Design Project

Motorbike Optimisation Simulator and Live Driver Coaching

Group 15:

Adam Bosch - s3161803

Bartosz Sanowski - s3152340

Maja Korzeniowska-Jęczeń - s3141837

Huy Ho - s3188981

Chris Sirbu - s2700433

Client:

Electric Superbike Twente

Supervisor:

Dr. Faizan Ahmed

April 17, 2026

UNIVERSITY OF TWENTE.

Abstract

The project is a high-performance simulation designed to aid the integration of complex mathematical modelling and real-world racing strategy. Its main purpose is to help engineers build, test, and optimise vehicle performance using a virtual environment that mirrors reality. It provides a specification laboratory where the engineer can write their own physics formulas. The project includes a live web dashboard on which you can watch the simulation unfold on a specified track map, adjust vehicle settings, and, in the end, download the generated optimal lap data. Additionally, the project features a Live Driving Coach to guide the rider throughout their lap.

Table of Content

Chapter 1. Introduction

Electric Superbike Twente is a student racing team from the University of Twente. Their main focus is on reshaping electric motor racing. Each year, a new team of students designs and builds a fully electric motorcycle from scratch. With this motorcycle, the team competes in the MotoStudent competition, an international event held every two years at the MotorLand Aragón circuit in Spain. The competition is a challenge not only for the engineering team but also for the bike rider himself, who has to make split-second decisions in the high-performance racing environment. Under enormous pressure, human error is bound to occur, reducing the quality of their decisions and overall performance. To counter that, the Electric Superbike Twente team aims to add an onboard Live Driving Coach, which would provide real-time feedback to the rider. The system is intended to suggest optimal actions based on the motorcycle's current state and a calculated optimal line to support the rider in their decision-making process.

Over the course of ten weeks, our group took on the challenge of developing a high-performance motorbike optimisation simulator based on Model Predictive Control (MPC). The MPC aids in predicting future behaviour by solving an optimisation problem. The simulation uses different control strategies to determine an optimal racing line for the track provided by the user. The system developed by our group consists of two main components. The first component is a dashboard web-app that allows users to define the vehicle-focused specifications, control inputs, and output variables for a specific vehicle, define mathematical formulas, and run simulations to find an optimal lap. The second component is the onboard Live Driving Coach, which provides live feedback to the rider during the race using the generated optimal lap.

This report details the implementation and technical specifications of the simulator and the Live Driving Coach. It outlines future development plans and provides an evaluation of the final product and individual contributions. Chapter 2 discusses the project proposal and our chosen direction. Chapter 3 outlines the system requirements defined by our group. Chapter 4 describes the project structure and team contributions. Chapter 5 explains the technical architecture and key design decisions. Chapter 6 introduces the Model Predictive Control approach used in the simulator. Chapter 7 focuses on the implementation of the Live Driving Coach. Chapter 8 shows our testing methodology and results. Chapter 9 explains in detail how to

use the system. In Chapter 10, we talk about the potential future improvements, and finally, Chapter 11 evaluates the project and its outcomes.

Chapter 2. Project Proposal

2.1. Client's proposal

During the initial talks with the Electric Superbike Twente, the team proposed the development of a real-time Live Driving Coach with the aim of improving rider performance during races. The system would use the onboard sensors, historical racing data, and track-specific analytics to optimise the race strategies. By using various data such as lean angle, yaw, acceleration, and power usage in combination with historical lap logs and track layout, the team wanted to train a machine learning model capable of predicting the optimal racing line and speed. To refine this further, a physics-based simulation environment was suggested for reinforcement learning, allowing the model to run in a virtual space. The ultimate goal was to deploy this intelligent system onto the bike's NXP S32 embedded system using TinyML, providing the rider with live feedback on the bike dashboard.

However, in the first sprint of the project, our group noticed that the dataset provided by the client is way too small and insufficient in quality to properly train a high-quality machine learning model. As a result, in collaboration with the client, the project direction was adjusted towards a physics-based simulation approach.

2.2. Our Direction

After the initial discussions with the client and reviewing the data provided by them, we settled on a small change of direction. Since the dataset was way too small to properly train a driving coach on it, we decided to focus on implementing a motorbike track simulation to model an optimised race line for the race. Instead of relying primarily on machine learning, we shifted our strategy towards a physics-based simulation approach. Secondly, we aimed to use the optimised path file produced by the simulation to further implement the requested Live Driving Coach that the client could deploy on the bike and connect to their onboard dashboard. By combining optimisation system simulation with live feedback, the two components enabled us to properly implement the client's requirements. This not only addresses the limitations of the initial idea but also makes sure that the final system remains practical, deployable, and aligned with the client's goals.

Chapter 3. Requirement Specification

3.1. Functional Requirement

Simulation/Web-app

- ☑ As a user, I want to access a simulation web application so that I can configure and run motorcycle simulations.
- ☑ As a user, I want to define inputs, variables, constraints, and their relationships, so that I can model different riding scenarios accurately.
- ☑ As a user, I want to create my own cost function so that I can optimize the simulation according to my personal performance criteria.
- ☑ As a user, I want to specify one or more track files, so that I can simulate riding on different tracks.
- ☑ As a user, I want to customize multiple vehicles and their specifications, so that I can compare performance across different setups.
- ☑ As a user, I want to configure simulation parameters so that I can control how the simulation behaves.
- ☑ As a user, I want the simulator to use state variables (e.g., position, speed, yaw) and control inputs (e.g., throttle, brake, steering), so that the simulation reflects realistic vehicle dynamics.
- ☑ As a user, I want the simulator to compute an optimal path that minimizes lap time and satisfies my requirements, so that I can improve performance.
- ☑ As a user, I want the simulator to use MPC-based trajectory planning, so that the generated path is mathematically optimal and dynamically feasible.
- ☑ As a user, I want the simulator to generate an optimal path file so that I can reuse it later.
- ☑ As a user, I want to download the optimal path file so that I can deploy it to my onboard system.

Live Driving Coach

- ☐ As a rider, I want the system to run entirely on the bike, so that I can receive feedback without relying on external connectivity.

- ☑ As a rider, I want to upload an optimal path file to the onboard system so that it can be used as a reference during riding.
- ☑ As a rider, I want the system to calculate my deviation from the optimal path at each point, so that I understand how closely I follow it.
- ☑ As a rider, I want to receive live feedback while riding, so that I can adjust my behavior in real time.
- ☑ As a rider, I want the system to integrate sensor data and compare it to the optimal path, so that feedback is accurate and context-aware.
- ☑ As a rider, I want the system to process sensor data with low latency, so that feedback is timely and actionable.

System & Deployment

- ☑ As a developer, I want the system to be deployable on a GPU-enabled device, so that computationally intensive simulations run efficiently.
- ☑ As a user, I want the web application to match the client's website design, so that the experience is visually consistent.
- ☑ As a tester, I want the system to support testing with historical lap data, so that performance can be validated without live runs.

As shown above, the majority of functional requirements have been fulfilled within the project. The only requirement above which has not been fulfilled concerns running the project on the bike. This was unfeasible due to the bike not being functional during the time period of the project, which prevented us from testing our project.

3.2. Nonfunctional Requirement

- ☐ As a system integrator, I want the onboard system to run on the NXP S32K344 platform, so that it is compatible with the client's hardware environment.
- ☑ As a developer, I want the model size to fit within available memory, so that the system can run reliably without memory overflow.
- ☑ As a rider, I want the system to provide feedback with a latency of 0.1 seconds or less, so that feedback is effectively real-time and actionable.

- ☑ As a user, I want the web application to match the client's website design, so that the user experience is consistent and professional.
- ☑ As a user, I want the web application to work on modern browsers (Chrome, Firefox, Edge, Safari), so that I can access it regardless of my preferred platform.

As shown above, there is one non-functional requirement which was unfulfilled. This concerns operating the driving coach on the main onboard computer (NXP S32K344), which is due to our group not having access to this microcontroller for testing. To combat this, our group researched that the microcontroller can run files in the C language and thus made a script in C to perform as our driving coach, and tested it locally.

3.3. Client's requirement

- ☑ As a rider, I want the real-time coach to provide a continuous stream of feedback in real time.
- ☑ As a user, I want the system to allow customisation of vehicle parameters to fit the potential modifications of the bike in the future.

The client requirements above describe some extra requirements that they added after we created our own requirements for the project. These requirements were fulfilled as there were not many additional requirements for the project.

Chapter 4. Project Structure and Group Contribution

4.1 Project Structure

Within the project, we followed the Scrum methodology to work in effective sprints. This ‘agile’ framework uses sprints and scrum ‘masters’ to focus and lead discussion in meetings. While there was not a daily standup every day, there was often communication within a collective group chat discussing what each member was working on. During the project, we decided to have only one scrum master, instead of alternating, which was due to the chosen scrum master being very effective at managing conversations during standups. Lastly, we split the project into 5 sprints of two weeks, where each sprint had clear goals stated at the beginning. This improved motivation and coordination between team members.

To log tasks and team member contributions, we used a Trello board, which displayed all of our tasks. These tasks were able to fit into five categories: backlog, to do, doing, review, and done. Each member was able to assign themselves to a task and use the Trello board as they saw fit. This methodology was extremely useful for keeping accountability between members of the amount of work they provided.

Outline of each sprint:

Sprint	Content
1	Choosing a topic, Looking for a supervisor, Contacting clients, Defining clients’ needs
2	Brainstorming ideas, Design, Identifying directions, Frontend
3	Frontend, Backend, MPC, Live driving coach
4	Connecting the web-app to the MPC, Finalizing live driving coach
5	Testing, Writing a report, Making a poster, Presenting

4.2 Group Contribution

Throughout the project, work was distributed among members based on their interests and expertise. This was done as fairly as possible and adjusted to maintain fairness, depending on the assignment's difficulty and complexity.

Team Members	Contribution
Bartosz Sanowski	Creation of MPC + Extraction of Tracks Real World + Database
Adam Bosch	Front-End + Front-End connection to Back-End
Chris Sirbu	Live Driving Coach Implementation
Huy Ho	Dynamic Cost Function for MPC
Maja Korzeniowska-Jęczeń	Design + Front-End + Communication and Coordination (Scrum master)

Chapter 5. Technical Architecture and Design Choices

The overall system architecture is illustrated in Appendix D.7 (Block Diagram), showing the interaction between the web-app, simulation engine, database, and live driving coach.

5.1. Frontend

Within our project, we describe the front-end as how the user interacts with our web-app. In our case, this includes four HTML files for the four different web pages. Combined with these files, we have a CSS file to add styling. The front-end has a distinct color palette and font, meant to imitate the colors and text used by the client on their website and Logo design.

While creating the front-end, there were discussions of using frameworks (like React) to make the process easier. However, due to the front-end not being very complex and the simple integration with Flask, it was decided to disregard the usage of these frameworks and instead create the front-end in a more simplistic manner. If the project were to become bigger, it would be wise to migrate to a framework for better readability and component reusability.

Based on the requirements for the web-app, we decided to only have four pages for the user to interact with, these being the Vehicles page (see appendix A.1), the Specifications page (appendix A.2), Simulation page (appendix A.3), and lastly the Runs page (appendix A.4). The Vehicles page allows the user to select a vehicle, also known as a profile, which will have different settings and constraints for the simulation. This was added for the client to see how different setups of the motorcycle can affect lap time and movement. The user is able to create and delete vehicles at their own discretion. Secondly, the Specifications page allows the user to add their inputs, variables, and tracks for the simulation. The variables need formulas to be added; these are processed via a LaTeX formula editor. This was a design choice since we needed a way for the user to add constraints for the simulation while making it easy to parse for the simulation. Lastly, there is the Simulation page, where the user selects a track and their parameters for the simulation and then is able to run the simulation. The simulation is displayed on a real map where the track barriers are overlaid based on the selected track file. Once the simulation has run 5 iterations, the path will be sent to the webpage, and then the user can see the path at the end speed it takes. Lastly, the user can see statistics of the simulation on the right side of the screen, concerning velocity, acceleration, and their respective averages and peaks. To

implement the map, we used Leaflet, an open-source JavaScript library for displaying maps on webpages. We used the selected track file to introduce the track boundaries (since the track file has GPS coordinates of the boundaries). Lastly, we can push the data of the motorcycle from the simulation onto the maps and track its speed using colors (from red to blue). For the statistics of the simulation, we used Chart.js to render our velocity and acceleration vs time graphs. Other calculations (apart from acceleration) are done within the frontend in a JavaScript function. The last page (Runs) allows the user to download the previous simulations and delete them from the database.

5.2. Backend

In order to run and interact with the simulation, there needs to be a connection layer between the front-end, the database, and the simulation. This is done through what we call the back-end. The backend is a Flask app, which is a lightweight web framework written in Python, that allows clients to run the web app with ease. We used Flask since it is very easy to set up, and our web-app was not complicated enough for a more sophisticated and difficult framework. Furthermore, our group was already familiar with using Flask and therefore deemed it to be a reasonable choice.

The majority of the backend is written in a file named ‘app.py’, which runs the Flask app. The functions in this file are actions that the front-end can call on specific routes (e.g., ‘@app.get(“specification/<int:vehicle_id>”). The majority of these routes are connected to the database for insertions, deletions, or updates. To decrease the size of the app file, we have a database connector file (‘db_operator.py’) which handles the real connection with the database through SQL commands (see Appendix D.4: Sequence Diagram, for more information). This file also deals with providing the correct credentials for the database.

While there is some security measure to ensure that the database cannot be tampered with fully (through enforcing everyone to make their own .env file for the database, which is not provided on GitLab), there is a distinct lack of security for the web-app itself. This is largely due to there not being a real need. From our discussions with the client, they described using the system locally on their own servers, and only accessible through a local connection. Therefore,

there is not much security implemented within the application, since an attacker would have to be physically close enough to use the application.

5.3. Database

The database plays a key role in the system. It allows users to define their vehicle dynamics according to the specification of their vehicle, as well as the goal that they are aiming to achieve in the simulation. The vehicular dynamics consist of hundreds of formulas; defining them before every run would be tedious. Before running the simulation, all of the necessary data is loaded from the database, using a special function which parses JSON format.

As can be seen on the database class diagram (see Appendix D.6: Class Diagram), the system consists of two components (Track_file and Vehicle), which together generate a simulation run. The track file is a KML file stored as a data array, defining the boundaries of the track in their real position. The Vehicle has two subclasses, namely the Control Input, which defines the operations that can be performed by the driver (such as throttle, braking, steering), and Variable, which defines the vehicle dynamics based on the Inputs. In order to calculate a Variable, a Formula must be used, which defines a mathematical formula used to calculate the next step value for a Variable during a simulation. Since the formula may use other Variables and Control Inputs, a Formula_Variable class is defined, which references the formula, the symbol in the formula, and either a Variable (id) or a Control Input (id).

For each of the classes, we used serial IDs as the primary key, with the exception of Formula_Variable, where no two of the same symbols should be used within one formula, hence the primary key is (formula_id, symbol). As mentioned previously, a parser function is used for the retrieval of necessary information from the database for a simulation run. That function returns a JSON with Control Inputs, Variables, Formulas, and Formula_Variables.

Chapter 6. Model Predictive Control

As mentioned previously, the simulation is designed based on Model Predictive Control (MPC). MPC is an advanced, model-based strategy that calculates optimal control inputs to forecast the system's future behaviors over a finite horizon (our system allows users to freely define the horizon) while explicitly handling constraints.

At each moment, the MPC looks ahead and generates a large number of sequences of actions for the next and predefined timesteps, returning an optimal control sequence that has the highest score (Chapter 6.1). However, the system only applies the first step of this result and then re-resolves at the next timestep with the shifted horizon.

An illustration of the MPC result can be found in Appendix B. The internal workflow of the MPC process is visualised in Appendix D.1 (State Machine Diagram).

6.1. Cost Function

The cost function is the heart of the MPC. It is how MPC decides which plan is the best. Every candidate sequence is assigned a score. The principle is that the higher the score, the better the plan. The score is built on two simple principles. The system will be rewarded when it makes choices in accordance with the wanted behaviour and will be penalized when unwanted behaviour is committed. Only the highest-scoring candidate is kept.

The total score always includes two built-in components that are present in every run. The first one is a progress reward based on the distance travelled along the track. The next one is a speed reward based on the accumulated velocity. These aren't configured by users.

In our implementation, the cost function incorporates user-defined rewards and penalties (or cost in the context of the project). The total is computed by the sum of all active rewards minus all active penalties, with each term scaled by a configurable multiplier. The cost function is defined based on the users' ultimate goal (details of this are explained in Chapter 6.3)

6.2. Constraints

The system must respect the physical limit of the vehicle and the boundaries of the race track. The constraints are defined by users in the UI. According to the client's request, the implementation only needs to take hard constraints into account. In the context of MPC, hard constraints define the physical limits which can't be violated under any circumstances.

The implementation enforces hard constraints on both control inputs and user-defined variables. Firstly, all control inputs are strictly numerically bounded by their minimum and maximum value, meaning they can’t physically exceed those limits. Secondly, after each user-defined variable formula is evaluated, its result is checked against the configured bound. When they are violated, the MPC sequence is terminated with an extremely large negative score. This aims to ensure that the vehicle will maintain a physically stable state, resulting in desired behaviors and avoiding unwanted behaviours like slips, jerks, or sharp steering.

6.3. User-defined Variable Rewards and Costs

As mentioned previously, in the total score, the system allows users to define target goals as reward generators and unwanted behaviour as cost generators. When the vehicle does something desirable, its score goes up, and vice versa when the vehicle does something undesirable, the score is deducted. Suppose a target goal would be to have high velocity, so the user could set it as a reward generator and apply a multiplier to the score to define how important that reward is. On the other hand, when the user aims for battery efficiency, they can define a cost to battery usage, which will take the previous and current battery levels to calculate the change and penalize the system for the voltage drop.

Unlike control inputs, the user-defined variables are not assigned fixed values. Instead, they are computed by a specific formula inserted with control inputs or other variables by using LaTeX. The control inputs are parsed and converted into an executable expression in the backend. This allows the formula to reference any necessary control inputs or variables.

Chapter 7. Live Driving Coach

The Live Driving Coach is an embedded system that gives continuous feedback to the driver to improve track times or energy conservation. The main motivation behind this design is to provide concise and easy-to-understand pointers such that the rider can change his driving behavior, either by correcting mistakes he unconsciously makes or by encouraging him to push harder and extract additional performance. The runtime interaction flow of the coach is shown in Appendix D.5 (Sequence Diagram).

7.1. Design choices

As with other parts of this project, the Live Driving Coach went through multiple ideas and iterations before settling on a final suitable approach. Initially, the main idea was to provide the driver with feedback on the optimal speed and the current position on the track relative to its centerline. By telling the driver that he should have been more on the inside or outside of a certain apex or corner entry, more of the drivable track surface could be used, which in turn results in a higher cornering speed. Subsequent meetings with the client showed that the two most important aspects to improve are finding the optimal braking point before a corner and having an appropriate bike lean angle during a corner.

Typically, it is under braking where riders make the most mistakes and lose the greatest amount of time. Being too late or too early influences the speed they can achieve when entering and exiting a corner. The bike's lean angle is also paramount for increasing speed. Its main role is to balance the two biggest forces applied to the bike during a turn, centrifugal and gravitational, and keep them aligned through the tires, resulting in a tighter turning radius and higher speeds. Besides the optimal lean angle for the specific position, the max lean angle for the upcoming corner is also shown in advance, as per the specifications of the client.

Due to the onboard nature of this system, a hard requirement identified was that the live coach was to be used on an NXP S32K344 microcontroller, designed for Automotive General Purpose. For this reason, a low-level programming language had to be used, for which C was chosen, to coincide with the language planned to be used by the client.

7.2. Functionality

For the system to function, it makes use of two files. The first one, consisting of the live coach implementation, is the core file. It contains the main functioning loop and all of the logic necessary to prepare the feedback to be shown to the rider. The second file contains the optimal trajectory calculated by the MPC simulation. This file is to be exported from the web-app dashboard after a simulation is completed. Every time a simulation with different parameters is finished, and the user wants to use it for the live coach, the file must be downloaded again. The live coach implementation, including the optimal trajectory, is then flashed onto the microcontroller at the beginning of every driving session.

The optimal trajectory consists of multiple trajectory points laid on the optimal racing line of the circuit. Each point contains the optimal speed and lean angle for that position. When initiated, the system looks at all of the coordinates from the optimal trajectory and compares the live bike location with them. The closest point is located and set as the current bike location with the speed and lean angle compared with live sensor readings, and gives the difference between them, along with a brake warning when the actual speed is higher than what is expected, and the lean angle for the upcoming corner. After the corner is passed, the lean angle changes to show the value for the next one. A sliding window then starts, and the system looks again for the closest coordinate in the proximity of the current one. This is followed again by comparing values, and giving feedback, continuing the process until the end of the lap is reached, where the system loops around, or it is ended by the user.

7.3. Limitations

The biggest current limitation of the coach is represented by its static nature. The system can only give feedback on the current state compared to the optimum one; it does not dynamically change the optimal trajectory based on what the driver is currently doing. For example, the optimal lean angle is shown for optimal speed, but if the driver is driving much slower than that, it is not possible to achieve that number. Of course, the driver also has access to the difference in speed compared to the optimal one, but it is impossible to expect him to also take it into consideration while he’s concentrating on driving. Due to the nature of the entire project as a whole, changing it would also mean radical changes to the way simulations are made.

Another limitation is the lack of incorporation possible with the microcontroller. Due to the client not having current access to the microcontroller or the bike, it was not possible to integrate the system with the current setup that they have. As a result, the client is aware that testing on the microcontroller was not possible and that managing sensor reads and the way feedback is given could not be implemented at this stage and is planned for future integration.

Chapter 8. Test Plan

8.1. Testing Approach

The main focus of our testing plan was on functionality testing. Due to the complex nature of the project and lack of pre-existing available data to compare with, complete validation of the accuracy was not possible. The testing plan included manual unit tests to verify individual components, integration tests to see if these components work correctly together, and finally, system tests to validate the entire system.

The testing schedule was left to the discretion of the implementer. If the feature passed unit testing and integration testing for the immediate components it affects, it would be pushed to the review board on Trello, for the other members to test themselves before eventually being approved. Finally, a large-scale system test was conducted to catch any possible bugs and validate the system functionalities against the initial requirements.

8.2. Tested functionalities

Tested functionality	Level of risk
Model Predictive Control	
Initial MPC testing (very simple formulas to see if it would take steps)	H
Testing MPC with multiple formulas	H
Testing MPC with a dynamic cost function	M
Testing MPC interaction through the web-app	H
Web-app	
Testing vehicles page (adding, removing vehicles)	M
Testing specifications page (adding/removing/updating inputs, variables, tracks)	H

Testing the specified inputs and variables affects the simulation	H
Testing simulation page (testing starting, stopping, resetting simulation; testing visualization of the simulation)	H
Testing past runs page (observe/add/remove/download previous runs)	M
Testing the conversion of the .npy file to the .h file	H
Live Coach	
Testing find_closest_point to localise bike location	H
Testing get_next_corner_target to get the target lean angle for the next corner	H
Testing the calculations of lean angles based on the trajectory	H
Testing the reading of .csv files to simulate live sensor reading	M
Testing run_comparison(), the main logic for generating feedback	H

8.3. Test Results

8.3.1. Simulation

Simulation testing was done through unit tests. This was done by running the simulation and observing if the change made a difference and if this was ideal for the simulation. One of the initial problems with the simulation was that it was far too simple to produce a realistic physics simulation; to combat this, we allowed the client to specify their own constraints for the simulation. In this way, the client could decide for themselves how realistic the simulation should be. Another issue that was faced was that the simulation was extremely slow, and we still wanted

to add complexity. This was resolved by using PyTorch and running the simulation on a GPU, rather than a CPU. This allowed us to add to the complexity of the simulation while still making it possible to test efficiently.

Integration testing was also done with the web-app once both were completed. This testing included the creation and updating of inputs and variables, and comparing the differences seen in the simulation. There were no issues seen in the testing of the integration.

8.3.2. Web-app

Throughout the testing of the web-app, there were many small issues that were observed. These include the creation of variables with formulas in LaTeX format, and largely visualization issues for LaTeX formatting, and displaying the simulation. In creating variables in a LaTeX format, we initially used a different package, which was not displaying and causing issues; this was later fixed by using a different package. This was also the case for the other visualizations of the LaTeX formulas for the variables. Lastly, there were issues with visualizing the simulation on a real-world map. To do this, we initially wanted to use the Google Maps/Earth API for displaying the simulation. The main issue with this was that it cost money to use this API. To overcome this, we use “Leaflet”, a JavaScript library that uses OpenStreetMap, which is a free mapping tool.

Testing was generally done in an integration testing fashion. The web-app was tested by creating a feature in the back-end, then the corresponding feature in the front-end, and then testing the integration of these two. This often worked for smaller features; however, some initial features were tested differently. To initially test the database, unit tests were written with fake data to see if the database would receive and store the data correctly.

8.3.3. Driving coach

During testing of the live coach, several small issues were identified and corrected, mainly reading and parsing the .csv file, but also not correctly initiating the feedback loop due to not recognizing the current bike position if it was not at the beginning of the lap during initialization. This was remedied by scanning the entire track coordinates first, so the system can work regardless of where the biker finds himself. Another identified issue during the testing stage

showed that when searching for the optimal lean angle for an upcoming corner, a local peak value must be used instead of the maximum, which fixed the issue of only showing the corners with the most lean needed and ignoring the shallower ones in between.

Chapter 9. Manual

9.1. Installation

To install the project, the first step is to pull from GitLab. This will contain files from both the web-app and live driving coach. The web-app runs on a file called app.py in the directory/folder src/simulator/app.py. The live driving coach is found in the /src/coach directory. To run the web-app, it is useful to create a virtual environment and then install the requirements in requirements.txt. To run the web-app, one must run the command:

python -m src.simulator.app

This command should be run from the main directory. To run the live driving coach, the user has to compile the file and run it (more information in Section 9.3).

9.2. Web-app

The top right corner of the web-app contains 4 navigation tabs: **Vehicles**, **Specifications**, **Simulation**, and **Runs**. By default, the Vehicles tab is the first tab upon entering.

9.2.1. Vehicles Tab

This is the primary workspace for managing active bike profiles, creating or removing new instances of vehicles, and selecting existing instances of vehicles.

Creating a new instance of a vehicle

1. Click the red **+ (Plus)** button located next to the search bar at the top of the table.
2. This will open the **Create New Vehicle** box to prompt users to enter the name of the new instance.
3. Once it's done, click **Create** at the bottom right corner in the same box. By clicking **Create,** users will be taken to the specifications tab of that newly created vehicle immediately.

Deleting an instance of an existing vehicle

1. On the main **Vehicle** tab, click the **Remove** button in the same row as the vehicle that needs to be deleted.

2. The warning box will pop up on top of the page to confirm the action. After clicking **OK,** the vehicle will be removed.

Sorting vehicles

Sorting offers 2 sorting options: **newest** or **oldest**. The sorting is located in the top right corner of the vehicles table.

Searching for any vehicle

To search for any vehicle, use the search bar located on the left of the **+ (Add)** button.

Select the vehicle

To adjust the technical specification of the vehicle, right-click on the **Select** button of that vehicle.

9.2.2. Specifications Tab

The **Specifications Tab** contains more vehicle-focused functionalities like modification of control inputs and variables, track upload, or removal.

Control Inputs

The Control Inputs panel is on the left. The Control Inputs are behaviors or signals that come directly from the vehicle itself, such as throttle, brake, and steering angle. The Control Inputs contain 3 primary fields: Name, Min Value, and Max Value.

- **Name**: identifier for this input. This name will then be used as a symbol in the variable equation.
- **Min Value**: the lowest value this input can take
- **Max Value**: the highest value this input can take

Adding a Control Input

1. Fill in the Name, Min Value, and Max Value fields in the blank form located at the top of the Control Inputs box.

a. Click the **Save Input** red button underneath the fields. The new input will appear as a card below.

Removing a Control Input

1. Click the **- (Minus)** red button located at the top right corner of that control input.
2. A pop-up card will appear on top of the screen. Click **OK** to confirm the action.

Updating a Control Input

1. Adjust the **Min Value** or **Max Value** of the Control Input that needs to be edited.
2. Click the **Update Input** button underneath to confirm the actions. A pop-up banner will appear on top of the screen to confirm the action is successful.

Variables

Variables are behaviors derived from control inputs and/or other variables through a mathematical equation. The Variable panel contains several fields:

- **Name**: the identifier for this variable
- **Initial Value**: the value this variable starts at
- **Min Value**: the lower bound
- **Max Value**: the higher bound
- **Equation**: The mathematical formula used to compute this variable's value
- **Segmented Control**: define the function of this variable, reward or cost
- **Direction**: How the variable affects the vehicle's "score", direction tells the simulation if it should reach for higher (ascending) or lower (descending) values for that variable
- **Aim Sector**: multiplier used to scale this variable when applying it to the MPC model

Writing Equations

Equations should be written in LaTeX notation. The entries can be referenced as follows:

- **Control Inputs** by their name, prefixed by backslash. For example, \throttle, \brake.
- **Other variables** by their symbol. For example, a for acceleration, v for velocity.

Adding a new variable

1. Fill in all the required fields in the Variable form located at the top of the Variable panel. The Equation is filled using **the LaTeX Equation Editor** located in the top right corner of the page.
2. [relation between segmented control and direction]
3. Click the red **Save Variable** button underneath the field sections. The new input will appear as a new card below

Removing an existing variable

1. Click the **- (Minus)** red button located at the top right corner of the variable card.
2. A pop-up banner will appear on top of the screen. Click **OK** to confirm the action.

Modifying a variable

1. Locate the variable that needs to be edited
2. Update any fields directly on the card
3. Click **Update Variable** to save the changes. A pop-up banner will appear on top of the screen to confirm that the changes are successful.

Track

A track file is a KML file that defines the physical boundaries of a race. The track file must be in **KLM format** (.kml). Other formats are not supported. Multiple tracks can be uploaded and stored simultaneously. The track panel is located in the top left corner of the site.

Adding a Track File

1. Click the **Add Track File** button
2. Select the .kml file from the file system
3. Once uploaded, the track will appear as a new entry showing its file name

Removing an existing Track File

1. Click **- (Minus)** red button on the right next to the track name
2. A pop-up banner will appear on top of the screen. Click **OK** to confirm the action

9.2.3. Simulations Tab

The Simulation is the main interface for running and reviewing the MPC simulation. This page comprises of 2 main components: **Optimal Path** on the left and **Telemetry** on the right.

Optimal Path

The toolbar at the top of the panel contains all the parameters needed to run the simulation. From left to right:

- **Track file dropdown**: selects which uploaded track to simulate on (.klm)
- **Latitude**: latitude of the simulation start position on the track
- **Longitude**: longitude of the simulation start position on the track
- **MPC Horizon (steps)**: the maximum number of simulation timesteps. A higher value allows the vehicle to travel further, but increases computation time.
- **Time Step (s)**: duration of each step in seconds
- **Clockwise toggle**: set the direction of the vehicle around the track

Running the simulation

1. Ensure the vehicle is configured fully and accurately
2. Select the desired track in the track file dropdown
3. Set the desired latitude and longitude of the start position; otherwise, the default latitude and longitude will be applied
4. Configure MPC Horizon, Time Step, and Clockwise direction if needed
5. Click the Run **Simulation** button.

Resetting the simulation

By clicking **Reset** during the simulation, the simulation will return to the initial state. This will stop the simulation and remove the progress of the simulation.

Downloading the simulation

The **Download** option is disabled during the simulation run. To download the track trajectory, wait until the simulation is completed or pause the simulation. The download returns the .h header file with the optimal trajectory to be used for the live coach.

Telemetry

The telemetry panel is located on the right, displaying the performance data of the simulation. It remains empty until the simulation has been run for at least 5 iterations.

Chart

- **Speed vs. Time:** Shows the velocity vs time of the simulation
- **Acceleration vs. Time:** Shows the acceleration vs time of the simulation

Telemetry Summary

- **Peak Speed:** The top speed in km/h of the simulation
- **Avg Speed:** Average speed of simulation in km/h
- **Distance:** The total distance of the motorcycle, in meters
- **Peak Accel:** The peak acceleration of the simulation in m/s^2
- **Peak Brake:** Peak braking in m/s^2 of simulation (negative acceleration)
- **Duration:** Time in seconds of how long the simulation has taken in simulation time

9.2.4. Runs

This tab contains a historical log of all simulations that have been executed. The page offers features like browsing, filtering, downloading, and deleting past simulation records.

Summary Statistics

At the top of the page, there are 4 summary cards intended to provide a quick overview of the simulation history.

- **Total Runs**: The total number of simulations
- **Unique Vehicles**: The number of distinct instances of vehicles
- **Unique Track**: The number of distinct track files that have been run
- **Latest Run**: The time elapsed since the most recent simulation was completed

All Simulation Runs Table

The main table lists every saved simulation run.

Filtering and Searching Vehicles

Search by typing any text into the search bar to filter runs by vehicle name or track name.

All Vehicle dropdown filters existing vehicles by vehicle name. Select a vehicle name from the dropdown, or leave it as All Vehicles to show all run instances.

All Track dropdown filters existing tracks by track name. Select a track name from the dropdown, or leave it as All Tracks to show all run instances.

Downloading a Run

1. Locate the run of choice
2. Click the **.h** button. This downloads the full trajectory data for that run as a .h header file.

Deleting an existing Run

1. Click the **red trash icon** of the run instance that needs to be removed.
2. A pop-up banner will appear to confirm the action. Click **OK** to remove the run instance.

9.3. Live Driving Coach

The Live Driving Coach comes in two files. The first file is configured to allow testing of the live coach by simulating sensor readings using data from a CSV file. The second file is similar to the first, but instead of simulating sensor readings, it leaves the implementation blank for future integration. For any of the live coach files to work, they need to be compiled with the downloaded trajectory from the web-app.

Chapter 10. Future Improvements

To build on our current progress, further work on this project would include: deployment on the real bike, more realistic MPC simulation, a more secure web-app, and more general improvements to user experience. The main improvement for the project would be to really test the system on the client's motorcycle. This would add to the validity and allow for true testing of our project. This was, however, not feasible during the project due to multiple factors. These factors include: the client not having the bike made yet, the client not having the microcontroller for us to test on, and nowhere to test the system, even if the bike were prepared. The client was to create the bike in the following years and thus was not yet prepared for us to perform our testing on it. Secondly, we did not have the microcontroller to test our implementation of the live bike coach, although even if we had the microcontroller, there would not be any data for us to use on it. Furthermore, the client wanted to create their own method of informing the driver how to take each corner, and thus, this was not necessary for us to implement.

Secondly, a more realistic MPC simulation would be in order to provide an improved driving coach. A more realistic MPC would be in 3D, and while this should be possible, it would take far more processing power. Additionally, the realism would only be slightly affected since the changes in elevation on the tracks are not substantial. While our current simulation is simplistic, our product allows a user to add more constraints and make the bike behave in a more realistic way. This includes adding friction dynamics and drag, but also other physical limitations of racing motorcycles.

Lastly, the web-app could have some improvements. These mainly include additional security and more general user experience improvements. As mentioned previously (Section 5.2), security was not a large concern within the project due to the assumption that the client would run the web-app locally, and the main threat would be attackers who are physically close enough to connect to the local hosting. While this is acceptable for now, we do believe in a future when this product could be hosted on a real website for multiple users and open to worldwide attackers. To ensure the safety of the product, users should be able to log in and see their own vehicle profiles. Furthermore, connections to the back-end should be sanitized. Ultimately, there are many attacks that would need to be prevented if the application were remotely hosted. On the

other hand, the user experience could be improved, which includes more information on how to use the web-app, and running the simulation when multiple users are on the web-app. While the majority of the web-app is reasonably self-explanatory, there are aspects (such as adding a new variable) that can be difficult to understand if seen for the first time. Currently, when hosting the web-app, only one simulation can be run at a time. While this is fine for the current situation of the client, if the web-app expands, multiple simulations should be runnable at the same time or have a lock so that other users cannot stop the simulation.

Chapter 11. Evaluation

11.1. Planning

After deciding each member's responsibility, everyone proceeded to fulfil their assigned responsibilities, which are detailed in Section 4.2. The internal communication was carried out on WhatsApp. The team also had a separate group with clients for any urgent inquiries. In addition, a meeting with the supervisor was carried out every week to report the progress. Similarly, the team met clients on a weekly basis to review the advancement and to request feedback. The group would meet every week to discuss further solutions or the existing issues a member was struggling with. There were no strict deadlines, but everyone was fully aware of their tasks and responsibilities. This ensured the entire project stayed on track.

11.2. Team Evaluation

In the first 2 weeks, the team experienced a lack of communication as some members had never worked with the rest, and the group was not fully integrated. Upon recognising it, the team immediately resolved it, and the communication was enhanced noticeably afterwards, as well as the workflow. In the occurrence of questions, members often asked their questions on WhatsApp or during the group meeting. During the project period, Maja always actively reminded other members of deadlines, coordinated the group's structure, and communicated with the supervisor and clients on behalf of the team. On the other hand, Bartek was mostly in charge of the technical coordination of the project.

11.3. Final Result

In the end, the goal was met with the successful implementation of a functional web-app and onboard live driving coach. The web-app operates as intended with all initial desired features fully implemented. The UI design is intuitive, and all features are easy to navigate. However, the simulation takes a long while to finish due to high computational complexity. Although the live driving coach could not be tested on the microcontroller, we believe it should not raise any problems, as its computational power is relatively high compared to the workload. The team is not responsible for the integration of the onboard live driving coach to the superbike, but this will be carried out by clients who have more expertise. The current design is a clear indication

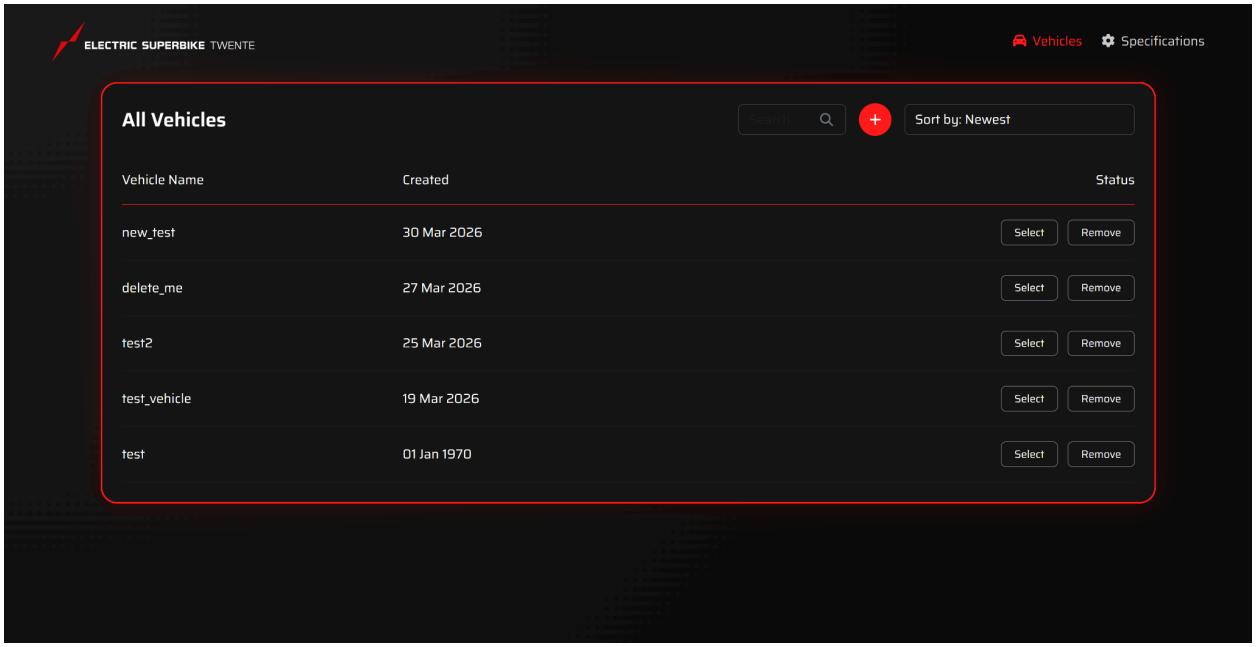
that the majority of the proposed requirements are met, except for a few, due to a lack of access to certain components (please refer to Chapter 3 for more details).

Appendices

A. Mock-ups

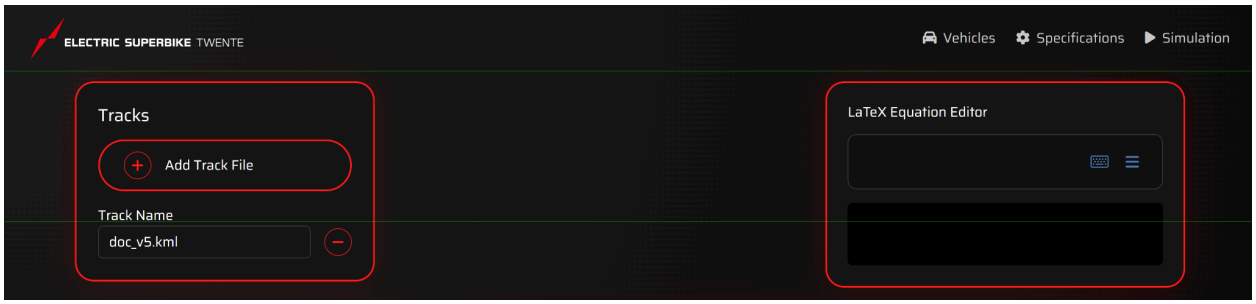
1. Vehicles Page

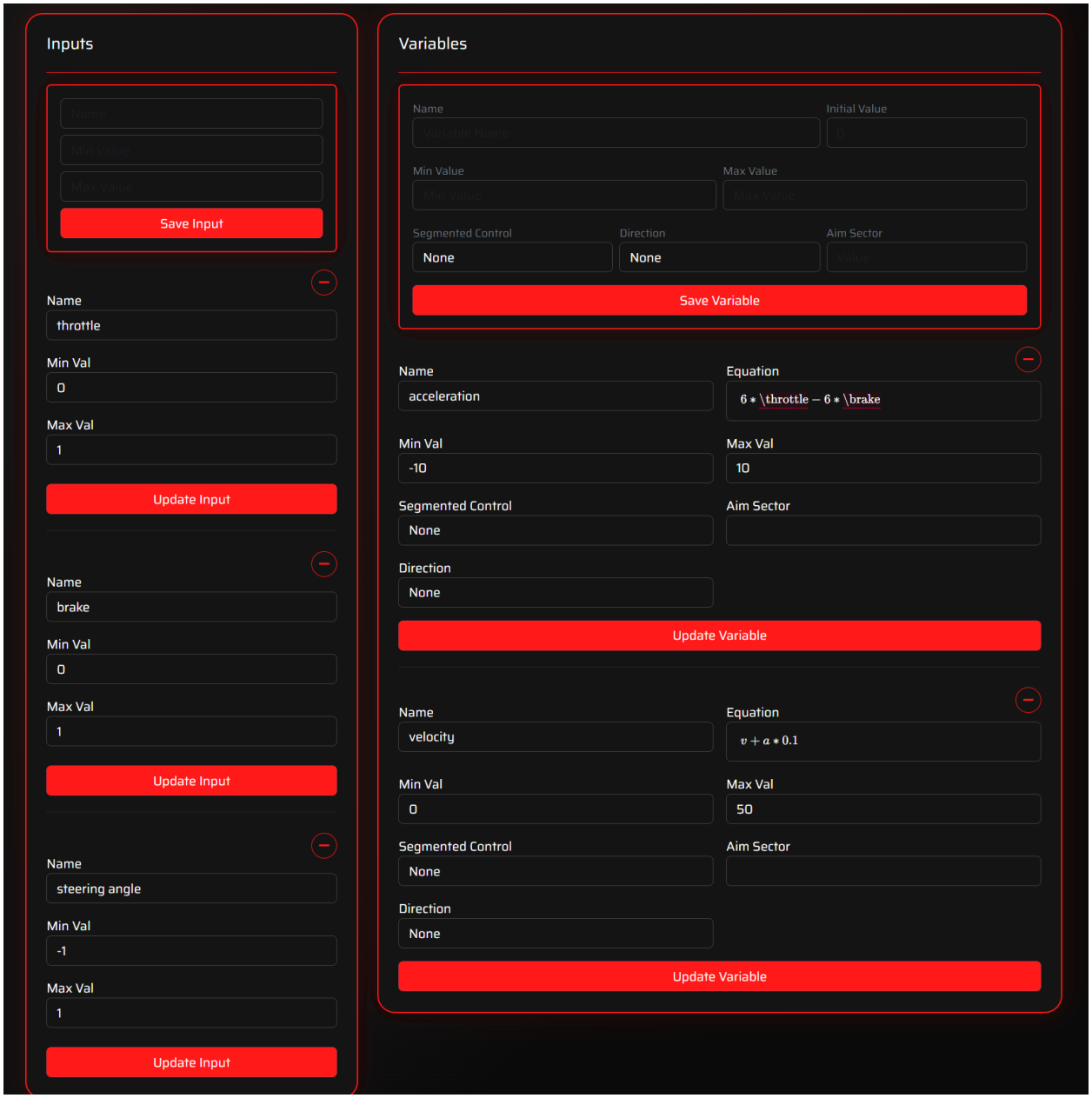
The Vehicles page serves as an overview of all the vehicle profiles defined by the user. It shows a sorted list of all available profiles, including their names and creation dates, which allows users to quickly select a specific setup. The interface is designed in a clean and easy-to-navigate way with a search bar and sorting button. The user can easily select, add, and remove a vehicle directly from this page.



2. Vehicle Specification

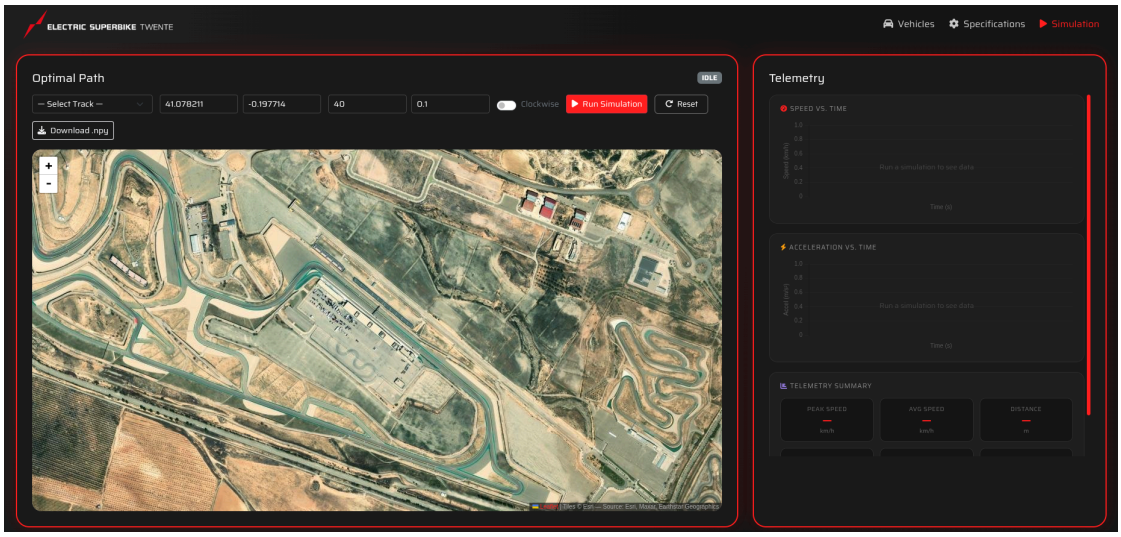
The specifications page provides a laboratory for the user to customise each vehicle profile using Variables, Inputs, and formulas. It allows users to upload and manage track files for the optimisation process. The users can define control inputs like throttle, brake, and steering as well as their bounds, which aids in the modelling of the simulation. An important feature is the ability to create and customise system variables. We allow the users to specify equations that describe the relationships between inputs and system dynamics using the LaTeX editor.


ELECTRIC SUPERBIKE TWENTE
Vehicles
Specifications
Simulation
Tracks
Add Track File
Track Name
doc_v5.kml
LaTeX Equation Editor


Inputs
Name
Min Value
Max Value
Save Input
Name
throttle
Min Val
0
Max Val
1
Update Input
Name
brake
Min Val
0
Max Val
1
Update Input
Name
steering angle
Min Val
-1
Max Val
1
Update Input
Variables
Name
Variable Name
Initial Value
0
Min Value
Max Value
Segmented Control
None
Direction
None
Aim Sector
Value
Save Variable
Name
acceleration
Equation
$6 * \throttle - 6 * \brake$
Min Val
-10
Max Val
10
Segmented Control
None
Aim Sector
Direction
None
Update Variable
Name
velocity
Equation
$v + a * 0.1$
Min Val
0
Max Val
50
Segmented Control
None
Aim Sector
Direction
None
Update Variable

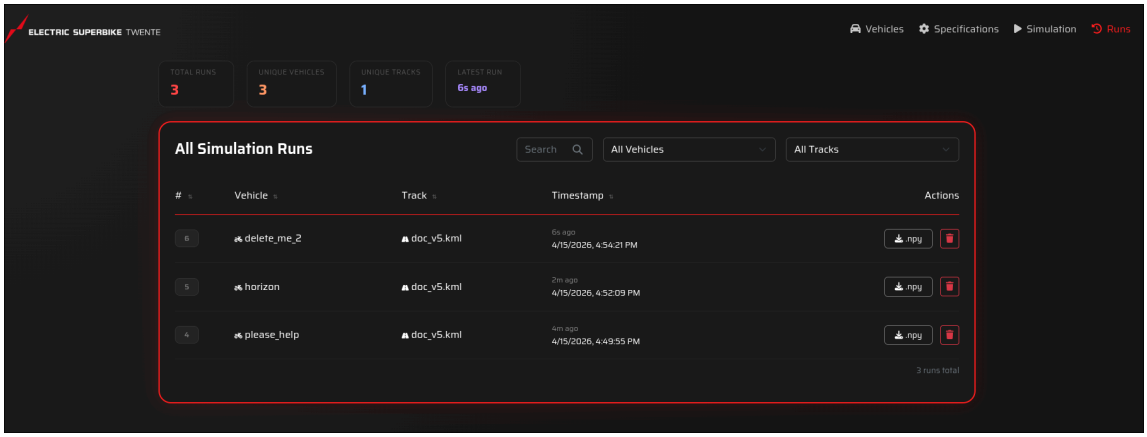
3. Simulation Page

The simulation page of the web-app shows an interface for visualizing and analyzing the performance of each vehicle profile. The left side shows a user-specified track on which the simulation plots the optimal racing path. To the right of the map, diagrams of time versus performance are displayed, including speed-versus-time and acceleration-versus-time graphs. The panel below provides the most important data, such as peak speed, average speed, and total distance covered.



4. Previous Runs Page

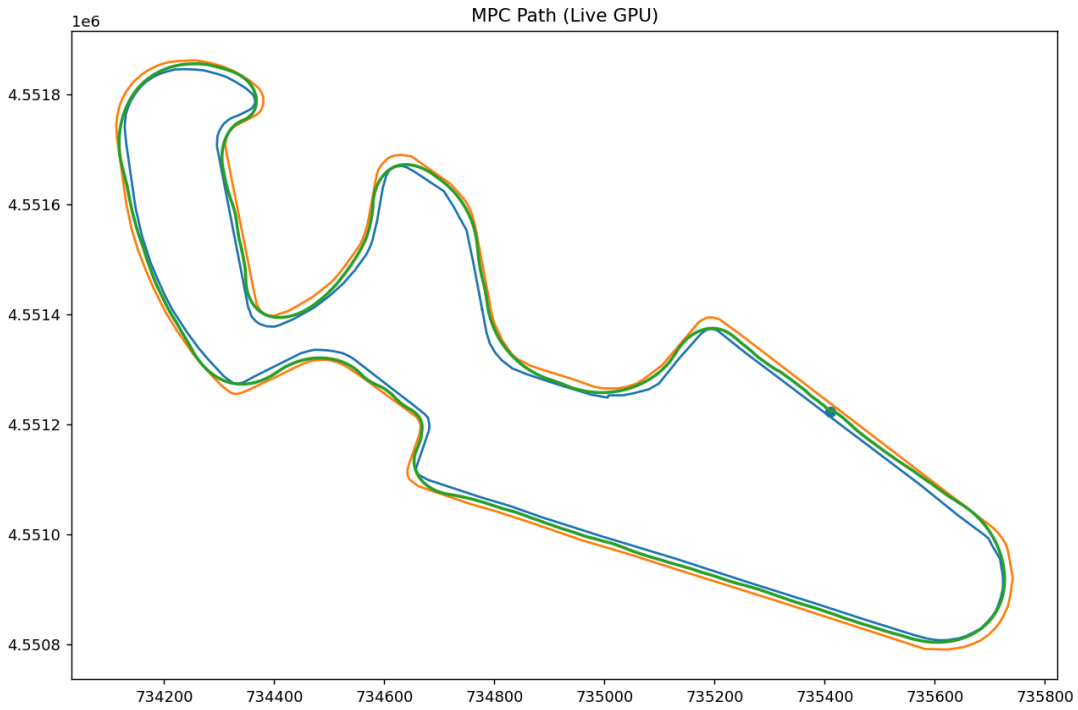
The previous runs page shows the previous runs from the simulation. These are stored in the database since they can take a while to run. These runs can be downloaded and deleted, but also searched for and filtered.



B. Simulations

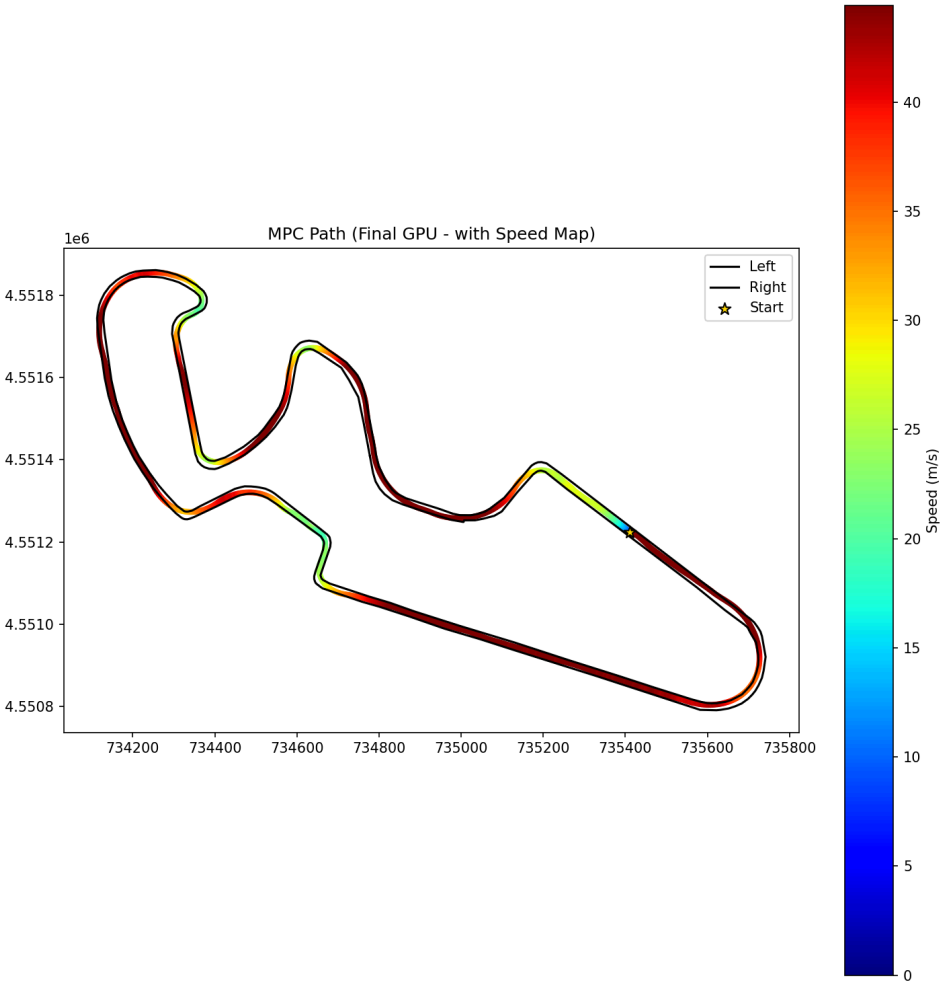
1. MPC Path

This figure shows the simulated optimal race path generated with (MPC) on a specified track. The green line represents the computed path, while the yellow and blue lines are track bounds.



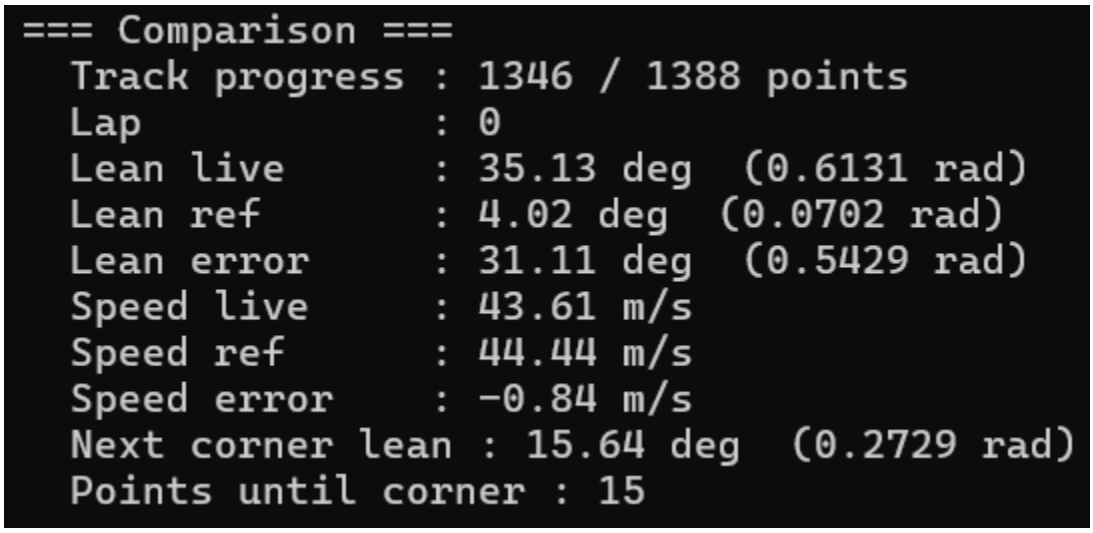
2. MPC Path with Speed Map

The figure below again shows the optimal path on the track using Model Predictive Control. The different coloured lines show the speed on the track, with dark red being the highest speed and dark blue the lowest.



C. Live Coach

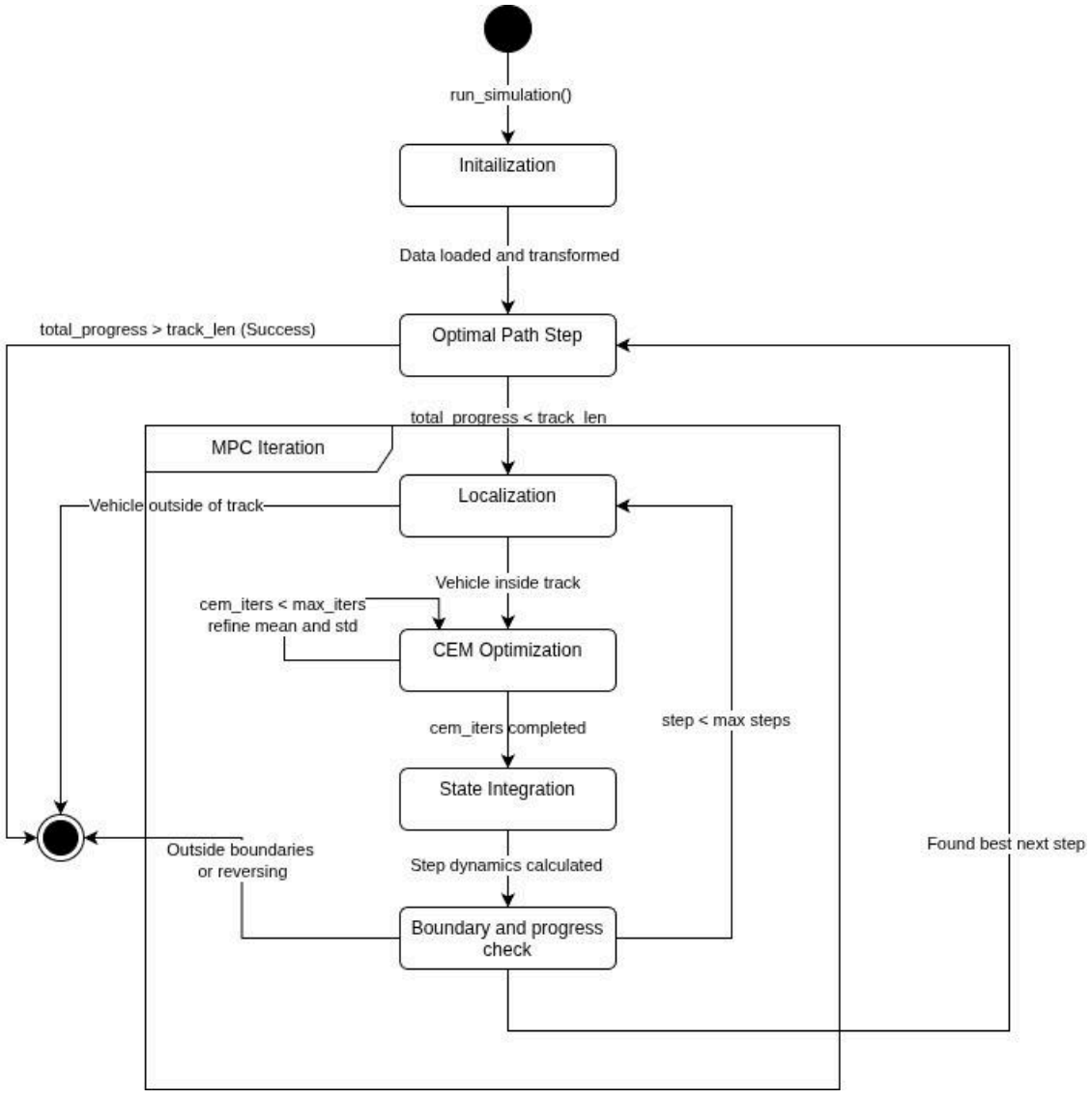
The image below shows all of the contents of the DriverFeedback struct and a basic way to display the feedback.



D. Diagrams

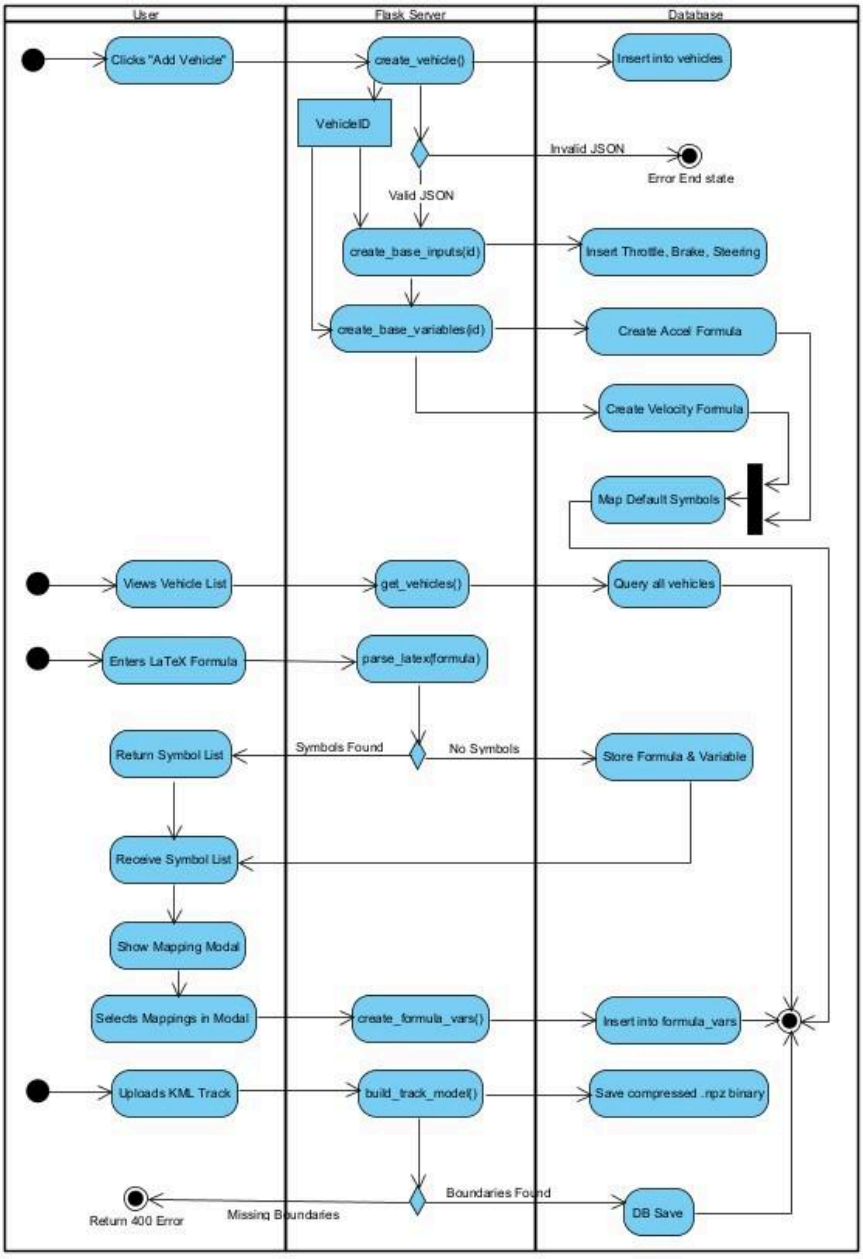
1. State Machine Diagram

The diagram below shows the process that the MPC simulation takes in order to produce our simulation. The MPC has two main loops, the first of which checks if the total progress is less than the length of the track. If it is, then we are able to move inside the MPC iterations. Within the MPC iteration, we localize the current position of the vehicle. If this is outside of the track, then we have failed, and the MPC halts. If the vehicle is inside the track, then we begin the CEM optimization to find the possible best paths. Once a best path is found, then we move one step forward, and loop back to the localization, until the number of steps has reached the horizon.



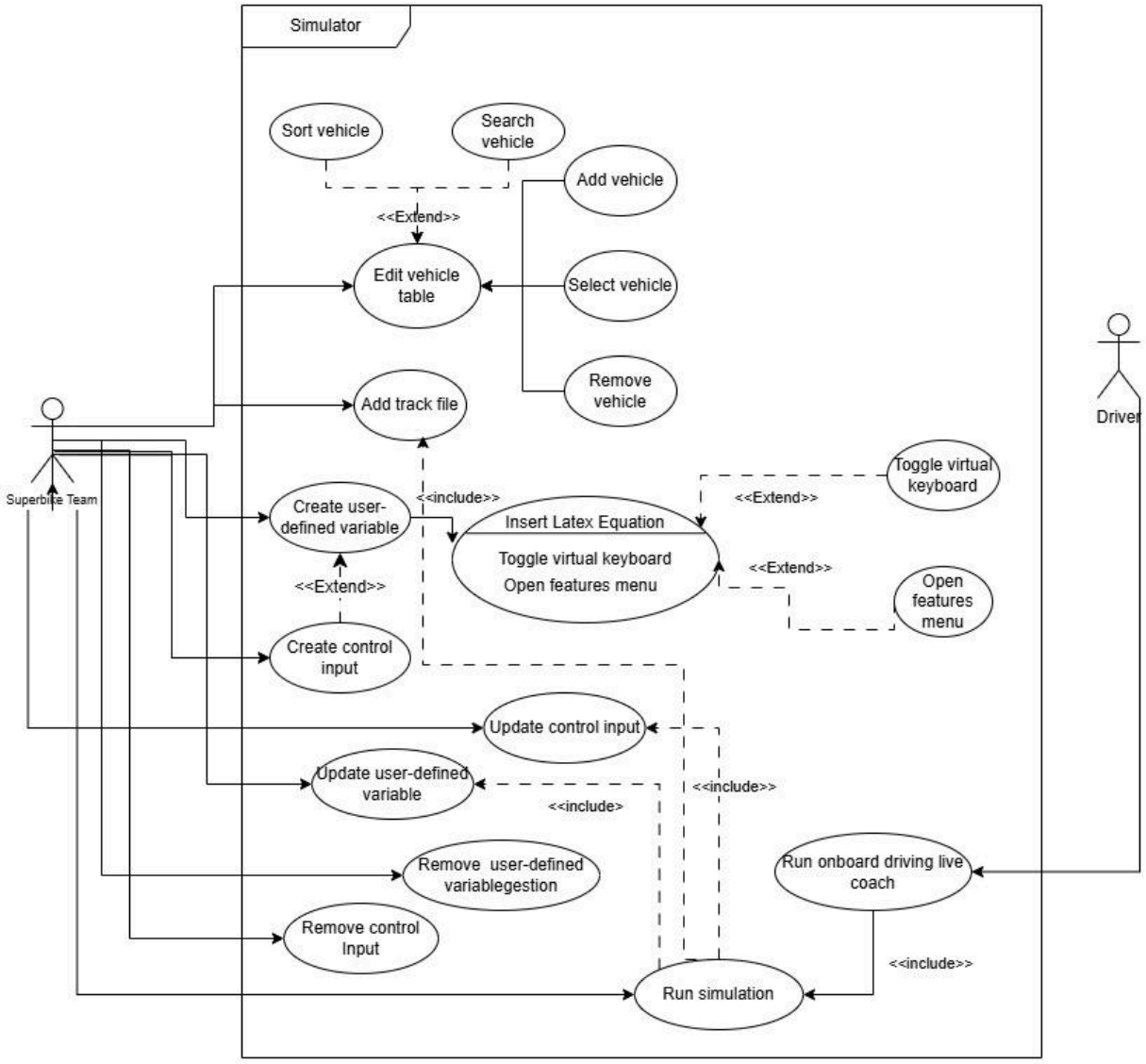
2. Activity Diagram

This activity diagram shows the workflow of the simulation application with interactions between the User, Flask Server, and PostgreSQL Database. It focuses on the most important processes, like the initial vehicle creation, in which the system automatically generates base inputs and corresponding formulas. The diagram also shows the variable creation: first parsing LaTeX, and then requiring user-driven mapping to link those symbols to specific inputs or variables. It also includes other processes like uploading track files.



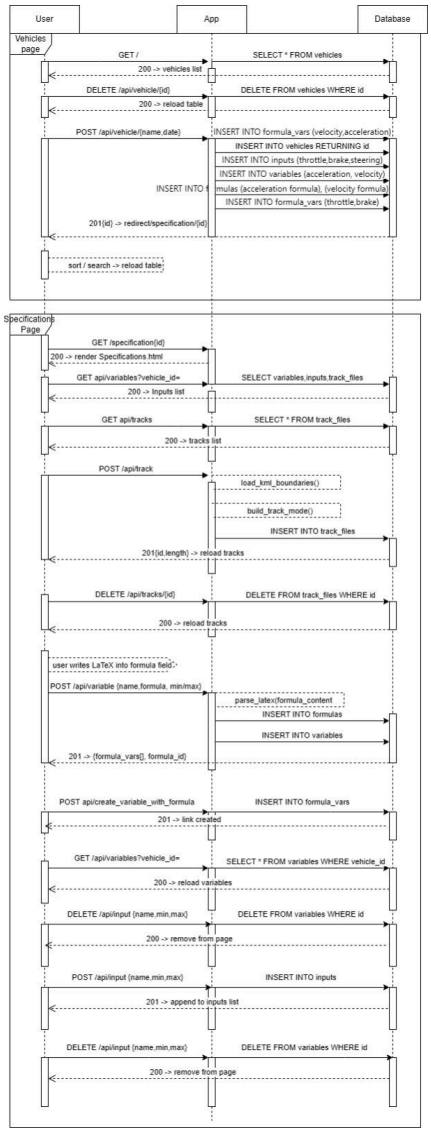
3. Use Case Diagram

The use case diagram illustrates all possible interactions between 2 actors: Superbike Team and Driver. The Superberbike Team acts as the technical administrator responsible for vehicle data management, such as adding or removing instances of vehicles in the database, and parameter configuration, such as creating and updating control inputs. The diagram also illustrates the relation between the run simulation and the parameters. The simulation strictly relies on inputs, variables, and the uploaded track file. While the Driver only focuses on the real-time execution of the onboard Driving Live Coach.



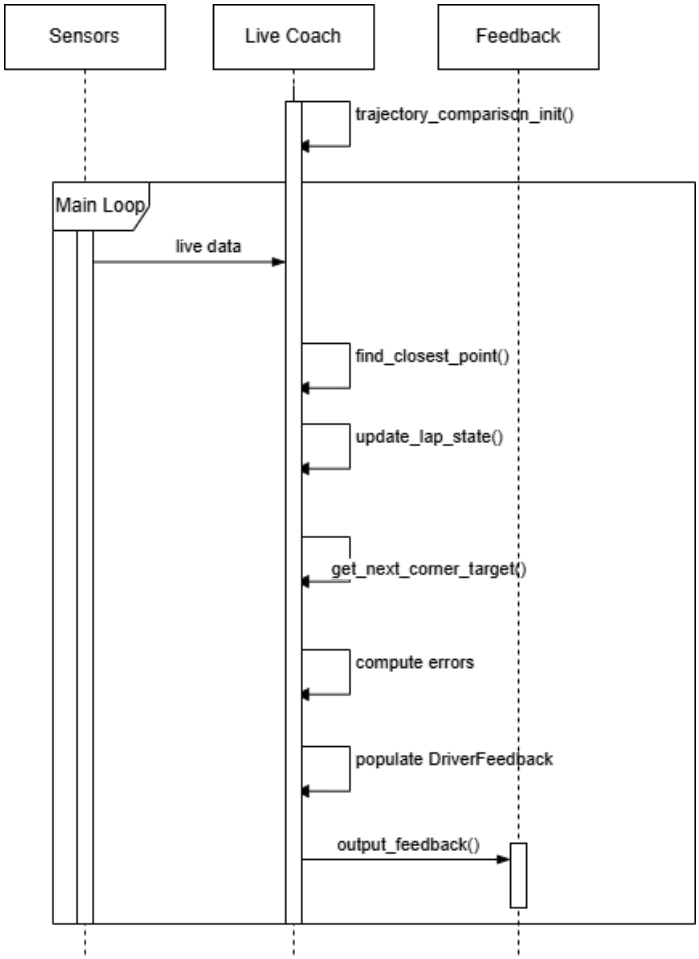
4. Sequence Diagram

This sequence diagram displays the dynamic between the user interface, application layer, and the database for the Vehicles and Specifications pages. It shows all of the actions involving creation, deletion, and viewing of the simulation records. These actions trigger HTTP requests, after which the application processes by executing SQL operations. Actions happening only at the application layer are also shown, such as uploading and preprocessing of track files and LaTeX formulas, which will be used during simulation.



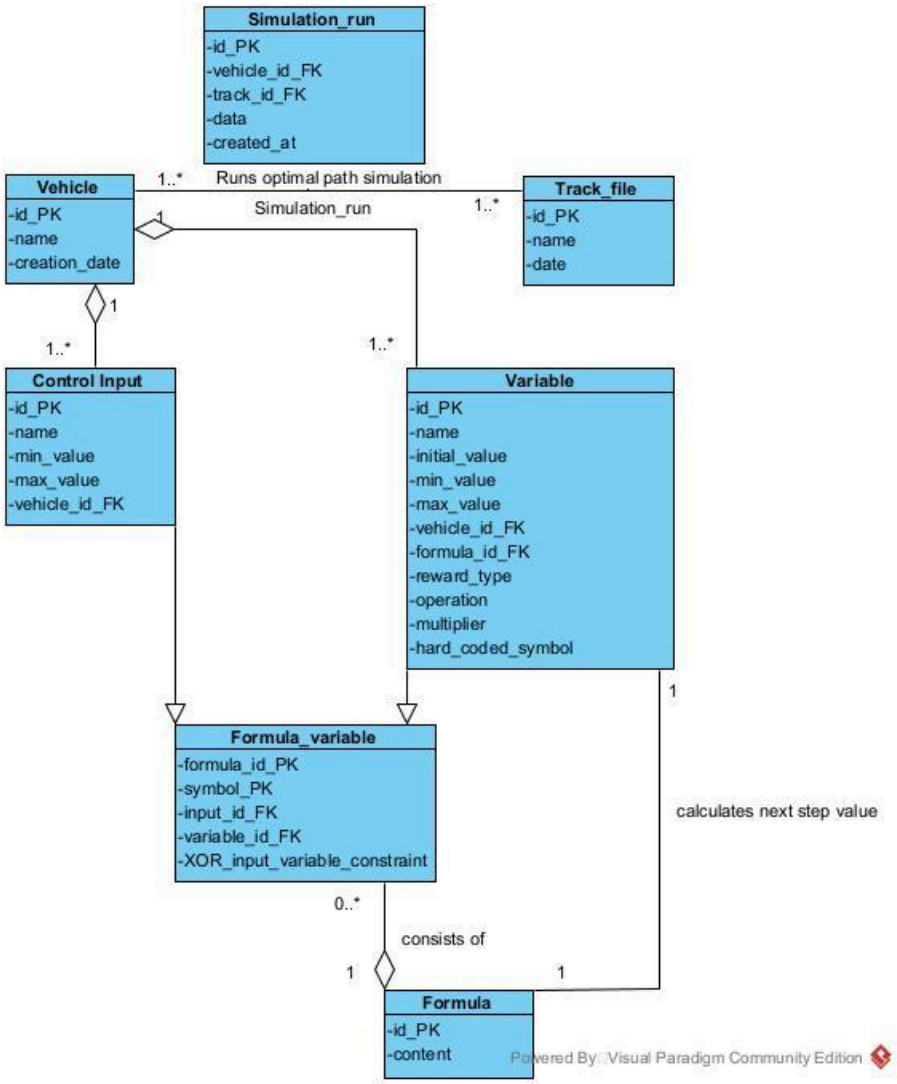
5. Live Coach Sequence Diagram

The sequence diagram for the live coach shows the order of the processes that give feedback to the driver. The initialisation can be used at the start of the session, or after the coach has been temporarily stopped (e.g., when exiting the pit lane). After the initialisation, the loop begins, and data from the bike sensors is collected. The first action the coach does is to find the closest point in the optimal trajectory compared to the real lat/lon coordinates and sets it as the current location. The system then takes the optimal values regarding speed and lean angle for that location. get_next_corner_target() checks the peak lean angle for the upcoming corner and shows it until that point is reached, when the next one is calculated. Finally, the errors are computed, and the DriverFeedback struct is populated, ready to be given to the driver, ending an iteration of the loop.



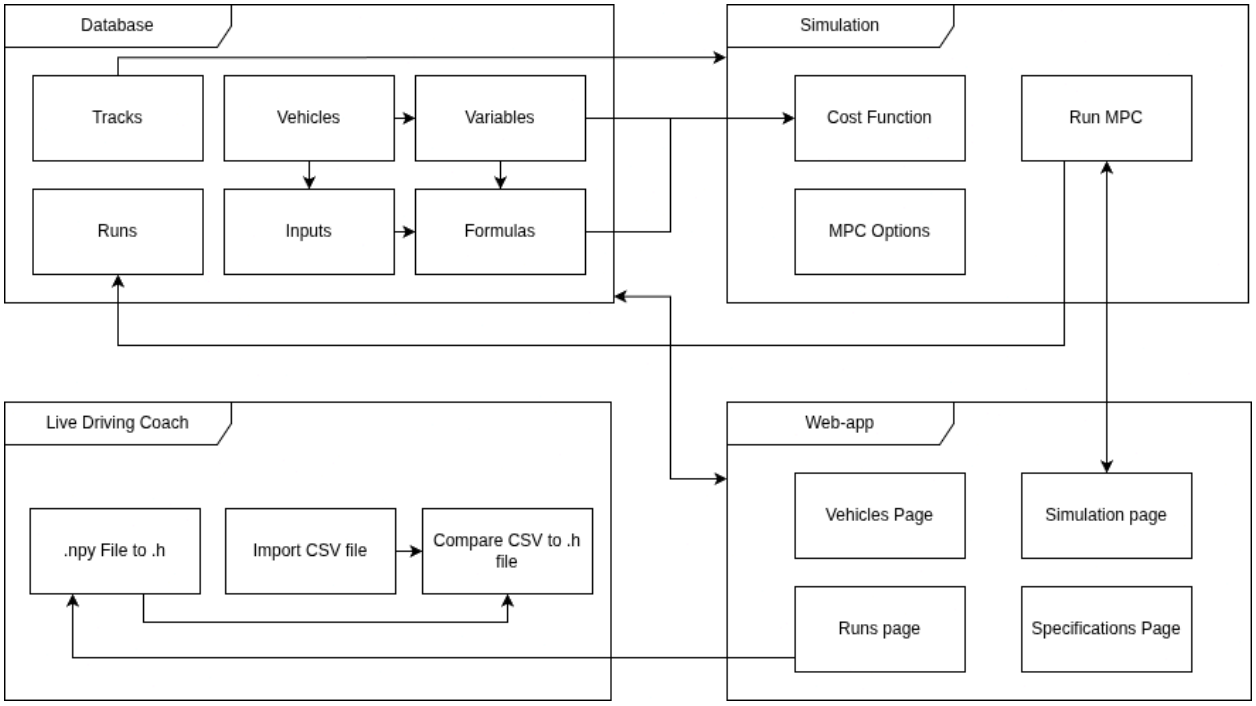
6. Class Diagram

The class diagram portrays the structure of the database. The data stored in the database is used for parsing the necessary formulas for the simulation. It consists of two main components, the vehicles and the track files, which together amount to a simulation run. The vehicle table acts as a connector to all of the important tables regarding the vehicle profile, namely the Control Inputs, the Variables, and their Formulas. The Formula_variable table connects Control Inputs and Variables to appropriate symbols within the Formulas.



7. Block Diagram for the whole system

The diagram below displays how the system works together. The large blocks consist of the database, simulation, web-app, and lastly the live driving coach. Within each block, there are smaller blocks which correspond to what is contained within the larger block. For the database this contains the relevant tables which are used (tracks, runs, vehicles, inputs, variables, and formulas), for the simulation there are the relevant functions (cost function, MPC options, and run MPC), the web-app has the pages for the frontend (vehicles, simulation, runs, and specification), and lastly the live driving coach has its own function (.npy to .h file, import csv, and compare csv to .h file). These blocks are connected based on their interactions within the project.



E. AI Statement

Throughout the project, some artificial intelligence (AI) was used. AI was predominantly used to refine the language for the report once it was written. Furthermore, limited amounts of AI were used for refining code to make it more readable and to add documentation to the code if there

were concerns for readability. All core decisions and implementation logic were developed by the project team.