3D model and orient the coordinate system the right way in Blender. Then, we exported the model to an .fbx format and simply opened it in RoadRunner, resulting in the view below.

One thing to note is that we did not create a bone hierarchy for the car, so it is moving as a solid 3D object, without spinning wheels or effective suspension. In our case, the model is usable as is, but can always be perfected later to create more realistic simulations if desired by future students.

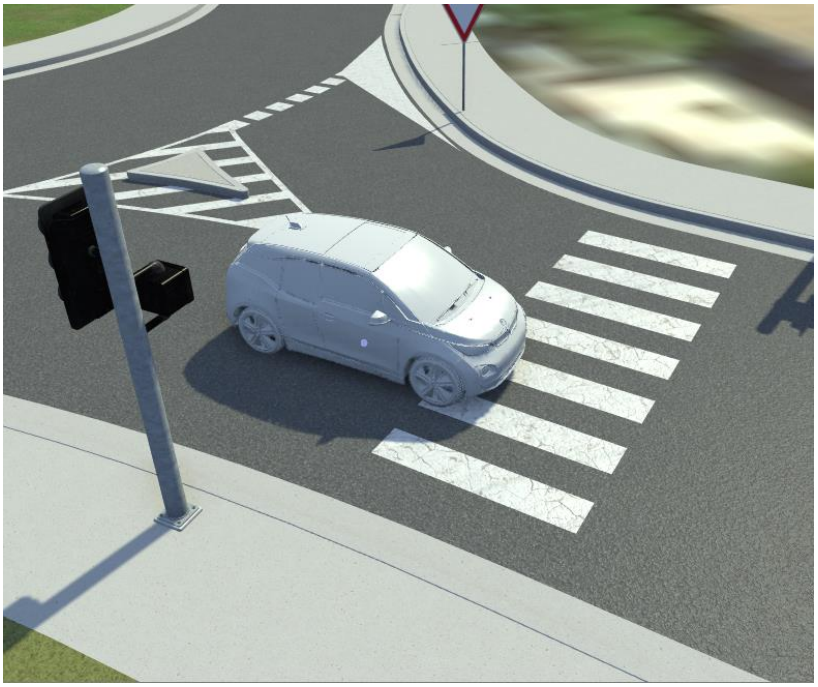


Figure 11: BMW i3 model imported in RoadRunner

3.2 Design of Vilanova i la Geltrú

During the first weeks of our project, we quickly came to the conclusion that realistic 3D visualisations would be possible to make with the tools we had during this project. Therefore, we first thought of modelling the city's road network using the Driving Scenario Designer. This is a toolbox that comes with Simulink and allows

quick and easy design of simple road networks on which certain driving scenarios a real car might encounter can be modelled, albeit without the surroundings.

3.2.1 Vilanova i la Geltrú in Driving Scenario Designer

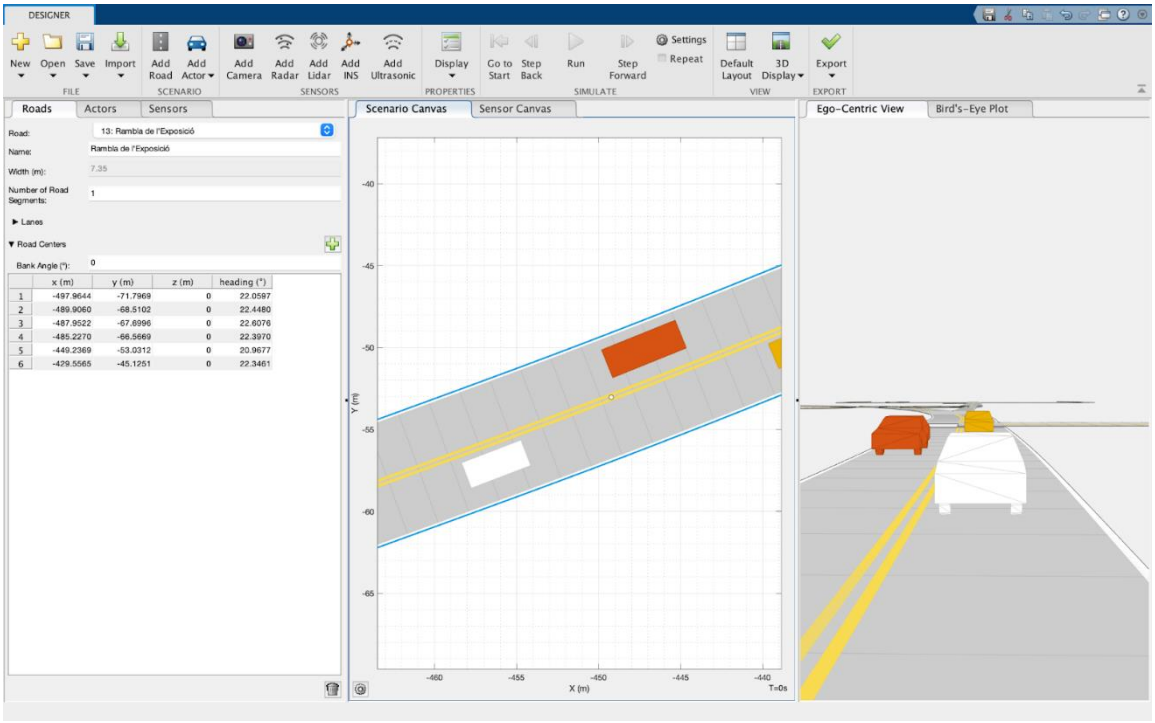


Figure 12: Typical interface of the Driving Scenario Designer, with the trajectory planner in the middle and the 3D visualization on the left-hand side

The Driving Scenario Designer toolbox is compatible with the OSM data format. This is the format of OpenStreetMap, a free and open world map database built by a community of volunteers. These maps are not fully accurate but convey the necessary information to create a crude road network. After exporting the map from the OpenStreetMap website, it can be imported in the Driving Scenario Designer. During import, you can select the different roads and streets you wish to use, and they will be inserted in the designer with their associated name and lane number.

This allows for a rapid design of a section of the Vilanova i la Geltrú road network, but some work is still required. Because the OSM format does not convey the elevation of the different streets, they are all imported on the default plane. Every section of road can be individually modified to make it closer to the real-life situation. Also, the junctions between the roads often need a lot of work to make everything blend in the right way.

In the Driving Scenario Designer, there are additional objects and actors, like bikes, pedestrians, guardrails, cones, etc. Still, there are no sidewalks available to be detected by the different sensors because there is no default "ground" where there are no roads. This could possibly be solved by adding slim roads to the side of the main road, which will act as sidewalks and interact with the sensors.

However, it should be noted that if the sidewalks are raised to their appropriate level, between five and ten centimetres above the road, a gap appears between the two objects, as they are only 2D representations. This

could potentially influence the different sensors, and therefore, testing should be carried out in order to validate the correct working of the simulation in this configuration.

This Driving Scenario is also able to crudely simulate sensors like LiDARs, ultrasonic sensors, and cameras which can all be observed during the simulation from the bird's-eye view window. Therefore, we should have been able to replace the default maps used for 3D visualisation with ones created with this toolbox.

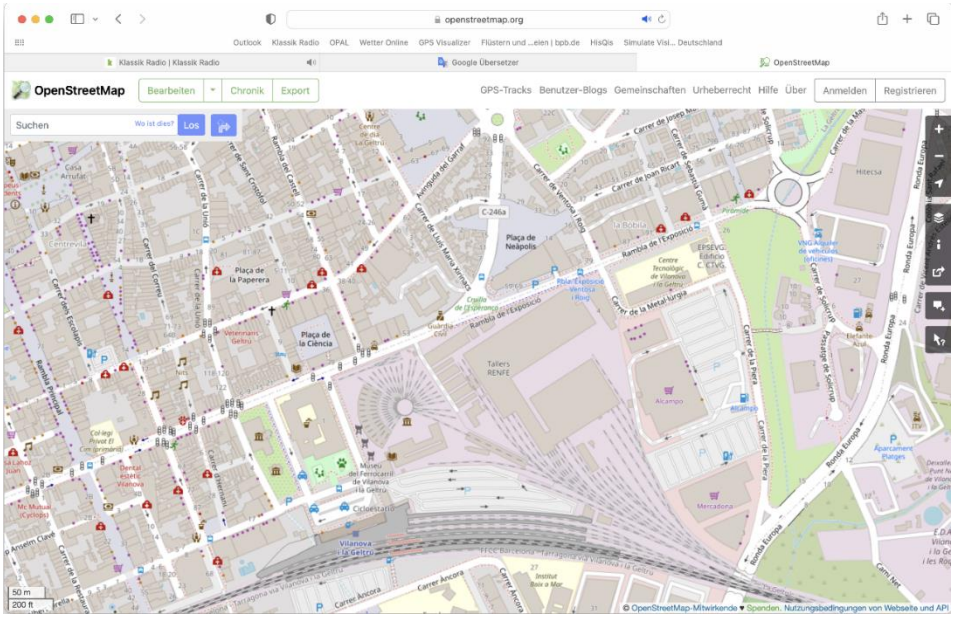


Figure 13: A view of OpenStreetMap looking at a section of Vilanova i la Geltrú

At this time, we only had tested the design part of the toolbox but already saw this would be extremely difficult to make something of good quality. The whole purpose of this section was to include real scenarios in the Simulink simulations, in order to try out the switchover from autonomous to teleoperated driving in different configurations and therefore decided to stay with the default Simulink maps.

3.2.2 Vilanova i la Geltrú in RoadRunner

But thanks to the advice from our MathWorks contact, we decided to try out RoadRunner because it would, in theory, speed up our workflow while at the same time rendering much better results for our simulations. However, with less than half of the project time left, he was nice enough to give us some crash courses in order to quickly get familiar with the software.

RoadRunner is a software created by MathWorks specially to create custom scenes and scenarios as close to reality as possible, to use during simulation of autonomous vehicle software or other driving-related matter. Our MathWorks contact showed us the different import possibilities to facilitate our work, as well as the main functions of the software.

To validate this method, we first imported an image downloaded from the Instituto Geográfico Nacional [5], Spain's national website where all the land and map related information can be found. This image is of remarkably high definition, so we could easily use it to layout the road network. It could then also be used to locate the different markings, signs, as well as all the props like trees, bushes and so on in order to make it as realistic as we could while not spending too much time on it.

Figure 14: The RoadRunner user interface, with the imported aerial image later used for the scene design

Once we tried out the ease of use of the program, we quickly came to the realisation that it was indeed going to save us an incredible amount of time. Now that we had an easy-to-use software to create 3D scenarios, we could finally create good visualisations while at the same time simulating the sensors used for driving the car, it being autonomously or with teleoperation.

3.3 MATLAB courses

Because of the team's limited knowledge about MATLAB and Simulink, our professor recommended to take some online courses available on the MathWorks website. The courses he recommended us to take were the MATLAB & Simulink Onramp courses.

The MATLAB Onramp course was a basic introduction of how MATLAB works. This tutorial is designed to familiarize you with the different capabilities of MATLAB and allows you to write some basic code in order to better understand the more complex examples and functions of the language.

In the same way, the Simulink Onramp course was a basic introduction to get started with Simulink. In the course they explained some commonly used blocks and workflows which made us able to understand the very complex graphs from the HTW team we will have to work with.

Finally, some time was spent on small home exercises from our mentor to learn the basic workings of different Simulink Toolboxes we will use during our project before really diving into it.

4 Human-Machine interface

The Human-machine interface (HMI) is the system or interface that allows communication and various interactions between the teleoperator and the autonomous vehicle. It is the various elements and technologies that allow the operator to monitor and control the vehicle's functions remotely when needed.

In this case, this human machine interface is in the form of a digital cluster. The digital cluster incorporates visual displays to provide the operator with real-time information about the vehicle's status which can include speed, battery level, navigation instructions, and other relevant information in regard to the status of the car. In the real-use case, this display will be in the form of an augmented reality interface provided to the teleoperator but can also be in the form of traditional screens and heads up displays which are more practical in this scenario.

There are various options in which the teleoperator will be able to work with the HMI such as touchscreens, buttons, knobs, or even voice commands but in the real-use case a full driving simulator would essentially be the controller of the vehicle as well as the commands on the HMI. The simulation is controlled through the start and stop functions on the cluster and then transferred to an external device when it needs the teleoperator to take over. The HMI could also provide the alerts and warnings in situations where the teleoperator is needed, whether that is through lights, sound and audio or something tactile like vibration.

Due to the transfer of data being so light, mainly the speed of the car, there is great possibility for the use of cellular networks such as 5G networks. It is well-suited for transferring low-bandwidth data with good reliability and wide coverage.

4.1 MathWorks App Designer

To create the HMI MATLAB offers a toolbox called App Designer. This can be used to create various interactive apps without the need for extensive knowledge in programming. Although it is not primarily designed for creating car clusters, its functions can be adapted as needed to fit the specifications of the teleoperators screen.

The basic layout of the cluster is achieved through simple drag and drop of the various blocks available. Some which include controlling functions of the chosen project while others only displaying information of the project. Its intuitive layout helps aid in creating an equally user-friendly and intuitive layout of the cluster.

The App Designer allows for real-time data processing collected from external sources such as sensors. The code is loaded from the Simulink simulation into the code of the HMI through various options. The LiDAR sensor can be used to collect the speed of the vehicle and then transferred to the cluster with the use of integrator blocks then adding derivative blocks to calculate the speed based off of the changing position of the vehicle. This data can then be transferred to MATLAB through TCP/IP communication which is then received and displayed using the call-back function in the app designer.

4.2 Design rationale

It's important for the HMI design to focus on user experience, simplicity, and intuitive design, considering the future potential complexity of the vehicle's functions and the teleoperator's controls. Clear and concise information is presented only, with the well-designed controls are key to ensuring safe and efficient teleoperation

of the autonomous vehicle. Only the most basic controls are necessary at this stage. Start and stop functions are what are used to initialise the processing of the speed data being received from the simulation. This can be initialised either when the teleoperator needs to take control or can be left running while the vehicle is in autonomous mode. Other considerations such as lights and indicators are included to ensure the vehicle is performing safely on the roads, being a fail-safe in which the teleoperator could notice and report.

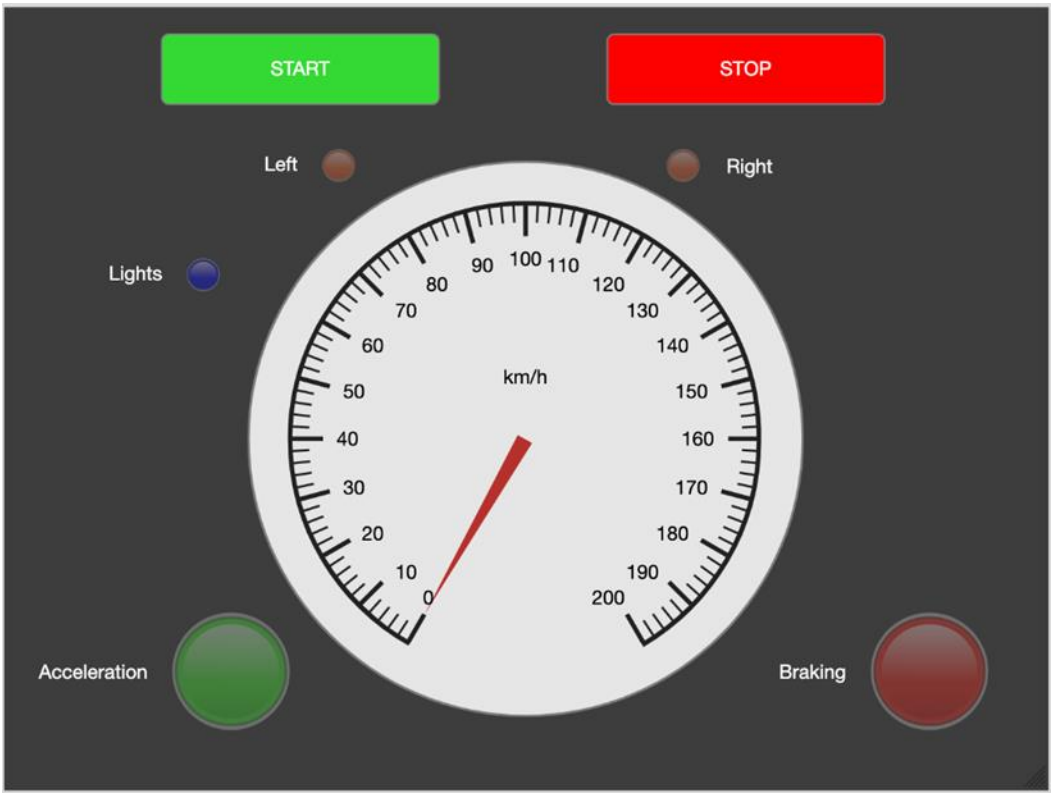


Figure 15: First iteration of the HMI used for testing

4.3 Simulink and App Designer integration

The initial goal was to have the functionality and aesthetic of the app designer work in tandem with Simulink's simulation. Due to crossover connection issues and incapability of blocks the final goal was not able to be completed as intended, however it is theoretically possible with some more time and some fine tweaking.

When working directly with the app designer itself, the design interface is easily operable with the ability to make quick adjustments. Issues arose when working with the callback functions, which is what allows the code to be altered so as to have an animated interface. Unfortunately, when working with the callbacks, it does not allow the altering of all the code with some being predefined based on the components that have been added. This first became an issue when pre-testing the gauge or in this case the speedometer. Initially, the LiDAR was thought to be the best option for determining speed, but it was discovered to require many more components which would take time to research each individually and how they would work when linking them together and to app designer. As sine wave blocks are being used already in the Simulink model, it is possible to use this to determine the speed while also linking it to another sine wave block in the interface. The issue that arose was the inability to add a sine block to the callbacks in the appropriate position due to the predefined code. With some work it could be potentially possible to use an additional component as a dummy block such as the button block to imitate the sine wave block, however it could cause more issues when linking it to Simulink.

It was still the goal to stick with the sine block so a usable workaround was to use MATLAB editor which could be used by inserting the code from the design of the interface but without the functions. Matlab editor allows a complete edit of all the code, meaning the sine block needed could be inserted in the code in its correct position which led to the movement of the speedometer from 0-30 reflected in a manner of a sine wave. This meant that the two sine blocks from both the editor and the Simulink could theoretically be linked to display the speed data.

More issues arose then when attempting to link them together. Although the 'from workspace' block on Simulink is supposed to work by pulling the data from whichever file you desire. The block could in fact find the file but wasn't compatible once linked, showing errors and not displaying the data, potentially due to missing blocks connecting the from workspace block, since the code was functioning, and the block was pulling the information in so it seems that's the only plausible issue which could be resolved in future project teams.

The functionality can be simplified by performing all animation in Simulink itself as it has a gauge, but only that, in terms of design. When connecting a sine block alone it does not function even though it shows no errors, this became a common issue with repeated trial and error with various blocks. Eventually, the addition of a gain block and an input1 block in combination with the gauge, it began to function. The settings can then be modified to adjust for speed and the limits of the needle. This means that by connecting these two blocks with a gauge in the actual Simulink model it should display the real-time speed of the vehicle.

This won't produce a usable interface for the teleoperator, however it's a proof of concept and would still provide enough data to make an informed decision when controlling the vehicle even if slightly less user-friendly. This clarifies that all work thus far is capable of being used to create a more aesthetic and better arranged design for the teleoperator, once the linking issues are resolved.

5 Teleoperation switchover

As has been stated in section 2.3.2, the switchover from autonomous driving to teleoperation is a difficult procedure because of all the different parameters involved. In order to make the changeover as safe as possible, different procedures must be implemented as well as a backup in case of failure of any part of the system.

5.1 Algorithm failure detection and switching

To initiate the changeover between the driving modes, there first needs to be an input signalling that the software is not able to make an appropriate decision to drive the vehicle in the encountered situation. But because the delay that will take place before the operator is actually controlling the car while being aware of its surroundings, two situations can be encountered.

As soon as the algorithm detects its inability to continue, the call to the teleoperator is initiated. While the car waits for the input of the operator, it will try to move out of the lane in order to come to a stop in a safe place (emergency lane on the highway for example). If there is no room to stop in a safe place, the vehicle will need to perform an emergency braking while minimising the deceleration in order to allow the cars behind to slow down as well.

When the operator finally takes over the commands of the vehicle, he can drive the vehicle around the problematic situation while taking all the necessary decisions and precautions. With the vehicle past the trouble zone, the operator can press a button in order to give back the control to the self-driving algorithm. This might be done while driving, as the program is still running, but it will need to be tested out to ensure this possibility.

5.2 Safety backup protocol

As has been stated before, the fact that the driving of this vehicle is based on technology can lead to numerous problems which means there has to be a safety protocol that can be deployed if the communication is lost during teleoperation or switchover.

This protocol will be very similar if not identical to the one that will be activated when the algorithm struggles, as the car will have to come to a halt as soon as possible, but without endangering its occupants or the vehicles around it.

In order to detect the connection loss between the operator's workstation and the vehicle, the simplest way is to check the '*client.connected*' bit which returns a *1* when the connection is established and a *0* as soon as it is lost, no matter what the cause is.

5.3 Integration in the Simulink model

With our project running out of time, we have not yet tried to implement the algorithm failure detection, as this implies that we must first understand the path-generating algorithm in order to add this feature. Nevertheless, we were able to simulate this input in Simulink by importing a step signal, which can be parametrised to change state at a certain time period, thus mimicking the signal coming from the code.

For this proof of concept, the signal is combined with the one carrying the state of the connection between the vehicle and its operator, as either one of them should enable the same response. In our case, we only implemented the emergency braking procedure, as the other option requires a deep knowledge about the driving and obstacle detection algorithms.

The end result is presented in Figure 16. If the Algorithm status signal is triggered, the system will switchover from the autonomous input to the teleoperated input. But in case of communication failure, the connection status will be triggered false and therefore change the commands from teleoperation to the default values in the three squares in order to stop the car quickly. This will have to be perfected, as we only managed to create a proof of concept with a very rough response that could be smoothed out by using complementary functions.

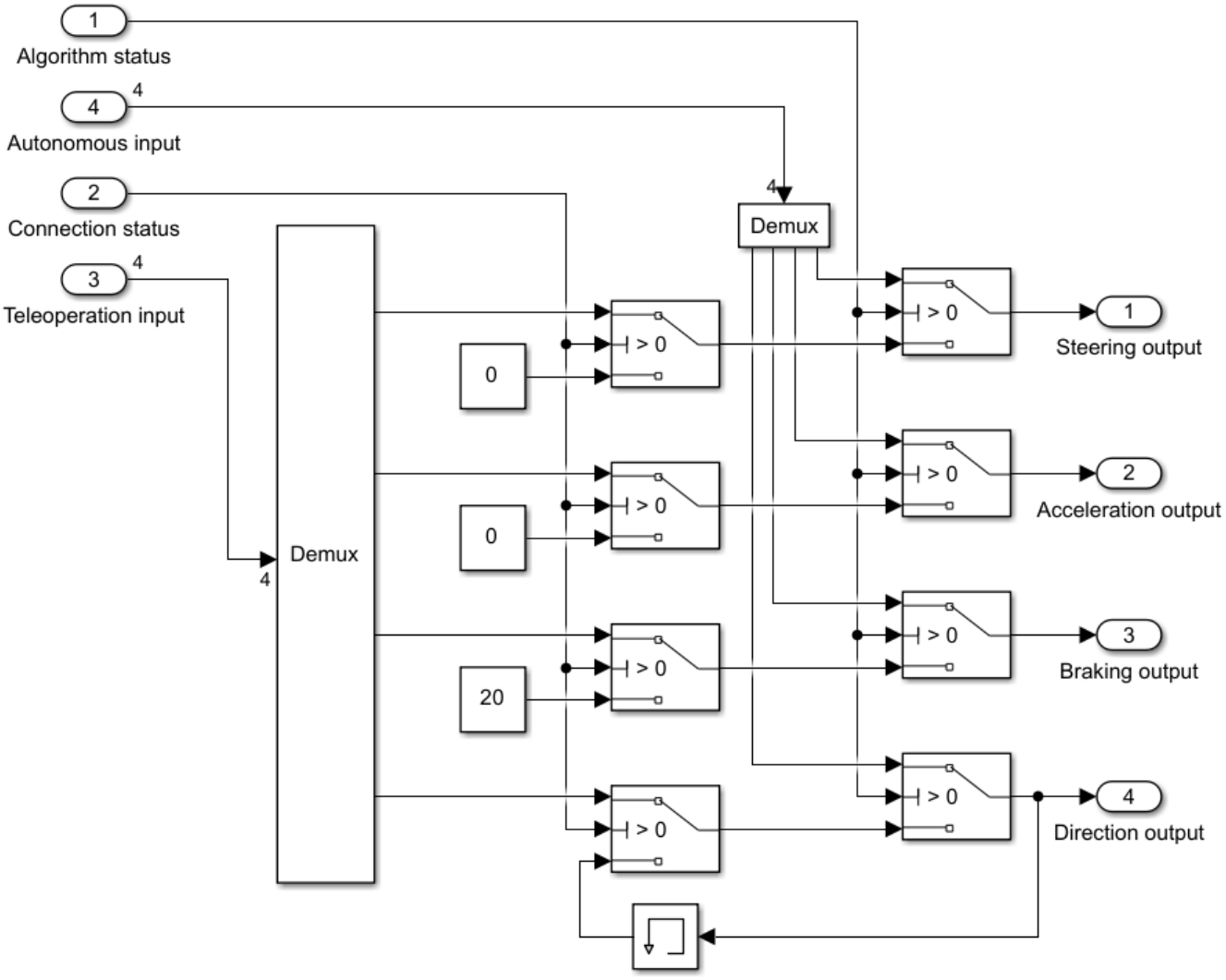


Figure 16: Fail-safe subsystem developed for the teleoperation switchover

6 Bidirectional communication between vehicle and operator

To allow remote operation of a machine or vehicle, there are two main data streams that must be transmitted with the least latency possible for the operator to maintain good control of the vehicle. For teleoperation of a car, video data of the surroundings is transmitted to the operator while the commands are transmitted to the car.

6.1 Transmission theory

Nowadays, bidirectional communication is used in the vast majority of data exchanges, so the technology is nothing new. With the advent of the world wide web, it has become extremely easy to communicate between distant machines very reliably and with low latency in most cases. However, the biggest challenge remains video data transmission, because of the vast amount of data that has to be sent continuously in order to obtain a free-flowing video at the receiving end.

The theory is to transmit in real-time a video stream from the car to the operator and the different commands from the operator to the vehicle. At first sight, this does not seem exceedingly difficult. But as will be explained further, we will quickly meet the limits of the standard transmission protocols and therefore, we will only present a proof of concept, that will need to be optimized in order to be used for real teleoperation.

6.2 Operator to vehicle communication

For our project, our mentor Toralf suggested us to use the TCP/IP which stands for Transmission Control Protocol/Internet Protocol, as it defines the majority of the software and hardware layers used for communication by combining both protocols. This protocol allows for end-to-end communication between two devices interconnected by networks around the globe. The main device that hosts the communication is called the server and the device that connects to it is called a client.

The IP part of the communication defines how the information is routed through the networks to ensure delivery, whereas the TCP part defines how the data is handled by the sender and receiver in order to allow correct assembly and disassembly of the information.

From the operator to the vehicle, the data to be transmitted is very light, consisting of only a few 8-bit integers representing the different commands like steering angle, throttle position, brake pedal position, selected driving mode (park, neutral, reverse, drive) and the state of the different light and wiper switches. All this data represents around 100 bits, less than 10% of the maximum that can be transmitted in one packet. This in turn means that the packet of data will be sent faster, as there is less data to transmit and receive.

To try out this first part, we built two simple Simulink models. The first one is a simple model which reads the input values of a joystick connected to the computer and packets the signal values in order to send them through its client connection to the other machine. The second model is composed of a 3D scene configuration block, a 3D vehicle block, and obviously the receiving server side of the communication with the required signal conditioning blocks for the 3D vehicle to work with them.

This simple simulation worked well as our two machines were connected to the same network, which will never be the case in a real-life situation. For that reason, further testing by a future EPS team should be carried out between different networks, with a wireless connection preferably to be the closest to reality possible. During our test, the driver had to look directly at the 3D scene to know where to go with the car. Therefore, our next step was to implement the live video feed from the vehicle to the operator.

6.3 Vehicle to operator communication

To allow for the teleoperator to drive the car as in a real scenario, a live video stream coming from the vehicle's interior must be transmitted in real-time. This allows the driver to know the car's surroundings and control the vehicle based on it.

6.3.1 TCP/IP connection

As we had already managed to use the TCPI/IP for the driver commands, we first tried to use it for the video data transmission as well. Communicating with TCP/IP offers quite some advantages, as it functions independently of the operating systems of the devices, which means there are no incompatibility issues. It also supports many routing protocols to allow a secure connection, called a socket, that is maintained until the server or client part is stopped. This socket communication is also very lightweight and does not overload the network or devices.

One thing that is not present by default with TCP/IP is encryption. This means that anybody who is able to intercept the communication can interfere with the data transmission and can therefore take control of the vehicle. This is a big security issue that will require the implementation of an extra layer, on top of the TCP/IP, encrypting the data to make the communication secure.

To transmit a video stream, each separate frame has to be sent one by one. If we use an average definition of 1280*720 pixels for the video, each frame contains more than 2.7 Mbytes of information, or 22 million bits that must be sent over the network. Because the vehicle will use 4G LTE or 5G cellular data to connect with the operator, the average upload speed will be around 10 Mbits/s.

1With this information, we can calculate the theoretical average time it will take to send one video frame: 22 / 10 = 2.2 seconds. This time is neither considering processing time by the server and client computers nor packet header size. As a video requires at least 25 frames per second in order to be fluid, the average time it should take for a frame to be sent is 1 / 25 = 0.04 seconds. Therefore, it is mandatory for the video to be processed in order to reduce the amount of data to be transmitted, as a single frame must not be greater than 400 kbits in order to accommodate the requirements.

A simple method that is quite easy to implement is reducing the video definition. Also, the colour levels can be lowered by for example, only using four bits per colour, and therefore reducing the size of each frame by half. But as you can see, this is still nowhere near the required compression. To perform this task effectively, the H.264 codec has been created in 2016, as an update of the H.263, and can achieve a 2000:1 compression rate.

By implementing the encoding part of this codec in the vehicle and the decoding part in the teleoperator's workstation, a live video feed from the vehicle should be made possible. However, as our project reached its end, we were not able to take the time to implement this solution. Instead, we reduced the quality of the main video by turning it into a black-and-white stream, only using one third of the necessary bandwidth. It should also be

pointed out that all our testing was done on a 500 Mbits/s network, allowing us to use less effective compression methods.

6.3.2 UDP connection

As was suggested during our midterm presentation by our jury, we tried to replace the TCP/IP communication with a UDP connection. This protocol is used in situations where low-latency communications are required such as in live streaming or voice-over-IP. This protocol is also loss-tolerating, which means that there is no guarantee the information will arrive to the receiver, and the sender does not get any confirmation from the receiver.

In most circumstances, the fact that there might be some data loss is not a big issue, as it will simply appear as a lag or a cut in the stream. At first, we were not able to transmit any data over UDP as the blocks we used were the wrong ones. But later on, we managed to transmit data using a different set of blocks parametrised in the right way which led to a successful video stream with little latency.

Yet it has to be pointed out that except from the delay, which was greatly reduced, we still faced the same problems as with the TCP/IP, about the maximum packet size, frame size and resolution. Consequently, we continued the research to find a different method to establish the communication in hope for a better-quality video stream.

6.3.3 ROS connection

Because of the trouble we had while working with UnrealEngine as mentioned in chapter 3, we were in direct contact with a MathWorks employee who was keen to help us out if we had any more issues. For that reason and because we were not able to find a good video streaming solution, we decided to approach him again.

After discussing our goals and issues during yet another online meeting, he advised us to use the Robot Operating System toolbox in Simulink, and kindly provided us with a simple example we could build upon. ROS is a framework designed to write software for automated systems and provides a wide collection of libraries aimed at simplifying software development. Even though we eventually managed to transmit our video stream, there were a lot of points that made us reject this option in the end.

Firstly, the data transmission rate was not exceedingly high, as a 200*400-pixel video was limited to about 10 frames per second making it very jerky. In addition, the setup of the ROS network was inconvenient, since a master device would need to be setup first, which could not be achieved directly via Simulink. Furthermore, the system itself was overly sensitive to connection trouble and would therefore often require to be restarted.

On top of that, we spent a lot of time learning how the framework functioned in order to fully implement it in our Simulink models. Despite our best efforts, we were never able to transmit all the necessary data for teleoperation of a vehicle as needed through the ROS framework. On account of all these points, we decided to stay with the TCP/IP connection, but the upcoming students continuing our project will need to implement a suitable codec for the livestream.

6.4 Implementation and proof of concept

After exploring all possibilities known to us, we finally managed to make a functioning teleoperation model with the operator's workstation and the car being simulated on different devices while communicating over ethernet. The commands were sent to the vehicle via TCP/IP while the video data was transmitted over UDP in order to limit the delays. Still, the video stream was limited to 65000 bytes which meant the best we could do was transmit a 200*300-pixel black-and-white video.

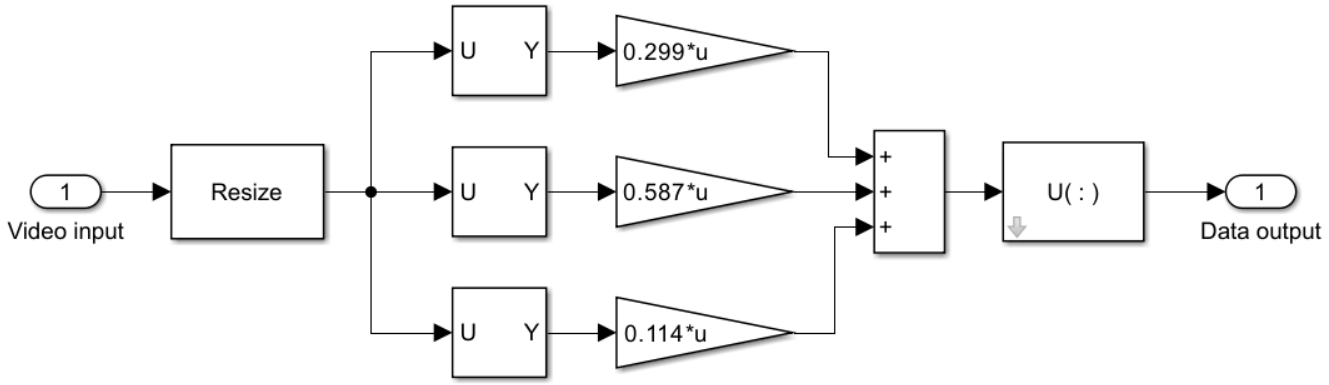
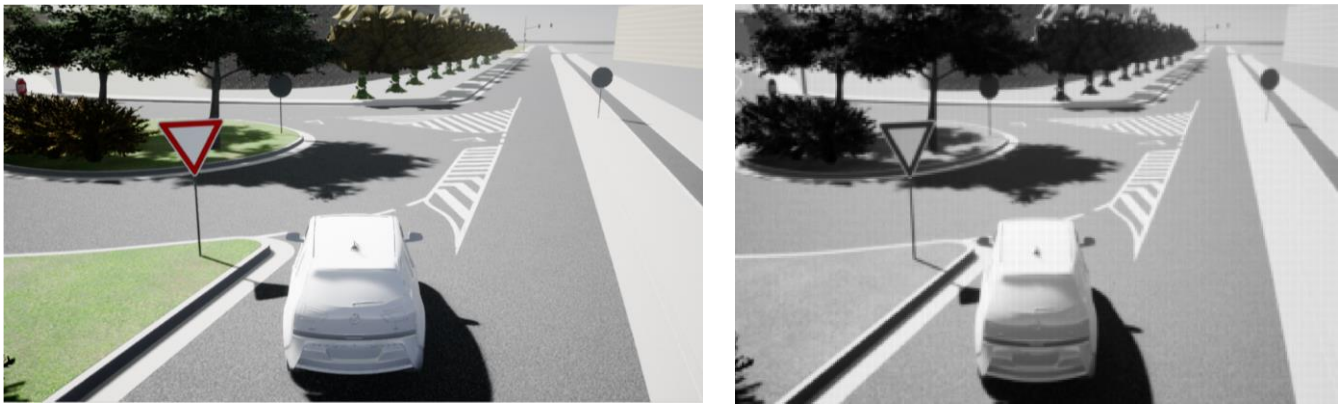


Figure 17: Frame processing and shaping for upcoming UDP transmission

In order to transmit these frames, they were first resized in order to shrink their resolution to the 200*300 maximum. Then, the distinct colours are separated in order to convert the image to black-and-white by applying different factors to the red, green and blue pixels. After adding the results, the resulting matrix is converted into a vector that can be transmitted via UDP.

On the receiving hand, the data vector is reshaped into a matrix in order to be decoded by the video viewer. The comparison between the original and received data can be seen in the figure below.



a) b)

Figure 18: Comparison between the original video frame a) and the received frame b)

The operator's side program is quite simple, as it only needs to format the commands received from the steering wheel and gas pedal and send them through a TCP/IP connexion, while also receiving the video stream through a UDP connexion. The simplified version can be seen hereunder, with the top section dedicated to the commands, and the bottom section dedicated to the video.

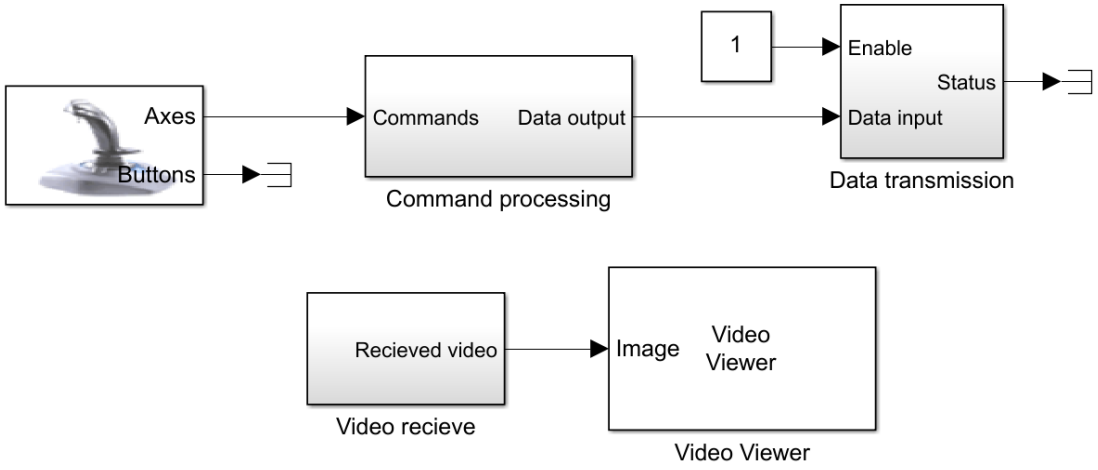


Figure 19: Simulink model of the operator's side of the teleoperation system

The vehicle side of the teleoperation system is much more elaborate because it incorporates safety switches and much more formatting in order to separate the data signals into the required commands. In the same way as before, the top section is dedicated to separating the commands while the bottom section is dedicated to the video compression and transmission. In the bottom right corner, there is the simulation blockset where we can parametrise the scene for the simulation. The 'algorithm status' and 'autonomous input' are normally connected to the autonomous algorithm driving the vehicle.

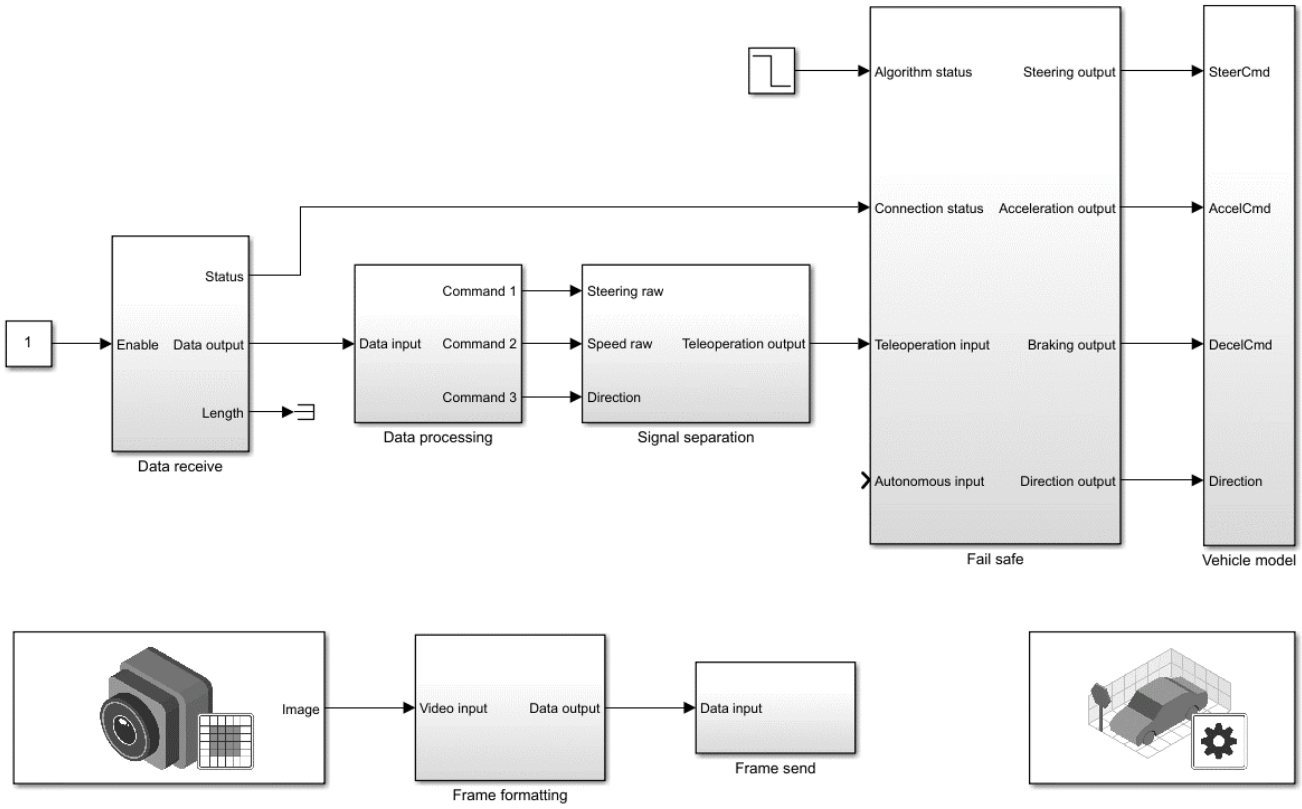


Figure 20: Simulink model of the vehicle's side of the teleoperation system

7 Simulation scenes and scenarios

With a much better scene designer software in hand, a much more promising outcome was to be seen. The initial work carried out in Driving Scenario Designer would therefore be left obsolete, and all our focus was directed towards RoadRunner and its incredible capabilities.

7.1 RoadRunner overview

Roadrunner's capabilities in terms of design ease and look of the simulation scenes being created is remarkably high, but it also is capable of running parallel visualisations with Unreal Engine and Simulink in order to make live changes in the scene while simulating the response of cars on the surface. In reference to Driving Scenario Designer, it is a real lifesaver.

Figure 21: Overview of a premade RoadRunner scene

The scenes created through RoadRunner are real 3D objects, which means that sensors like LiDAR or Radar sensors located on the simulation vehicle will pick-up the different props as they would in a real scenario. This in turn means that the response to real situations like pedestrians crossing, vehicles cutting you off, etc can be modelled closely, because different external actors like enumerated before can be parametrised to act as such.

Various weather conditions, traffic patterns, and road configurations are also some of the situations that can be included with the scenario in RoadRunner. This can then allow the driving algorithms to be adjusted without expensive and time-consuming testing before final verification in real life.

7.2 Different data for map design

To facilitate the workflow, RoadRunner offers the ability to import many diverse types of data to use for building the scene, or to use in conjunction with the different sensors of the cars during simulation. In order to obtain all this data, we used the Centro de Descargas website from the IGN that the MathWorks engineer we met provided us with. On the website, we were able to download many different data sets such as aerial images, topographical bases and LiDAR point clouds. All this data can be geolocalised in order for it to overlay perfectly when imported.

7.2.1 Aerial images

In our situation, the aerial image of the city and its surroundings will be the main data used to build its 3D model. This image is captured by flying over the area of interest and provides quite accurate dimensions and placement of the main parts of the road network and majority of the props. The remaining parts as well as the road markings can easily be located by using Google Street View.

Another great point of the aerial image is that it can be used to colour surfaces. For example, if a surface between the roads is a particular colour and texture, the image can be projected on top of it, making it look like the real surface. This can also be used to simply model the roofs of buildings in the scene.

The only downside is that even after the image has been corrected for earth's curvature before it was published, known as orthorectification, there are still some sections of buildings covering certain roads close by. This is due to the fact that the camera has a single focal point, and therefore is not able to see all sides of a building if it is located above it. This in turn meant that there were some sections of road which were not visible on the image, but by looking at the intersections and Google Street View, we could still place them correctly.

7.2.2 Topographic data

In RoadRunner, it is also possible to import topographical data. This data represents the contours of the most important physical features of the region of interest, by outlining rivers, lakes, buildings, and roads. In addition, it also provides the absolute elevation of the different landmarks it carries.

The fact that this data provides the exact outline of the different constructions and in most cases also their height, it is particularly useful to use as a reference because the aerial image dos not vehicle any of this information. Nevertheless, this data is more difficult to use directly to create buildings and roads, so we stuck with it just for verification purposes.

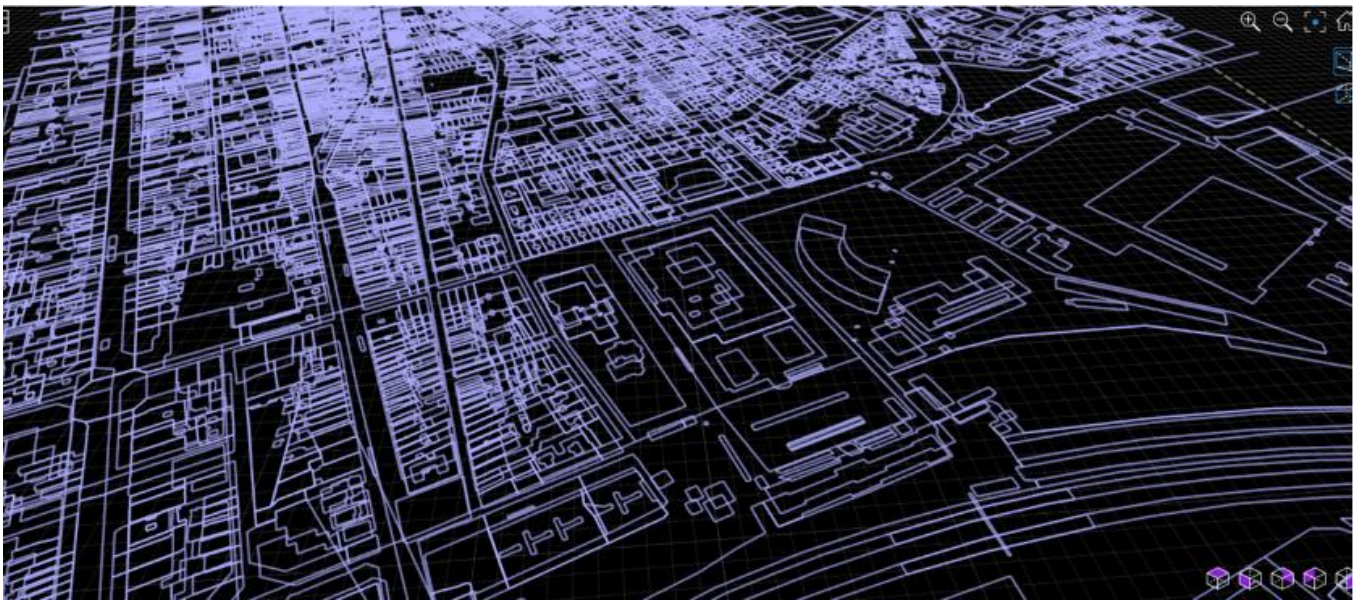


Figure 22: Example of a topographic base file imported in RoadRunner

7.2.3 LiDAR point cloud

Finally, the RoadRunner software also accepts a 3D point cloud, which was once again available on the IGN's website. This data has been acquired by flying LiDAR sensors over the country. Those sensors measure the distances by calculating the time it takes for the laser beam to travel from the sensor and back. This point cloud might be of acceptable quality in certain use-cases, but for our use it was too coarse.

Because the whole of Spain has been scanned using this technique, it had to be done fast while not creating too much data in order to keep the cost down. Because of this, the distance between two adjacent points was too big in order to create accurate measures when used in conjunction with the sensors on the simulated vehicles.

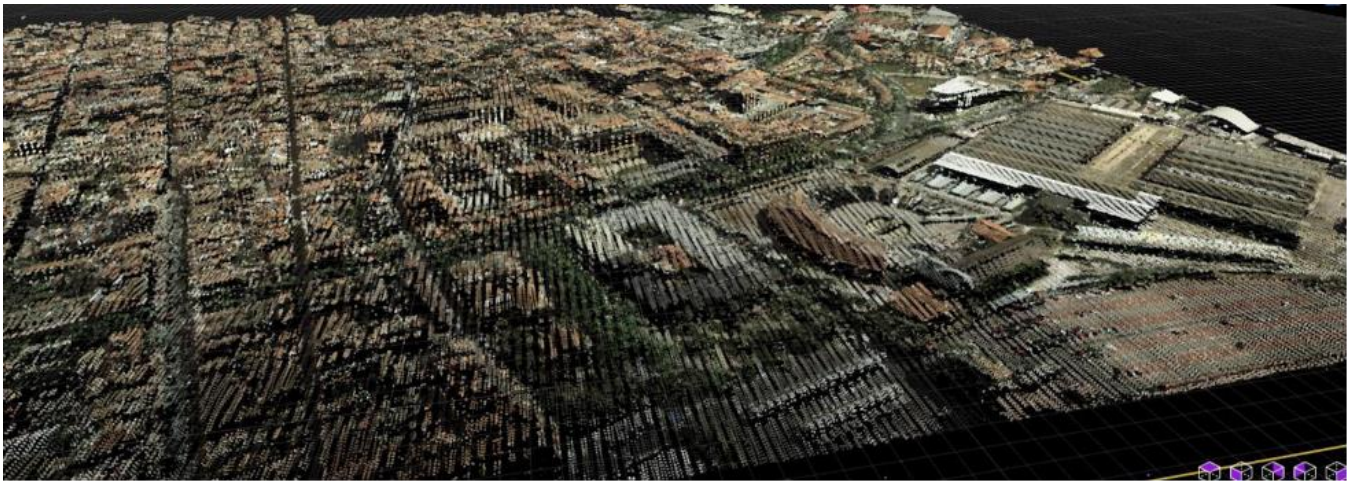


Figure 23: Result of a coloured LiDAR point cloud in RoadRunner

7.3 Recreating Vilanova i la Geltrú

During the first stay of our mentor in Vilanova i la Geltrú, he drove around the city with a camera in order to hopefully capture different scenarios which could end-up being problematic if encountered by an autonomous vehicle. He successfully managed to capture different situations, of which we could then model scenes in RoadRunner to replicate reality in the best possible way.

Now that we had the required data to make the design process faster, the objective was to create sections of the road network on which these different situations were captured, so a comparison could be made with the real location we had video from. RoadRunner offers the possibility to incorporate multiple scenarios in one scene. This means that we can select the scenario we want without having to change the map every time.

In the beginning, everything went well but a bit of time was needed in order to understand the workflow of the software, every action having its dedicated button. Still, the program was amazingly easy to learn, with the worst part of it being understanding the order in which you must execute the different steps to create different things. Most of the road building went without much trouble.

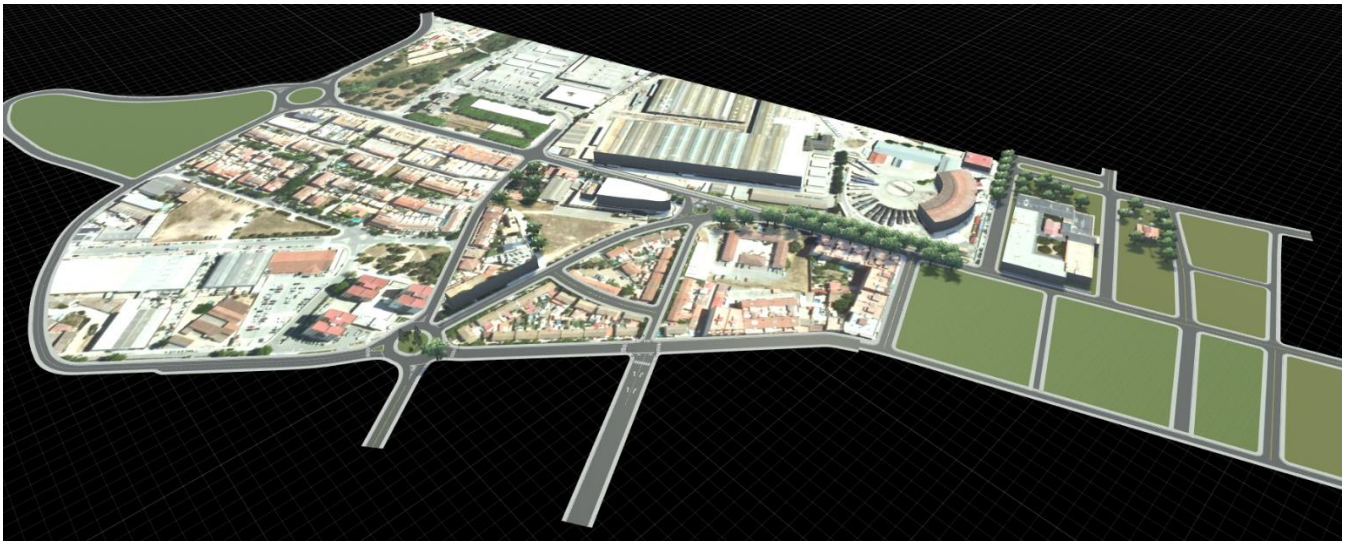


Figure 24: Screen capture of the section of the road network designed in RoadRunner

Despite that, two junctions that needed to be created in order to replicate the real scenarios captured on video were very difficult. In fact, after spending a lot of time perfecting these junctions, they still do not function properly while exporting the scene. While they visually look good, the path the car should take when its trajectory is generated by Simulink is non-existent, thus generating errors. In our case, because the car is driven either by the algorithm or by the teleoperator, the default paths are not used.

Figure 25: One of the problematic junctions during export due to its high complexity

To be able to model the encountered situations in the best feasible way, different custom props were created to speed up the map building process. Most of these props were simply put together from different objects already available while scaling them at the same time. But one of the objects was not available in the RoadRunner library, so we decided to create it ourselves with the help of Blender, as you can see below.

Figure 26: Example of a custom prop used in a fence (left) and the prop itself (right)

With this last prop, the map was complete enough to allow us to use it in our simulations even though it would require a lot more time in order to make it look better and add more sections to the map. However, before we can use it, other steps need to be performed in order to make it compatible with Simulink.

7.4 Integrating the map in Simulink

Finally, to be able to use our freshly created map for our simulations, two different routes are available, but both require the installation of Add-Ons “Automated Driving Toolbox Interface for Unreal Engine 4 Projects” and “Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects”. The difference on which one is chosen is mainly dependant on what you are trying to do, but most importantly, on the power of the computer running the simulation as you will see.

On the one hand, we have the possibility to create an executable file, this is, with an .exe extension. In this case, the RoadRunner map must be imported in a special Unreal Editor file provided by MathWorks, which can then be packaged as a standard executable file. This file is then opened in the Simulink block, and the simulation can be run with our custom map of Vilanova.

On the other hand, we can choose to co-simulate between Simulink and either RoadRunner or Unreal Engine. While using Simulink and Unreal Engine, easy modifications can be made to custom vehicles as the changes do not need to be saved before running another simulation. Nonetheless, the map still needs to be imported and saved with the help of the file provided by MathWorks, but without exporting it.

In the case the co-simulation is done with Simulink and RoadRunner directly, the beforehand created scenarios representing the real-life situations can be used. This is very convenient as it is much easier to create complex scenarios in RoadRunner directly than using Simulink to program them.

Apart from the different advantages enumerated previously, it should be kept in mind that Simulink is already a heavy program to run if the model in question contains 3D scenes. In the same way, Unreal Engine is an even more demanding software than Simulink.

Therefore, if a co-simulation is to be used, be sure to be equipped with a powerful computer, as the strain of running either of the two programs simultaneously is very intensive for the device. Because of this, it is often preferred to create the executable and simply run it independently in Simulink.

Overall, the combination of RoadRunner’s capabilities has greatly enhanced the outcome compared to the previous software we used. Integration in a Simulink model is more complex, but it has allowed us to fulfil the task better than we thought after having tried Driving Scenario Designer. This will allow for upcoming teams to thoroughly test their algorithms and protocols as well as latency before performing real-life tests.

8 Eco-Design

The environmental impact of modern technology must be held in high regard for the future generations. Although a large portion of this project has been primarily simulations, once implemented the following points must be taken into consideration. Hypothetically, these would reduce the environmental impact they have regarding energy consumption and emissions.

8.1 The use of electric vehicles (BMW i3)

These teleoperated autonomous driving systems can be integrated into current electric vehicles as well as developing newer ones, which have zero tailpipe emissions, reducing greenhouse gas emissions and air pollution. Electric vehicles are also far more energy efficient than traditional combustion engines especially when controlled autonomously for the large part.

Figure 27: Example of an electric car, here the BMW-i3 used by the HTW Dresden

8.2 Optimise routes taken

Autonomous driving systems can be programmed to improve the routes they take to reduce travel time, fuel consumption, and therefore emissions. By taking the most efficient route, autonomous vehicles can avoid congestion and unnecessary idling, reducing gas emissions for combustion cars thus improving air quality. They would also potentially be able to work in conjunction with other teleoperated vehicles to ensure journeys do not become considerably longer for the passenger compared to their combustion counterparts.

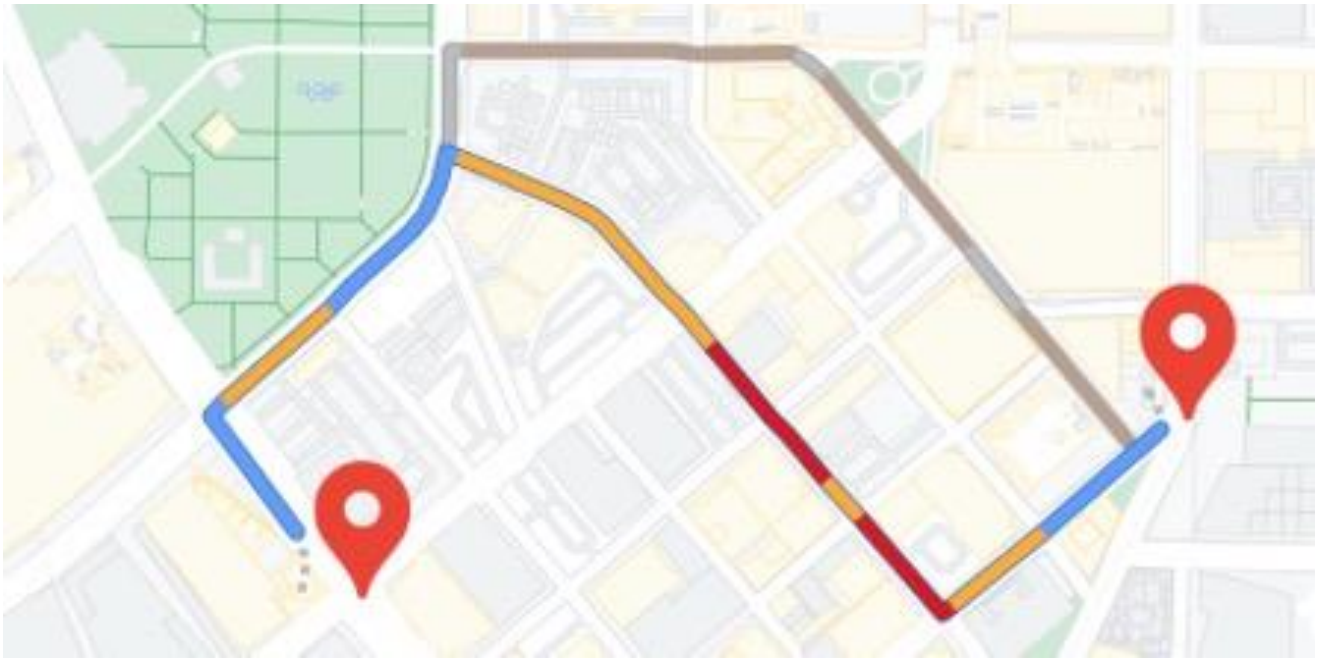


Figure 28: Example of an alternative route that would be longer but more efficien

8.3 Use of regenerative braking

Teleoperated electric vehicles can also be equipped with regenerative braking technology that captures the energy normally lost during braking and then uses it to recharge the battery pack which powers the car, in turn reducing energy consumption and emissions.

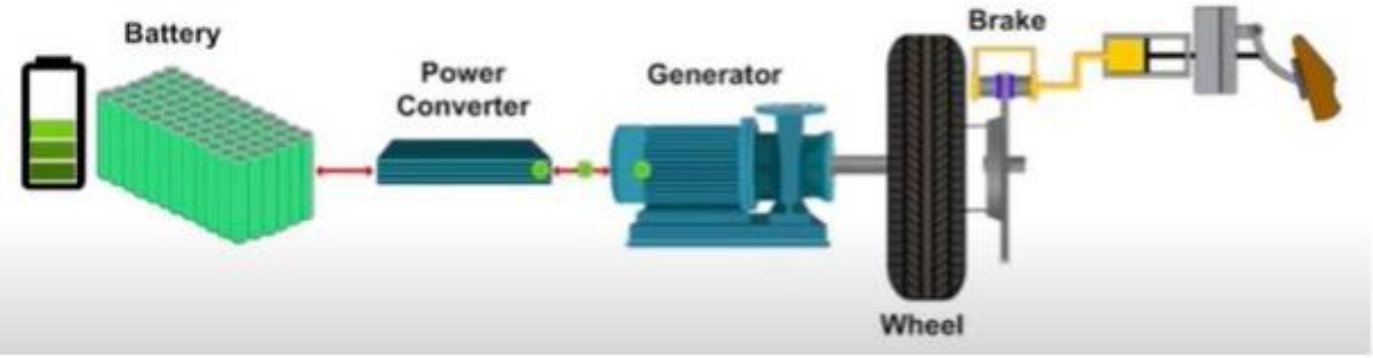


Figure 29: Diagram of a regenerative braking system used in electric vehicles

8.4 Optimised charging

Teleoperated vehicles could be programmed to charge predominately during off-peak hours, when electricity is cheaper and the grid is less stressed, reducing the overall environmental impact of the vehicle. The infrastructure would also need to be adapted so that the vehicles can be plugged and unplugged without human assistance, similar to the systems already currently available for Hyundai/Kia, or a completely cable free system in which the vehicle parks on top of the charger dock.

Figure 30: DC Fast-charging station for electric vehicles with a plug-in robot

8.5 Vehicle sharing

Autonomous driving could also be used to encourage vehicle sharing designs. By sharing vehicles, it means fewer cars are on the road, reducing any more unnecessary emissions with the incentive for free travel.

Figure 31: Example of a shared city-vehicle

Overall, the combination of autonomous teleoperated driving technology with electric vehicles, optimised routing, regenerative braking, smart charging, and vehicle sharing can significantly reduce the environmental impact of transportation.

9 Project management

To ensure that we were able to accomplish the tasks, we implemented different project management tools. All these tools enabled us to closely track the progress made during the project, but also allowed us to better understand the task itself by breaking it down in simpler and more feasible goals.

9.1 Project tasks breakdown and planning

To better understand the objectives of the project, the main tasks that needed to be accomplished to achieve the project's goal were defined, together with our mentors. These seven tasks were then refined in more detailed subtasks that are easier to manage and distribute between the team members. The work breakdown structure was as follows:

1) Introduction and familiarization.
   a) MATLAB introduction (OnRamp Course).
   b) Simulink introduction (OnRamp Course).
   c) Introduction to Simulink Toolboxes.
   d) Literature research.
   e) Model specification.
2) Design part of the Vilanova road network.
   a) Select minimum required network.
   b) Import network from OpenStreetMap.
   c) Clean-up imported network for simulation.
   d) Test and parametrize the map.
3) Define typical situations and use cases.
   a) Enumerate all possible use cases of teleoperated driving.
   b) Enumerate all possible situations leading to a teleoperator call.
   c) Select situations to be modelled off video data.
4) Modelling of preselected situations for simulation.
   a) Recreation of situations using predefined 3D scenes in Simulink.
   b) Implement surrounding sensors on simulated car.
   c) Test and parametrize simulation until simulated data matches measured data.
5) Include teleoperator connection.
   a) Detect algorithm failure and switch to teleoperator input.
   b) Implement teleoperator input in car software.
   c) Implement backup protocol for connection errors.
   d) Test and parametrize software.
6) Design teleoperator human-machine interface.
   a) Include transmission of vehicle parameter in software (speed, warning lights, etc.).
   b) Design typical dashboard to display values during simulation.
   c) Test and parametrize for similarity with real car.
7) Include TCP/IP connection.
   a) Enable bidirectional Localhost TCP/IP communication.
   b) Enable bidirectional TCP/IP communication between machines on different networks.
   c) Create digital twin of the car to test teleoperation.
   d) Test and parametrize software for teleoperation and connection errors.

e) Execute time delay measurements during teleoperation.

To keep track of the progress made, Trello was the main software we used. Our board includes four main sections. The first one gives a general overview of the To-dos that need to be done. This section includes tasks that we have not yet begun to work on but attached with a date and members that will work on it to keep the deadlines in mind.

The next two sections show the On-going and Done tasks. This really helped us to keep track of the different tasks and their deadline. Thanks to the included subtasks we could see the specific steps that needed to be taken to further accomplish the goal of finishing the main task.

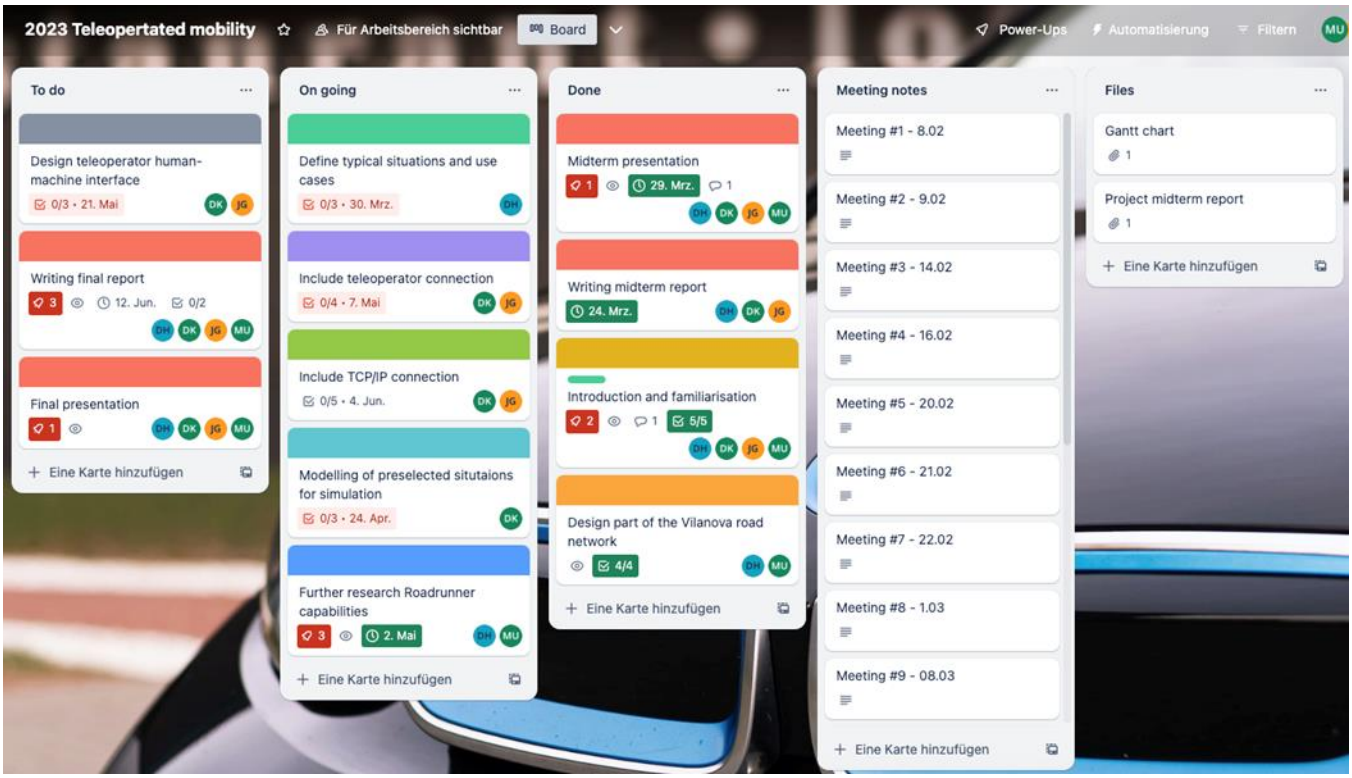


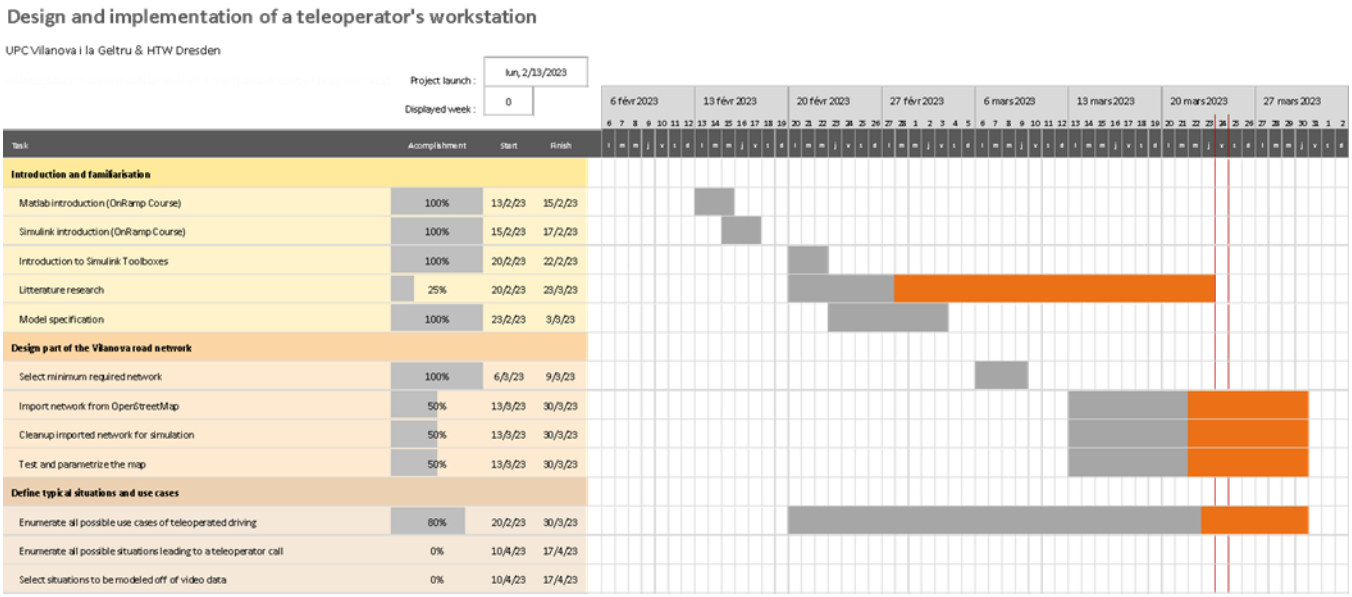
Figure 32: Capture of our Trello Board used to keep track of the progress of different tasks as well as sorting information relative to the project

Furthermore, we kept track of every meeting that took place in the section of the “Meeting notes”. This always included a list of contents of what happened during the meeting, to later have a better overview for everyone to look back and remind them of the things we talked about. After trying out different strategies to progress the project the best, we decided to have two fixed weekly meetings with all the team members and mentors if they had time to join us. These meetings were held in person on the campus of the UPC or the Neapolis building close by. We always created a Microsoft Teams call to let mentors or members of the team that could not join in person be part of the meeting.

Throughout the progress of the project, we always kept track of our deadline using the Gantt-Chart we created in the beginning of our project. This helped us to assess the current progress to maybe change priority of the upcoming tasks. Often, we had to push back the deadlines in the Gantt-Chart due to the amount of time and effort these tasks needed. While working on the project and trying to have it done on time, we realised that the

priority of tasks shifted, new information about specific tasks came up and some tasks just took longer to execute then expected in the beginning.

Table 2: First section of the Gantt chart used to plan our EPS project



9.2 Team management

We came to the conclusion early on while dividing up the tasks, that we would work in two teams consisting of two members. Because David and Martin both did not really have a lot of experience in the field of electrical engineering, they would work on the design part of the Vilanova road network, defining typical situations and use cases and teleoperator human-machine interface. As Joren and Duarte already have experience in this field of work, we decided that they would work on the 3D-visualisation of the car, including the TCP/IP connection and the teleoperator connection. We tried to find interesting and achievable tasks to work on for every member no matter their experience.

On top of that, we decided that Duarte would be the leader of the EPS project, as he has the most experience prior to this project and has worked on projects of this matter before. His task was to remind every member to keep everyone else up to date, keep the team members focused on their specific tasks and help other members out with his prior knowledge.

9.3 Future of the EPS project

At the end of the project, with all these tasks completed, it will be much easier for future EPS students to continue working on this subject and build upon it, since a solid base to develop a teleoperator's workstation will be available. This task selection and breakdown was completed thanks to the help of all the above-mentioned courses and research to better gauge their difficulty. Therefore, a non-exhaustive list of tasks for future EPS programs is displayed below, often due to their complexity or necessary time being too high.

1) Include 3D visualization for the simulations.

2) Include the compatibility for force-feedback hardware for the teleoperator.
3) Add static sensors in the simulation to allow for teleoperator control.
4) Implement a remote connection to the HTW Dresden.
5) Connect to the real car on the test track.
6) Design the HTW test track to use in simulation.
7) Improve the path following algorithm.

9.4 Project deliverables

In order to fulfil the project, we achieved the goal stated in the first paragraph of chapter 6, but also delivered the required outputs, that are as follows:

- A midterm report to be handed in on the 24/03/2023.
- A midterm defence to be held on the 31/03/2023.
- A final report to be handed in on the 12/06/2023.
- A final defence to be held on the 20/06/2023.
- A functional Teleoperator human-machine interface to be used for simulations and testing without physical equipment (steering wheel, pedals, etc.).
- A Simulink map of part of the Vilanova i la Geltrú road network to be used for simulation of the teleoperations.
- A selection of real situations and use cases that are implemented in the simulations for testing purposes.
- A Simulink/MATLAB code for the operator's desk including the input for a teleoperator connection.
- A Simulink/MATLAB code to validate and measure the latency of the TCP/IP connection between the Teleoperator's workstation and the car's digital twin.

Other goals have been put aside because we will probably lack time to complete them. These goals can be completed if there is spare time and will otherwise be completed during future projects with other students. They are stated as follows:

- Simulation use cases including 3D visualization scenes with Unreal Engine.
- A Simulink/MATLAB code for the operator's desk able to support force-feedback hardware (steering wheel, …).
- A Simulation of a predefined use case including surrounding on the car to test changeover from automated driving to teleoperated driving.
- Accomplishing a remote connection between the HTW Dresden and the UPC in Vilanova i la Geltrú.
- Connecting to the real test car at the HTW in Dresden.
- A 3D model of the HTW's test track to be used in the simulations.
- A 3D model of the BMW i3 also to be used in the simulations.
- Improvements on the path following algorithm used by the vehicle.

10 Conclusion

Our project aimed to develop a teleoperator system in order to implement it in an existing self-driving car. This is supposed to allow a teleoperator to take over control of the car when the driving algorithms encounter situations that they are unable to handle. Also, this solves many liability issues as the teleoperator who has started the vehicle is considered as the driver and, on that account, responsible for it, even when the vehicle drives autonomously.

To accomplish this, we heavily relied on the software and algorithms developed by the MechLab team at the HTW Dresden, the client of this project. Our work was centred around the BMW i3 which the MechLab team has converted into an autonomous vehicle with pedestrian and obstacle detection. Because of this existing car, we tried to make the simulations as realistic as possible by importing a 3D model of said vehicle.

During this project, we faced several challenges related to all parts of teleoperated driving, like the changeover from one driving mode to the other, data transmission, communication latency, driver immersion and more. To address the challenges related to driver immersion and latency, the different sensors already available on the car could be used in addition to the cameras to acquire extra data. This data should then be used to predict and overlay information on the video stream dedicated to the teleoperator, and by doing so create what is known as enhanced reality, which we were not able to spend any time on during our EPS stay.

In order to solve the issues that occurred during the switchover from automated to teleoperated driving, different solutions were available to us. The most evident solution is to ensure fast and reliable communication between the vehicle and the teleoperator, for example by using redundancy in the communication infrastructure, which we sadly were not able to test due to a lack of time. Furthermore, a specific algorithm to reduce the quality of the video stream based on the low importance of certain sections of the camera when driving would be a great solution. This should enable the transmission to work with the real-time bandwidth available for communication.

The next EPS team will therefore be able to focus on acquiring additional data from the vehicle sensors to predict its path and overlay this information on the video stream. Also, work needs to be carried out on implementing a very high compression algorithm in the video stream in order for the it to be more reliable and of better quality. In addition, some time could be spent on the improvement of the switchover from autonomous to teleoperated driving, with the creation of an output to trigger the change as well as a better fail-safe algorithm. Extra time could then be spent on creating more detailed and elaborate simulations and scenarios to better test the system's behaviour.

To conclude, we managed to achieve majority of the goals we started with, even after some heavy setbacks. Nevertheless, this left us with a time shortage leading to some unfinished goals. Even so, we were still able to create a solid base for other teams to build upon, with most of the theory sorted, now only lacking implementation. Our project also highlighted the interdisciplinary collaboration required between computer science, electrical engineering, and mechanical engineering to achieve ambitious goals in autonomous driving research. We hope that our work will support future students as well as future work and research in this exciting field full of innovation in the race towards teleoperated cars.

11 Bibliography

[1] Georg, Jean-Michael, et al. "Teleoperated Driving, a Key Technology for Automated Driving? Comparison of Actual Test Drives with a Head Mounted Display and Conventional Monitors." 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2018, https://doi.org/10.1109/itsc.2018.8569408.

[2] Hosseini, Seyed Amin. "Conception of Advanced Driver Assistance Systems for Precise and Safe Control of Teleoperated Road Vehicles in Urban Environments." Universitätsbibliothek Technische Universität München, 2018, https://mediatum.ub.tum.de/node?id=1415643.

[3] 5GAA Automotive Association. "Tele-Operated Driving (ToD) : Use Cases and Technical Requirements." *5GAA*, https://5gaa.org/tele-operated-driving-tod-use-cases-and-technical-requirements/.

[4] Hofbauer, Markus, et al. "Traffic-Aware Multi-View Video Stream Adaptation for Teleoperated Driving." 2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring), June 2022, https://doi.org/10.1109/vtc2022-spring54318.2022.9860513.

[5] Organismo Autónomo Centro Nacional de Información Geográfica. "Centro de Descargas Del Cnig (IGN)." Centro de Descargas Del CNIG, centrodedescargas.cnig.es/CentroDescargas/index.jsp

[6] Prakash, Jai, et al. "Vehicle Teleoperation: Human in the Loop Performance Comparison of Smith Predictor with Novel Successive Reference-Pose Tracking Approach." Sensors, 2022, https://doi.org/10.3390/s22239119.

[7] Cao, Songxiao, et al. "Design and Experiments of Autonomous Path Tracking Based on Dead Reckoning." MDPI, Multidisciplinary Digital Publishing Institute, 27 Dec. 2022, https://www.mdpi.com/2076-3417/13/1/317.

[8] "MechLab." MechLab HTW Dresden, https://www.htw-mechlab.de/.

12 Appendices

12.1 Extract of the German Road Traffic Act (translated)

(German) Road Traffic Act (StVG)

"Road Traffic Act in the version published on March 5, 2003 (BGBl. I p. 310, 919), which was last amended by Article 16 of the law of March 2, 2023 (BGBl. 2023 I No. 56)"

…

§ 1d motor vehicles with autonomous driving function in defined operating areas

(1) A motor vehicle with autonomous driving function within the meaning of this Act is a motor vehicle that

1. can perform the driving task independently in a defined operating area without a person driving the vehicle and

2. has technical equipment in accordance with Article 1e Paragraph 2.

(2) A defined operating area within the meaning of this Act refers to the locally and spatially determined public street space in which a motor vehicle with autonomous driving function may be operated if the requirements pursuant to Section 1e Paragraph 1 are met.

(3) The technical supervisor of a motor vehicle with autonomous driving function within the meaning of this Act is the natural person who can deactivate this motor vehicle during operation in accordance with Section 1e Paragraph 2 Number 8 and who can authorize driving manoeuvres for this motor vehicle in accordance with Section 1e Paragraph 2 Number 4 and Paragraph 3.

(4) A minimum-risk condition within the meaning of this Act is a condition in which the motor vehicle is with autonomous driving function at their own initiative or at the instigation of the Technical Supervisor brought to a standstill in the safest possible place and the hazard warning lights activated in order to ensure the greatest possible safety for the vehicle occupants, other road users and third parties while taking due account of the traffic situation.

…

§ 1e operation of motor vehicles with autonomous driving function; objection and challenge

(1) …

(2) Motor vehicles with autonomous driving function must have technical equipment that is able to:

…

3. to independently bring the motor vehicle into a risk-minimum state if the continuation of the journey would only be possible by violating road traffic law,

4. in the case of number 3 of the technical supervision independently

a) to suggest possible driving manoeuvres to continue the journey as well as

b) to provide data to assess the situation so that Technical Supervision can decide whether to approve the proposed driving manoeuvre,

5. to check a driving manoeuvre specified by the technical supervisor and not to carry it out, but to independently bring the motor vehicle into a risk-minimum state if the driving manoeuvre would endanger persons participating in traffic or bystanders,

6. immediately report any impairment of their functionality to the Technical Supervisor,

7. to recognize their system limits and to independently put the motor vehicle in a risk-minimum state when a system limit is reached, when a technical fault occurs that impairs the exercise of the autonomous driving function, or when the limits of the specified operating range are reached,

8. to be deactivated at any time by technical supervision or by vehicle occupants and, in the event of deactivation, to independently bring the motor vehicle into the minimum-risk state,

9. to indicate to the technical supervisor the need to activate an alternative driving manoeuvre, deactivation with sufficient time reserve as well as signals on their own functional status optically, acoustically, or otherwise perceptibly and

10. Ensuring sufficiently stable radio connections that are protected against unauthorized interference, in particular to the Technical Supervisor, and automatically putting the motor vehicle in a minimum-risk state if this radio connection breaks down or is accessed without permission.

…

12.2 Gantt chart

Below, you will find the complete Gannt chart we used to breakdown our project goals. This chart is bound to evolve as the project goes on.

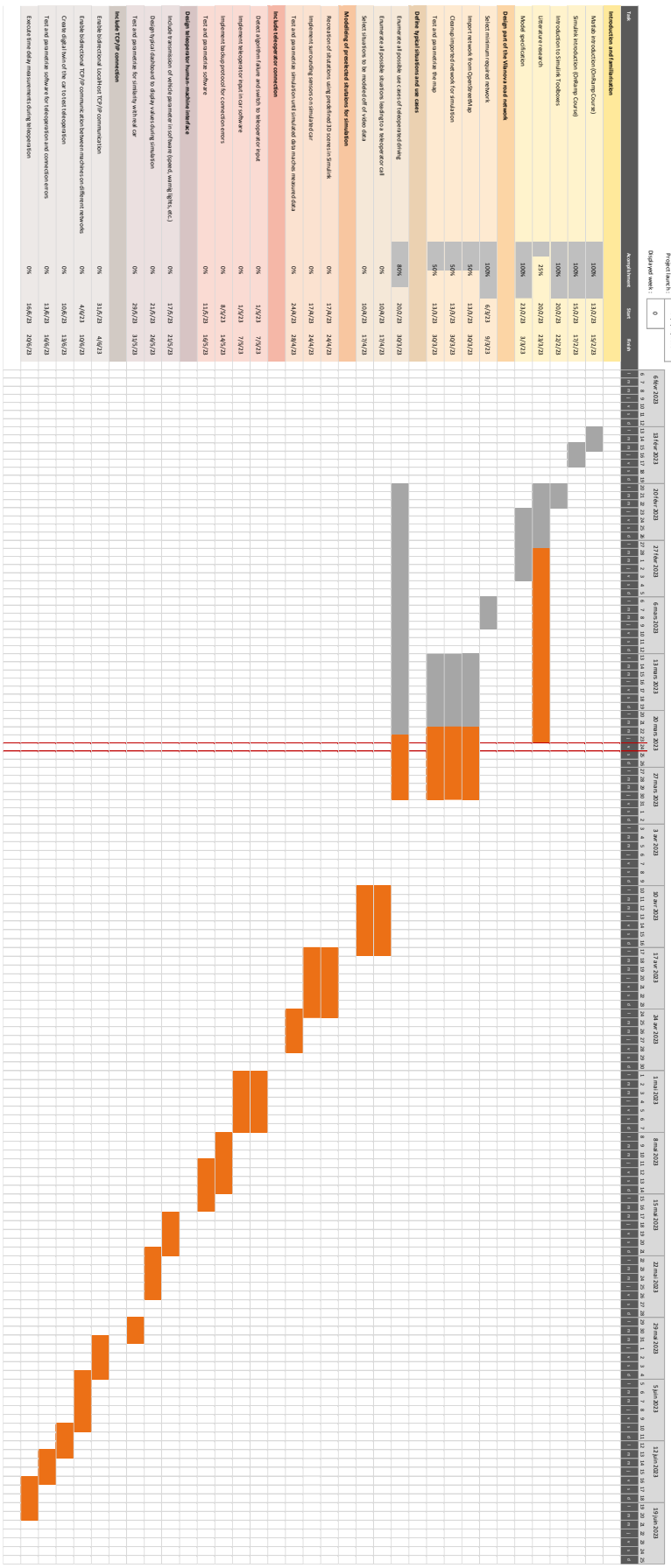


Table 3: Complete Gantt chart of our EPS project